Cornell net id _____ Name _____

Section day _____ Section time _____

CS 100J Prelim 3 13 November 2007

This 90-minute exam has 6 questions (numbered 0..5) worth a total of 100 points. Spend a few minutes looking at all questions before answering any. Budget your time wisely. Use the back of the pages, if you need more space. We have a stapler at the front of the room, so you can tear the pages apart.

Question 0 (2 points). Write your netid and your name, legibly, at the top of each page (Hint: do it now).

Question 1 (15 points) Recursion. By repeatedly summing the digits of a positive integer, one can reduce the integer to a single digit, in the range 1..9. For example, 56 reduces to 11, which reduces to 2. Similarly, 99934 reduces to 34, which reduces to 7. And 999999999999992 reduces to 128, which reduces to 11, which reduces to 2.

Because addition is symmetric (b+c = c+b) and associative ((b+c)+d = (b + (c+d)), it doesn't matter what order the summing is done. For example, we can reduce 998 is several ways, *some* of which are:

998 –> 99 + 8 –> 18 + 8 –> 26 –> 8
998 –> 99 + 8 –> 18 + 8 –> 9 + 8 –> 8
998 –> 99 + 8 –> 107 –> 10 + 7 –> 1 + 7 –> 8
998 –> 99 + 8 –> 107 –> 10 + 7 –> 17 –> 8

Below, write a function that reduces its parameter n to a single digit and returns that digit. You have to (1) specify the function properly, (2) write the function header, and (3) write the function body. The function body should use recursion. It may not use a loop.

Cornell net id _____ Name _____

Section day _____ Section time _____

Question 2 (15 points). Loops.

Below, we want a sequence of statements that calculates the element-wise averages of arrays a and b, where a and b have the same length. For example, if a = {3, 0, 4, 7} and b = {1, 5, 0 , 0}, the resulting array c is {2.0, 2.5, 2.0, 3.5}.

Below, we give: (1) a declaration of an array `c` that is to contain the answer, (2) a postcondition that says that `c` contains the answer, and (3) a loop invariant.

You job is to write the initialization and loop that will store values in array `c` so that the postcondition is truthified. Your initialization and loop *must* be developed to use the invariant shown, using the four loopy questions. An answer that does not attempt to use the invariant may receive 0 points.

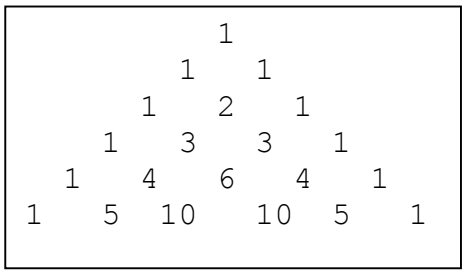
Feel free to draw the postcondition and invariant as pictures-diagrams, as we have been doing in class, if that helps you.

```
// double arrays a and b have the same length.
double[] c= new double[a.length];


// Invariant: c[k..c.length-1] contains the element-wise averages  of
//            a[k..a.length-1] and b[k..b.length-1].




// Postcondition: c[0..c.length-1] contains the element-wise averages of
//                a[0..a.length-1] and  b[0..b.length-1].
```

Cornell net id _____ Name _____

Section day _____ Section time _____

Question 3 (20 points). Arrays and loops.



Pascal's triangle of size `n+1` (for `n >= 0`) is as shown to the right, where `n = 5`. Row 0 contains one 1. Row 1 contains two 1's. Row `k`, for `2 ≤ k ≤ n`, contains `k+1` integers. The first and last are 1's, and each of the inner values is the sum of the two just above it.

Write function `pascal`, specified below. The returned array will be a two-dimensional rectangular array `b` (say). Row `k` (`0 ≤ k ≤ n`) contains `k+1` values, and they will go in the first `k+1` elements of row `k` of `b`.

