NAME ______________________________ NET ID ____________________

Final CS100J 15 Dec 2003, 15:00–17:30 Barton Hall HAVE A GOOD BREAK!

Question 0.	_________	(out of 01)
Question 1.	_________	(out of 10)
Question 2.	_________	(out of 12)
Question 3.	_________	(out of 08)
Question 4.	_________	(out of 09)
Question 5.	_________	(out of 10)
Question 6.	_________	(out of 15)
Question 7.	_________	(out of 20)
Question 8.	_________	(out of 15)
Total		(out of 100)

The results of this Final will be posted on CMS as soon as it is graded. Grades for the course will take several days to create and post to the CMS.

Submit regrade requests for the course using CMS where possible (email gries@cs.cornell.edu where it is not possible) by noon tomorrow.

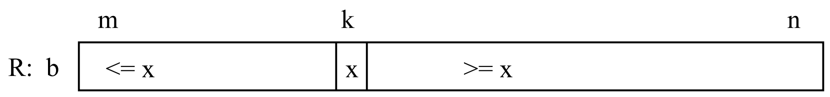
You have 2.5 hours to complete the questions in this final. The questions are numbered 0..8. The exam is worth 100 points. Please glance through the whole exam before starting.

In answering a question, read it through first. If it has several parts, concentrate on one at a time. Solve any problem in terms of several smaller/simpler problems. You may separate the pages and write on the back of each. We have staplers at the front of the room.

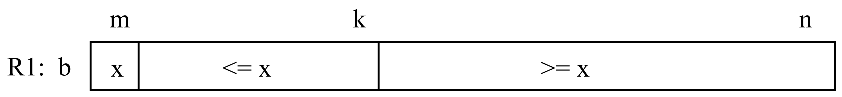
Question 0 (1 point). Print your name and net id at the top of each page. Please make them legible.

Question 1 (10 points). Algorithms. Write the partition algorithm, which, given a nonempty array segment b[m..n], permutes its values and stores a value in variable k to truthify the following, where x is the initial value in b[m]:

Postcondition R: b[m..k–1] <= x, b[k] = x, b[k+1..n] >= x



DON’T write a method. Just write the statements that perform the task. The algorithm requires ONE loop, whose postcondition is:



You are expected to write the loop invariant first and then to develop the loop from the invariant. You can write the invariant as a picture, in English, or in a mixture of English and mathematics. It is up to you. If you have to swap two array elements, for example b[c] and b[d], just say “Swap b[c] and b[d]”. You don’t have to write the Java statements to perform the swap. But all other statements should be in Java.

Question 2 (12 points). One-dimensional arrays and loops. You know how to add two integers together, as shown in the box to the right. The digits are added from right to left, and at each position there is a *carry* (could be 0 or 1) from the position to its right. For example, adding the 7 and 6 results in 3 with a carry of 1, which gets added into the next position to the left.

```
   99937
+     86
---------
  100023
```

Write Java code to perform such addition. Here are the ground rules:

The digits of the first operand are in b[0..bs–1], *with least significant digit first*;
The digits of the second operand are in c[0..cs–1], *with least significant digit first*;
Both b and c contain positive integers;
The digits of answer are to be placed in d[0..ds–1], *with least significant digit first*.

This means that you must assign both to ds and to the elements of d. Assume that d and ds are already declared and that d already contains an array that is big enough. Below, we show b, bs, c, cs, and the result d, ds for the addition shown in the box above.

Input: bs = 5, b = {7, 3, 9, 9, 9}, cs = 2, c = {6, 8}: Output: ds = 6, d = {3, 2, 0, 0, 0, 1}

Your algorithm should have one loop, with initialization and some "finalization" code after it. The loop must use the following invariant, and you should develop the loop and its initialization using the four loopy questions. No extra variables should be used. You can use functions Math.min and Math.max if you need to. It may help to draw the invariant as a picture.

invariant:
(1) d[0..ds–1] has been calculated. In detail:
d[0..ds–1] contains the sum of b[0..Math.min(ds,bs)–1] and c[0..Math.min(ds,cs)–1]
(2) d[ds] contains the carry from position d[ds–1] (or 0 if ds = 0).

Question 3 (8 points). Arrays and methods. Write a method with this specification:

```
/** = “array b is triangular”. By “triangular”, we mean that:
      row 0 has 1 element
      row 1 has 3 elements
      row 2 has 5 elements
      etc. …
  */
public static boolean isTriangular(int[][] b) {





}
```

Question 4 (09 points). Executing Java statements. On this page is a class C and a class Price.

(a) Draw (to the right) one folder of class C, showing the Object partition of the folder as well as any others.

