Name: ____

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

CS 455 Final Exam
Fall 2010 [Bono]

Dec. 13, 2010

There are 6 problems on the exam, with 77 points total available. There are 7 pages to the exam, including this one; make sure you have all of them. If you need additional space to write any answers, you may use the backs of exam pages (just direct us to look there).

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.

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

Types used in problems 5 and 6:

```
struct Node {
  int data;
  Node * next;
  Node() { data = 0; next = NULL; }
  Node(int d) { data = d; next = NULL; }
  Node(int d, Node * n) { data = d; next = n; }
};

typedef Node * ListType;
```

---

	value	**score**
Problem 1	4 pts.	
Problem 2	12 pts.	
Problem 3	6 pts.	
Problem 4	20 pts.	
Problem 5	15 pts.	
Problem 6	20 pts.	
TOTAL	77 pts.	

Problem 1 [4 pts.]

Suppose we're doing hashing with keys of type `String`. Each of the following hash functions is bad. For each, explain what is wrong with it. Assume that your hash table is an array with indices 0 through `HASHSIZE`-1. Note: all of the functions return a value in the correct range and do not generate any run-time errors.

Hash function 1.

Assume that the string has at least 2 characters.

```
public static int hash(String key) {
   return (key.charAt(0)
           + key.charAt(1)
           + key.charAt(key.length()-1)) % HASHSIZE;
}
```

Hash function 2.

Reminder: `Random` is the class for generating random numbers.
`nextInt(n)` returns a random number in the range [`0,n-1`]

```
public static int hash(String key) {
   Random rand = new Random();
   int result = 0;
   for (int i = 0; i < key.length(); i++) {
      return result += rand.nextInt(37) * key.charAt(i);
   }
   return result % HASHSIZE;
}
```

Problem 2 [12 pts.]

Briefly describe two possible internal representations one might use for the Java `Map` interface. For each one, give the 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.

```
  // returns true iff entry with the key ‘key’ is in the map
  public boolean containsKey(KeyType key)

  // 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:

```
_____ containsKey      _____ get       ____ put        ____ remove
```

Description of representation 2:

Big-O times for representation 2:

```
_____ containsKey      _____ get       ____ put        ____ remove
```

Problem 3 [6 pts.]

This problem is about the Stack ADT and about differences in object semantics and parameter passing in Java and C++. In the following programs assume that both Java and C++ have a generic/template Stack class with the usual operations (push, pop, top, isEmpty) with the usual meanings**. Show the output for each of the following two programs that look roughly the "same" in C++ and Java.** Show your answer for a program to the right of that program.

```
// Java program
public class MainClass {
  public static void myFunc(Stack<Integer> s, Stack<Integer> t) {
    s.pop();
    t = s;
  }
  public static void main(String[] args) {
    Stack<Integer> s = new Stack<Integer>();
    s.push(3);
    s.push(4);

    Stack<Integer> t = new Stack<Integer>();
    t.push(9);
    t.push(7);
    System.out.println(s.top() + " " + t.top());

    myFunc(s, t);
    System.out.println(s.top() + " " + t.top());
  }
}
```

```
// C++ program
void myFunc(Stack<int> s, Stack<int> t) {
  s.pop();
  t = s;
}

int main() {
  Stack<int> s;
  s.push(3);
  s.push(4);

  Stack<int> t;
  t.push(9);
  t.push(7);
  cout << s.top() << " " << t.top() << endl;


  myFunc(s, t);
  cout << s.top() << " " << t.top() << endl;
}
```

Problem 4 [20 pts. total]

Implement the **Java** method `canBalance` which returns true if there is a place to split the given non-empty array so that the sum of the numbers on one side is equal to the sum of the numbers on the other side (There has to be at least one value on each side for it to balance.) Here are some examples:

`nums` array	`canBalance(nums)`
`[4, 5, 3, 3, 3]`	*`true`* (4+5 = 3+3+3)
`[6]`	*`false`*
`[5, 5]`	*`true`* (5 = 5)
`[5, 6]`	*`false`*
`[3, 2, 2, 4]`	*`false`*

```
// PRE: nums.length >= 1
public boolean canBalance(int[] nums) {
```

Problem 5 [15 pts.]

Implement the C++ function `isSorted` that returns true if the values in its linked list argument are in increasing order. The linked list types used are given on the first page of the exam.

Examples: (lists shown as space separated list of values surrounded by parentheses)

`list`	**return value of `isSorted(list)`**
(1 3 3 7)	*true*
(1 2 3)	*true*
()	*true*
(3)	*true*
(1 7 5 2 2 7 4)	*false*

```
// PRE: list is a well-formed linked list
bool isSorted(ListType list) {
```

Problem 6 [20 pts.]

Implement the C++ function `split` that splits its linked list argument, **`list`**, at position **`loc`** into two parts, **`part1`** and **`part2`**. **`list`** has the value **`NULL`** after the call. A loc <= 0 means the split is before the first element, and a loc >= the number of elements in the list means the split is after the last element. The linked list types used are given on the first page of the exam. For full credit your function must not create or destroy any nodes.

Examples: (lists shown as space separated list of values surrounded by parentheses)

list	**loc**	**part1**	**part2**
(2 6 7 5)	*1*	*(2)*	*(6 7 5)*
(2 6 7 5)	*0*	*()*	*(2 6 7 5)*
()	any	*()*	*()*
(2 6 7 5)	*3*	*(2 6 7)*	*(5)*
(2 6 7 5)	*100*	*(2 6 7 5)*	*()*

```
// PRE: list is a well-formed linked list
void split(ListType & list, int loc, ListType & part1, ListType & part2)
{
```