Name: _____

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

CS 455 Final Exam
Spring 2013 [Bono]

May 14, 2013

There are 5 problems on the exam, with 65 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 two-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 USC loginid at the top of the exam. Please read over the whole test before beginning. Good luck!

	value	score
Problem 1	16 pts.	
Problem 2	20 pts.	
Problem 3	8 pts.	
Problem 4	6 pts.	
Problem 5	15 pts.	
TOTAL	65 pts.	

Problem 1 [16 points]

Implement the *Java* class `Digits`, which allows you to access a number by its individual digits. Here is the complete interface:

```
public class Digits {

   // create Digits version of num
   // PRE: num >= 0
   public Digits(int num) { . . . }

   // 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) { . . . }

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

   // returns the integer as a whole
   public int getInt() { . . . }

   // . . .

}
```

Here's an example of using it:

```
Digits zero(0);
System.out.println(zero.numDigits() + "   " + zero.getDigit(1));
                                                        // prints: 1   0

Digits digits(5207);
System.out.println(digits.numDigits());                 // prints: 4
System.out.println(digits.getDigit(1) + "   " + digits.getDigit(3));
                                                        // prints: 5   0
System.out.println(digits.getInt());                    // prints: 5207
```

Hint: the modulus (`%`) and integer division operators will be useful.

Space for your answer is provided on the next page. To make it easier to refer back to the `Digits` interface and its comments, we also put it on the code handout.

Problem 1 (cont.)

```
public class Digits {
```

Problem 2 [20 pts]

Write the *Java* function `sortedSequenceLength` which returns the length of the *longest* sorted sequence in an array. For this problem a sorted sequence is a sequence of non-decreasing values. Here are some examples:

`arr`	return value of **`sortedSequenceLength(arr)`**
[-1, 2, 3, 3]	*4*
[-7 3 99 -10 0 0 43 10 20 30 5]	*4*
[3, 2, 1]	*1*
[32]	*1*
[]	*0* *(the only array for which we return 0)*

```
public static int sortedSequenceLength(int[] arr) {
```

Problem 3 [8 pts]

Implement the **Java** method `sortByDistance`, which sorts an `ArrayList` of points so they are in increasing order by distance from the origin. I.e., points closer to the origin appear on the list before farther points. *You must use the Java sort utility* (more info about that on the code handout), and include any additional code necessary to make it work (outside of the `sortByDistance` method).

The code handout also has more about the `Point` and `ArrayList` classes from the Java library.

Note: for the purposes of this problem you do not need to worry about floating point round-off errors.

```
public static void sortByDistance(ArrayList<Point> points) {
```

Problem 4 [6 pts. total]

Assume we are trying to compile the **C++** `grades` program we did for assignment 5. Here's a reminder of the file organization:

`Table.h` Header file for the `Table` class (contains `Table` class definition)

`Table.cpp` Implementation file for the `Table` class (contains `Table` method definitions)

`grades.cpp` A program that uses the `Table` class (contains `main`)

Consider the following `Makefile` for the program (compile commands are labeled at the right for your convenience):

```
grades: grades.o Table.o
      g++ -ggdb –Wall grades.o Table.o –o grades          (a)
Table.o: Table.cpp Table.h
      g++ -ggdb –Wall –c Table.cpp                        (b)
grades.o: grades.cpp Table.h
      g++ -ggdb –Wall –c grades.cpp                       (c)
```

Consider the following *sequence* of Unix commands below. I.e., assume they are done in the order shown with no other commands done in-between. **After each of the gmake commands, say what actions above, would be done by gmake, and in what order** (there may be more than one valid order). If no actions are done by that gmake call write "up to date". You may identify an action by its letter above (i.e., *a*, *b*, or *c*). Note: the Unix shell prompt is shown as a `%` below.

```
% ls
grades.cpp       Table.cpp       Table.h         Makefile          (output of ls)
% gmake Table.o
```

Actions done by this call to `gmake`:

```
% gmake grades
```

Actions done by this call to `gmake`:

```
% touch Table.cpp
```
(Note: touch changes the last-modified date. Alternatively you could edit the Table.cpp file and save a changed version.)

```
% gmake grades
```

Actions done by this call to `gmake`:

Problem 5 [15 pts]

Implement the *C++* function `reverseCopy` which returns a linked list that is a copy of the given list, but with all of the elements in reverse order from the original list. The original list is unchanged by the function. The `Node` and `ListType` definitions are on the code handout. Hint: you only need to traverse the list once.

`list`	return value of **`reverseCopy(list)`**
(1 2 3)	*(3 2 1)*
(7 6 7 5 8 2)	*(2 8 5 7 6 7)*
(4 2)	*(2 4)*
()	*()*
(3)	*(3)*

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