Name: ______________________________

USC NetID (e.g., `ttrojan`**):**______________________

CS 455 Final Exam
Fall 2016 [Bono]

December 13, 2016

There are 6 problems on the exam, with 67 points total available. There are 10 pages to the exam (5 pages double-sided), including this one; make sure you have all of them. There is also a one-page code handout that accompanies the exam. If you need additional space to write any answers, pages 8, 9, and 10 of the exam are left blank for that purpose. If you use these pages for answers you just need to direct us to look there. ***Do not detach any pages from this exam.***

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 username (a.k.a., NetID) at the top of the exam. Also, put your NetID at the top right of the front side of each page of the exam. Please read over the whole test before beginning. Good luck!

Problem 1 [10 points]

[C++] Consider the following two C++ programs that use a `Student` class and answer the questions below each one (see code handout for documentation of `Student` class methods):

```
// Program 1

void foo(Student * s) {
   s->addScore(15);
   s = NULL;
}

int main() {
   Student * joe = new Student("Joe");
   foo(joe);
   cout << joe->getScore() << endl;
}
```

Part A. In the space above, draw a box-and-pointer diagram (a.k.a., memory diagram) showing all object variables, objects, and their state as they have changed during the code sequence. This includes showing `foo`'s parameters.

Part B. What is the output of Program 1?

```
// Program 2

void bar(Student * & t) {
   t->addScore(25);
   t = NULL;
}

int main() {
   Student * sal = new Student("Sal");
   bar(sal);
   cout << sal->getScore();
}
```

Part C. In the space above, draw a box-and-pointer diagram for Program 2 like you did for Part A.

Part D. What is the output of Program 2?

Problem 2 [8 points]

[Java] Consider the following code to do linear search in an array (written in Java). It doesn't currently work in all cases that satisfy the precondition.

```
// returns the location of target in nums, or -1 if not found.
// PRE: nums is not null and nums has no duplicates
public static int lookup(int[] nums, int target) {

   inti  = 0;

   while (nums[i] != target) {

      i++;

   }

   if (nums[i] == target) {

      return i;

   }

   else {

      return -1;

   }

}
```

Part A. Provide a test case where the code does **not** work correctly, and the corresponding result of that test case. (Show your answer under the headings below.)

nums **target** **result**

Part B. Provide a test case where the code **does** work correctly, and the corresponding result of that test case. (Show your answer under the headings below.)

nums **target** **result**

Part C. Fix the code above. Do not rewrite the whole method, but rather make minimal 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.

Problem 3 [10 points]

[Java] The binary search algorithm can be implemented recursively. Below is a partially complete recursive binary search implementation (consists of the two functions below). **Complete the code below so it does what its comments say. The only missing parts are indicated by the boxes and underlined areas. (i.e., fill in the boxes and underlined areas provided.)**

```
// returns the location of target in words array, or -1 if not found.
// PRE: words is in alphabetical order and has no duplicates
public static int binSearch(String[] words, String target) {
   return binSearchR(words, target, 0, words.length-1);
}



// returns the location of target in the part of words with indices [low,high]
// or -1 if not found
// PRE: words is in alphabetical order and has no duplicates
public static int binSearchR(String[] words, String target, int low, int high) {
   if (low >high) {
      ┌──────────────────────────────────────────────────────┐
      │                                                      │
      └──────────────────────────────────────────────────────┘
   }
   int mid = (low + high) / 2;
   if (target.equals(words[mid])) {
      ┌──────────────────────────────────────────────────────┐
      │                                                      │
      └──────────────────────────────────────────────────────┘
   }
   else if (target.compareTo(words[mid]) < 0) {    // target < words[mid]

      return binSearchR(words, target, ______________________,____________________);
   }
   else {                                          // words[mid] < target

      return binSearchR(words, target, ______________________,____________________);
   }
}
```

USC NetID: ___________________________

Problem 4 [4 pts]

[Java] Recall that the `Concord` class computes the number of occurrences of all words from a document. Consider the two variations of our `Concord` class below. Version A is the one we did in lecture and lab: it takes a `Scanner` as a *parameter* to the `addData` method. Version B's `addData`, instead creates a *local* variable for the `Scanner` (and initializes it inside the method). To highlight the differences between the two they are shown in **bold** in the code below. (The other methods (not shown) of both classes are a constructor that creates an empty version of the concordance (i.e., with no words in it), and another method to print out the results.)

Describe a scenario that illustrates an advantage of Version A over Version B. Limit your answer to one to two sentences.

Version A.

```
// Computes the number of occurrences of all words from a Scanner
public class Concord {
    // . . . [other details of Concord left out]

    /**
      Add data from Scanner to concordance.
      @param in data to scan.  "in" will be at the end of its data after this
      operation.
     */
    public void addData(Scanner in) {
       while (in.hasNext()) {
          String word = in.next();
          < . . . add this occurrence of word to the number of occurrences . . . >
       }
    }
}
```

Version B.

```
// Computes the number of occurrences of all words in the input.
public class Concord {
    // . . . [other details of Concord left out]

    /**
      Add data from System.in to concordance (reads until end-of-file).
     */
    public void addData() {
       Scanner in = new Scanner(System.in);
       while (in.hasNext()) {
          String word = in.next();
          < . . . add this occurrence of word to the number of occurrences . . . >
       }
    }
}
```

Problem 5 [20 pts]

Write the *Java* function `printReversedSentences` which reverses the order of all the words in each sentence from a Scanner (it does not reverse the order of the sentences themselves), printing the reversed versions. **For full credit you must use a Java `Stack`** (see code handout for interface). A sentence has at least one word and is terminated with a period. To make the problem easier, we are not going to worry about proper capitalization. Here is an example of this function at work (pay attention to the placement of the periods):

input text

```
ahem. mary had a little lamb. its fleece was white as snow. the end.
```

corresponding output text

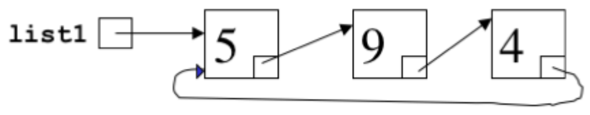
```
ahem. lamb little a had mary. snow as white was fleece its. end the.
```

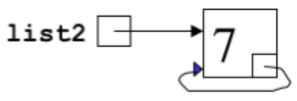
```
//  prints out a version of the words from the scanner such that the order of
//  words is reversed in each sentence.  Processes all the words from the
//  Scanner.  All output will be on the one line (i.e., no newlines printed).
//  PRE: the scanner contains a sequence of zero or more sentences, each one
//     ending with a period.

public static void printReversedSentences(Scanner in) {
```

Problem 6 [15 points]

[C++] A circular list, used in some list applications, is a special linked list representation where the last node in the list points back to the **first** node. Here are some examples of a circular list:





Write the C++ function `isCircularList`, which tells whether its parameter is a well-formed linked list or a circular list. Hint: in C++ you can use the equality operator (==) on pointers (analogous to comparing object references in Java for equality).

The `Node` and `ListType` definitions are on the code handout.

```
// returns true iff list is a circular list
// PRE: list has at least one node and is either a well-formed linked list
//      or a circular list

bool isCircularList(Node * list)
```

Extra space for answers or scratch work. (DO NOT detach this page.)

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.

USC NetID: ____________________________

Extra space for answers or scratch work (cont.) (DO NOT detach this page.)

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.

Extra space for answers or scratch work (cont.) (DO NOT detach this page.)
If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.