Hint. The function body will probably have nested loops. You do not have to write loop invariants for the loops if you do not want to. We suggest that you first write the outer loop with a comment in the repetend saying what the repetend is supposed to do. THEN attempt to implement that comment.

```
/** = a two-dimensional array that contains n+1 rows, where each row k contains in its
       first k+1 elements row k of Pascal's triangle.
       Precondition: n ≥ 0. */
public static int[][] pascal(int n){




}
```

Cornell net id _____ Name _____

Section day _____ Section time _____

Question 4 (28 points) Classes

(a) In modular arithmetic, or "arithmetic mod `m`", integers are kept in a range `0..(m-1)`. The value `m` is called the *modulus*. For example, minutes are always in the range 0..59 (so `m` is 60) and hours of the day in the range 0..23 (so `m` is 24). In modular arithmetic, the numbers wrap around. For example, the integers mod 4 are, in succession, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, … —adding 1 to 3 in mod 4 arithmetic yields 0, adding 2 to 3 in mod 4 arithmetic yields 1, etc. Interestingly enough,

```
(b mod m) + (c mod m) = (b+c mod m)
```

So, to add two integers in modular arithmetic, just add them as normally done and then change the answer to be in the right range.

An instance of class `Mod`, given on the next page, represents an integer in mod `m` arithmetic. Note the "class invariant" —the restrictions on fields `k` and `m` given as comments. These restrictions must be maintained.

Write the bodies of the constructor (note carefully its specification) and functions add and equals. In writing them, you do not have to check that preconditions mentioned in the specs are met.

(b) What is meant by "overriding a method"? How does one call an overridden method `m(…)` from within the class that does the overriding?

(c) What are the four kinds of variables in Java, and where is each declared?

(c) Does the first statement inside a constructor always have to be a call on some constructor? Explain. If yes, what constructor call is used if it is missing?

Cornell net id _____ Name _____

Section day _____ Section time _____

```
/** An instance is an integer in mod m arithmetic */
public class Mod {
   private int m;          // The modulus. m > 1.
   private int k;          // The integer. 0 ≤ k < m.

   /** Constructor: integer k in mod m arithmetic.  Precondition: m > 1 and k ≥ 0.*/
   public Mod(int k, int m) {









   }

   /** If  this object and r do not have the same modulus, return null; otherwise, return an object
       that contains the sum of the two mod m integers represented by this object and r. */
   public Mod add(Mod r) {









   }

   /** = "ob is a non-null Mod object with the same modulus and value as this one". */
   public boolean equals(Object ob) {









   }
}
```

Cornell net id _____ Name _____

Section day _____ Section time _____

Question 5 (20 points) Binary search.

Consider an array `d` of `String`s. Assume that the `String`s in `d` are in dictionary order (e.g. "`bcd`" comes before "`bed`"). Assume that function `comesBefore` exists:

```
/**= -1 if s1 comes before s2, in dictionary order,
       0 if s1 and s2 are equal, and
       1 if s2 comes before s1, in dictionary order.
       */
public static int comesBefore(String s1, String s2);
```

0 _____	out of 02
1 _____	out of 15
2 _____	out of 15
3 _____	out of 20
4 _____	out of 28
5 _____	out of 20
Total _____	out of 100

1. Write a specification (pre- and post-conditions) and header for a binary search function `bSearch` that searches for the rightmost occurrence (if present) of String `w` in sorted (in dictionary order) String array segment `d[p..q-1]`. If `w` occurs in the segment, `bSearch` should return the index of the rightmost occurrence of `w`; if `w` does not occur in the segment, `bSearch` should return the index where `w` belongs (as done in lecture).

Your specification may be in terms of pictures, English, mathematical notation, or a mixture thereof. But it must be a specification of the binary search algorithm *presented in class* (except that this one works on `String`s instead of **`int`**s).

2. Develop the body of `bSearch`. When developing its body, write the invariant of the loop first. Then develop the loop and initialization using the four loopy questions. An answer that does not have a suitable loop invariant will receive less than half credit.

Please read the directions given above and follow them carefully.