Name: _____

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

CS 455 Final Exam
Fall 2014 [Bono]

December 16, 2014

There are 6 problems on the exam, with 75 points total available. There are 9 pages to the exam, including this one; make sure you have all of them. There is also a separate one-page code handout. For on-campus students: 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 Net ID at the top of the exam. Please read over the whole test before beginning. Good luck!

	value	score
Problem 1	10 pts.	
Problem 2	9 pts.	
Problem 3	6 pts.	
Problem 4	15 pts.	
Problem 5	15 pts.	
Problem 6	20 pts.	
TOTAL	75 pts.	

Problem 1 [10 points]

The following ***Java*** code for insertion sort has a bug. (The comments will remind you how insertion sort is supposed to work.)

```
// sorts the array into increasing order using insertion sort algorithm

public static void insertionSort(int[] arr) {

   for (int i = 1; i < arr.length; i++) {

   // each outer loop iteration inserts arr[i] into the correct location of
   // the already-sorted sub-list: arr[0] through arr[i-1]

      int valueToInsert = arr[i];

      int loc = 0;

      while (loc < i && arr[loc] < valueToInsert) {  // find insertion location

         loc++;

      }

      for (int j = loc; j < i; j++) {          // make room for the new value

         arr[j+1] = arr[j];

      }

      arr[loc] = valueToInsert;
                              // put the value in correct location in sub-list

   // true here: arr[0] through arr[i] is a sorted sub-list

   }
}
```

Part A [3]. Show the results of calling the *current* version of the function on the following array:

`arr: [3, 9, 2]`

after call to `insertionSort(arr)`:

`arr:`

Part B [5]. Fix the code above. Do not rewrite the whole method, but rather make your changes right into the code above, using arrows to show where your new code should be inserted, crossing out code that you would get rid of, etc. The new version should do what the method comment says.

Part C [2]. What's the big-O worst case time for the code above?

Problem 2 [9 points]

Consider the following function that uses the **Java** Stack class:

```
public static void stackProb() {
   Stack<Integer> s = new Stack<Integer>();
   s.push(1);
   for (int i = 2; i <= 5; i++) {
      int val = s.peek();
      s.push(i);
      s.push(val * 2);
      System.out.print(s.pop() + " ");
   }
   System.out.println();
   // *
}
```

Part A. What is the output of the function?

Part B. Show the contents of the stack when the execution gets to the point labeled with an asterisk (i.e., contents after the loop). Put the contents of the stack in the box below, showing the top of the stack on the right:

Problem 3 [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 grades
```

Actions done by this call to `gmake`:

```
% gmake Table.o
```

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 Table.o
```

Actions done by this call to `gmake`:

Problem 4 [15 pts. total]

Part A [13]. Complete the implementation of the *Java* class `ElapsedTime`, which represents some amount of time in hours and minutes, such as 3 hours and 25 minutes (not a time of day). Such a class could be a useful building-block in an applications such as a timer, stopwatch, or scheduling programs (e.g., to schedule your DVR or flights for an airline).

The class allows us to create an elapsed time expressed using a number of hours and minutes or just total number of minutes, it also allows us to increase an elapsed time by some amount of time, to add two elapsed times together, and to convert an elapsed time to a String form (for printing or appending to longer messages). The detailed specification for each method appears in its method comments below.

Furthermore, **you are restricted to use the internal representation for `ElapsedTime` shown in the starter code for the class: internally we will store an elapsed time as its total number of minutes**. This representation actually would be one an implementer would likely use in a more fleshed out version of such a class because makes some operations easier to implement.

The class interface and space for your answer starts here and continues onto the next page.

```
/**
   ElapsedTime represents some amount of time in hours and minutes.
*/
public class ElapsedTime {

   // Problem restriction:
   //     do not change the private instance variable definitions

   private int totMinutes;  // representation invariant: totMinutes >= 0

   /**
      Creates elapsed time of the given hours and minutes
      PRE: hours >=0 and minutes >= 0
   */
   public ElapsedTime(int hours, int minutes) {









   }

   /**
     Creates elapsed time of the given total number of minutes.
     PRE: minutes >= 0
   */
   public ElapsedTime(int totalMinutes) {









   }
```

