Name: _____

USC loginid (e.g., `ttrojan`): _____

CS 455 Final Exam
Fall 2013 [Bono]

December 12, 2013

There are 9 problems on the exam, with 64 points total available. There are 7 pages to the exam, including this one; make sure you have all of them. There is also a separate 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 USC loginid at the top of the exam. Please read over the whole test before beginning. Good luck!

	value	score
Problem 1-3	7 pts.	
Problem 4 - 5	6 pts.	
Problem 6	6 pts.	
Problem 7	5 pts.	
Problem 8	20 pts.	
Problem 9	20 pts.	
TOTAL	64 pts.	

Important: for each of the short-answer problems, please limit your answer to two sentences or less. We will only read the first two sentences of your answer.

Problem 1 [2 points]

Why would the following not be a great hash function for Strings? (Assume `HASH_SIZE` is the size of the hash table.)

```
// PRE: key.length() > 0
public static int hash(String key) {
   return (key.charAt(key.length()/2)) % HASH_SIZE;
}
```

Problem 2 [3 points]

Consider the following instance variable and representation invariant for a `Stack` class (elements are ints):

```
private ArrayList<Integer> data;
```

Reperesentation invariant:
-- `data` stores the elements of the stack
-- the arraylist is empty for an empty stack
-- for a non-empty stack top element is at the front of the arraylist (i.e., in `data.get(0)`).

Is this a good representation? [YES / NO] Why or why not? (2 sentences maximum)

Problem 3 [2 points]

Explain one benefit of implementing and testing a subset of a class (or program) before implementing the rest of the class (or program). (2 sentences maximum)

Problem 4 [4 points]

Consider a class, `Grade`, that represents a letter grade such as A+ or B. Give two choices of instance variables that can be used for implementing the class (show types and names). For each choice, briefly describe your instance variables (e.g., what restrictions on valid values).

Choice 1:

Choice 2:

Problem 5 [2 points]

Suppose the implementor of the `Grade` class from problem 4 changes from one representation to another, keeping the public interface unchanged. What do the programmers who use the `Grade` class need to do and why? (2 sentences maximum)

[space left here because of how the paging turned out]

Problem 6 [6 points]

Here is the complete interface, and some of the implementation for a *Java* class `Digits`, which allows you to access a number by its individual digits:

```
public class Digits {

   // create Digits version of num
   // PRE: num >= 0
   public Digits(int num) {
      originalNum = num;
      // . . . [ code to extract each digit and put it in digits ArrayList ]
   }

   // gets digit i of the number, such that digit 1 is the most significant
   //     digit, and digit numDigits() is the least significant digit
   // PRE: 1 <= i <= numDigits()
   public int getDigit(int i) { return digits.get(i-1); }

   // the number of digits in the number
   public int numDigits() { return digits.size(); }

   // returns the integer as a whole
   public int getInt() { return originalNum; }

   // return an arraylist of the digits (most significant digit is in position
   //    0, . . . , least significant digit is in position size()-1)
   public ArrayList<Integer> getDigits() {  return digits;   }

   private ArrayList<Integer> digits;
   private int originalNum;

   //  representation invariant:
   //    digits contains the individual digits of originalNum such that
   //          digits.get(0) has the most significant digit and
   //          digits.get(digits.size()-1) has the least significant digit

}
```

Is the `Digits` class immutable? If so, explain why, if not, give concrete code illustrating that it is not:

Problem 7 [5 points total]

Consider the following **C++** code sequence:

```
Node * p = new Node(3, new Node(7));
Node * q = new Node(5);
// A
Node * tmp = p->next;
p->next = q;
q = p;
p = tmp;
// B
```

Part A. Show a box-and-pointer diagram of the state of the variables at the point labeled A in a comment in the above code:

Part B. Show a box-and-pointer diagram of the state of the variables at the point labeled B in a comment in the above code.

Problem 8 [20 pts]

Write the *Java* function `shrink` which returns an array like the one given, but with every `k` adjacent elements replaced with the sum of those elements. If the number of elements in the array is not divisible by `k`, you just add up what's left over for the last element in the new array. Here are some examples:

`arr`	**`k`**	return value of **`shrink(arr, k)`**
[2,3,6,10,7,4]	*2*	*[5,16,11]*
[2,3,6,5,3,7,1,9]	*3*	*[11,15,10]*
[2,3,6,10,7]	*2*	*[5,16,7]*
[2,3,6,10,7]	*1*	*[2,3,6,10,7]*
[3]	*2*	*[3]*
[2 3]	*3*	*[5]*
[2 3]	*2*	*[5]*

```
// PRE: arr.length > 0 and k > 0
public static int[] shrink(int[] arr, int k) {
```

Problem 9 [20 pts]

Implement the *C*++ function `isSubset` which returns true if its second argument, `sub`, is a subset of its first argument, `list`. Both arguments are ordered linked lists (in increasing order) with no duplicates. For full credit your code must work in O(*m*+*n*) time, where *m* and *n* are the lengths of the two given lists. Hint: you can do this by taking advantage of the fact that the lists are ordered. The `Node` and `ListType` definitions are on the code handout.

`list`	**`sub`**	return value of **`isSubset(list, sub)`**	
(2 3 5 10)	*(3 6)*	*false*	
(2 3 5 10)	*(3 10)*	*true*	
(3 10)	*(2 3 5 10)*	*false*	
(1 7)	*()*	*true*	*[empty set is subset of every set]*
()	*(3)*	*false*	
()	*()*	*true*	*[empty set is subset of itself]*
(2 3 5 10)	*(2 3 5 10)*	*true*	*[any set is a subset of itself]*

```
// PRE: list and sub are a well-formed linked lists whose elements are
// in increasing order with no duplicates within a list.
boolean isSubset(ListType list, ListType sub) {
```