(b) Execute the call

C.doIt();

Write the output of println statements directly beneath the println statements, in the space provided. You need not draw folders or frames for calls for us to see, but drawing at least folders may help you.

```
public class C {
   public static void doIt() {
      Price d= new Price("ball", 0);
      Price e= d;
      d.increasePrice(64);
      d= new Price ("doll", 128);
      Price f= d;
      f.increasePrice(256);
      System.out.println("d: " + d.getPrice() +
                     "\n e: " + e.getPrice() +
                     "\n f: " + f.getPrice());





      Price c1= new Price("jump rope", 64);
      Price c2= new Price("jump rope", 128);
      c1.increasePrice(64);
      System.out.println("d = e: " + (d == e));




      boolean a= f == d;
      boolean b= c1 == c2;
      System.out.println("a: " + a + "\n  b: " + b);
```

```
public class Price {
   private String item;
   private int price;

   public Price(String t, int m) {
      item= t;
      price= m;
   }

   public int getPrice() {
      return price;
   }

   public void increasePrice(int m) {
      price= price + m;
   }
}
```

Question 5 (10 points). Matlab. Write a Matlab function that, given an argument n, produces an array that contains the cumulative sums of the first n terms of this series:

$$-1/5 + 2/10 - 3/15 + 4/20 - 5/25 + 6/30 - \ldots$$

Loops may not be used.

Question 6 (15 points). Miscellaneous class stuff
(a) Define the terms *parameter* and ***argument*.**

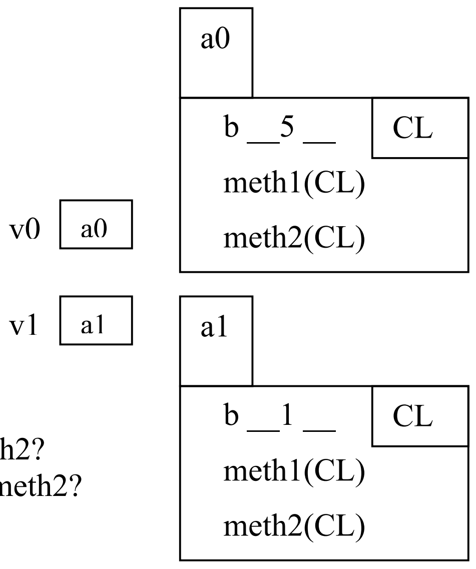
(b) State the reason for making a class abstract.

(c) Suppose we have the folders and variables shown to the right and that method meth1 is this:

```
public meth1(CL p) {
    meth2(this);
    meth2(p);
}
```

Suppose this call is executed: v0.meth1(v1);

1. Draw the frame for the call v0.meth1(v1);
2. What is the value of the argument of the first call to meth2?
3. What is the value of the argument of the second call to meth2?



Question 7 (20 points). Classes. The New York State Motor Vehicle department wants a Java class that will represent license plates. Each instance of class License will contain (1) the name of a person (a String) and (2) a license plate "number" that is assigned to the person (also a String, such as "NYS 00100").

The class has a constructor, but the normal user should *not be able to use it*. Instead, if someone wants a new license plate, they have to call method *assign*, with their name as argument. The method assigns them a license plate "number", creates an instance of License that contains the person's name and license plate "number", and returns (the name of) the instance. Here is an example of a call on *assign* and an assignment of the value it yields.

```
License lic= License.assign("David Gries");
```

Another point. The first license plate "number" to be assigned is "NYS 00001". The next one to be assigned is "NYS 00002", then "NYS 00003", and so on. Somehow, class License must keep track of which numbers have been assigned. One way to implement this is to have a variable, declared appropriately, that always contains the last license plate "number" assigned.

Another point. Class License should maintain a Vector of all instances of Licenses that have been created; thus, the Vector should contain all instances of the class that were ever created by the constructor.

Finally, it should be possible to ask through a function call whether there is already a license for a given person; the function should return the License with that registration number, if there is one. (What should it return if there isn't one? You must specify this.)

Write class License. It should contain declarations of all the variables and methods necessary to do what is described above. Variables and methods should be static or non-static, private or public, as required by good programming practices and to have the class work properly. Make sure you put a comment on each variable to describe its meaning.

Don't write method bodies. Instead, put good specifications as comments on the method headers and use {} for each method body. Write any getter methods (but not their bodies) that you feel are needed under good standard programming practices; they don't need specifications if the method names follow standard conventions.

This page is left intentionally nonblank (use it for the previous question if you want).

Question 8 (15 points). Subclasses

Below are two classes. Assume that the other class that is used, Address, exists. Class Date is in package java.util.

(a) Fill in the body of the constructor of class Student.
(b) Fill in the body of the constructor of class StudentAdmitted. Assume that the constructor of Date yields an instance with the date on which the instance was created.
(c) Draw a folder (instance) of class StudentAdmitted. You don't have to show class Object.
(d) Write the body of function equals of the second class. You can assume that Date and Address have equals functions that check the contents of Date and Address folders.

```
// An instance is a Student
public class Student {
   private String name; // Student's name
   private Address ad;     // Student's address

   /** Constructor: a student with name name and address ad */
   public Student(String name, Address ad) {


   }

   /** =  this student's name */
   public String getName() { return name; }

   /** = this student's address */
   public Address getAddress() { return ad; }
}
```

```
import java.util.*;

/** An instance is a Student along with the date they were admitted */
public class StudentAdmitted extends Student {
   private Date date; // The date the student was admitted

   /** Constructor: a student with name p and address ad;
       the student is admitted on the date this folder is created */
   public StudentAdmitted(String p, Address ad) {


   }

   /** = "this instance contains the same values as instance sa" */
   public boolean equals(StudentAdmitted sa) {



        }
}
```