Problem 4 (cont.)

```
   /**
      Increase the current elapsed time by the amount of time
         given by hours and minutes.
      PRE: hours >=0 and minutes >= 0
   */
   public void increase(int hours, int minutes) {









   }

   /**
      Returns the sum of "this" elapsed time and elapsed time "b"
      This method is not a mutator (does not modify "this").
      Note the return type is an ElapsedTime object.
      HINT: Java rules allow this method to access private data of other
            ElapsedTime objects besides "this": for example, b's private data.
   */
   public ElapsedTime add(ElapsedTime b) {









   }

   /**
      Returns a String form of the elapsed time using the format shown by
        example here: "8hrs 32min"
   */
   public String toString() {









   }
}
```

Part B [2]. Are `ElapsedTime` objects immutable? If so why, if not, why not?

Problem 5 [15 pts]

[Java] Consider a different representation for our `Polynomial` class. The representation we will use here is a Java `HashMap` that maps exponents to their corresponding coefficients; i.e., it would use the interface type `Map<Double, Integer>` (the `Term` class does not figure into this representation). As before, we will have a representation invariant of only storing non-zero terms (one entry in the `Map` corresponds to one non-zero term in the polynomial). Thus, the zero polynomial would be represented as an empty map.

Using this `HashMap` representation write the new *Java* `Polynomial` method `addIn` which adds polynomial `b` to this polynomial (i.e., analogous to += on `int`s). Your implementation must run in O(*n*) time when adding a polynomial of size *n* (i.e., *n* non-zero terms) to a polynomial of size *m*.

Note: `addIn` is a *new* Polynomial operation. It's like add, but it's a mutator for the polynomial we call it on.

Examples of using `addin` (calls shown in bold):

```
Polynomial p = new Polynomial(new Term(8,2));            // p is  8.0x^2
p.addIn(new Polynomial(new Term(7,1));          // p is now 8.0x^2 + 7.0x
Polynomial q = new Polynomial (new Term(10, 6)).add(
                      new Polynomial(new Term(4, 2));
                                                // q is  10.0x^6 + 4.0x^2
p.addIn(q);  // after the call p has the value: 10.0x^6 + 12.0x^2 + 7.0x
             //   (and q is unchanged)
```

Reminder: Java's rules allow us to access the private data of parameter `b` in `addIn`.

Hint: since HashMap's are unordered, you will not be using the merge algorithm.

Tip: liberal use of local variables will help you avoid reapeating lengthy expressions in your code for addIn (i.e., less writing)

```
public class Polynomial {

   private Map<Integer, Double> polyMap;
       // representation invariant: maps exponents to their coefficients,
       //                           storing only non-zero terms


   public Polynomial() {
      polyMap = new HashMap<Integer, Double>();
   }


   //  . . .  (rest of class implementation not shown)


   public void addIn(Polynomial b) {
```

[Space given for your answer on the next page]

Problem 5 (cont.)

```
public class Polynomial {

   private Map<Integer, Double> polyMap;

       // representation invariant: maps exponents to their coefficients,
       //                           storing only non-zero terms


   public Polynomial() {
      polyMap = new HashMap<Integer, Double>();

   }

   //  . . .  (rest of class implementation not shown)


   public void addIn(Polynomial b) {
```

Problem 6 [20 pts]

Implement the *C++* function `splice`, which inserts a list, called `subList`, inside another list, called `list`, at location `k` in `list`. This function does not create or destroy any nodes, but just re-links up existing nodes. Note that `subList` is unusable after the call (because its nodes are now part of a different list). The `Node` and `ListType` definitions are on the code handout.

Examples of calls to **`splice(list, k, subList)`:**

`list` before call	**`k`**	**`subList`**	**`list`** after the call
(2 3 8 3 9)	*2*	*(6 6 6)*	*(2 3 6 6 6 8 3 9)*
()	*0*	*(6 6 6)*	*(6 6 6)*
(8 7 5)	*0*	*(6 6 6)*	*(6 6 6 8 7 5)*
(8 5)	*2*	*(1 2)*	*(8 5 1 2)*
(8 5)	*1*	*()*	*(8 5)*

```
// PRE: list and subList are well-formed linked lists
//      and 0 <= k <= the length of list
void splice(ListType & list, int value, ListType subList) {
```