The prelim is 90 minutes long. There are 5 questions (plus a Question 0) and 6 pages in total. Be sure to answer all the questions. Please write clearly and show all your work. It is difficult to give partial credit if all we see is a wrong answer. Also, be sure to place suitable comments — method specifications, and variable definitions — in your programs. Use the backs of pages if necessary. You can tear pages apart; we have a stapler at the front of the room.

Question 0. (2 points). Please write your name and netid at the top of each page.

Question 1. Binary search (18 points). In class, we developed a binary search algorithm that had a specification like the one below. Write the body of this method. Do not use recursion. You will need a loop. Be sure you write the loop invariant and make sure that the loop can be understood in terms of the loop invariant.

```
/** b[0..N-1] is sorted (in ascending order).
    Return an integer k that truthifies this assertion:

        b[0..k] <= x < b[k+1..N-1] */
public static int search(int[] b, int x) {
```

Question 2. Short questions (15 points).

(a) State how one proves, by mathematical induction, that some property P(n) holds for all $n \geq 0$.

(b) Consider a class C with a method m. What are the two consequences of making C and m abstract?

(c) Function Integer.parseInt(String s) returns the **int** value of the integer that is in String s. But if s does not contain an integer, the function throws a NumberFormatException. Write a statement that stores in a variable dn the value of the function call:

```
Integer.parseInt(somestring)
```

but stores 1 in dn if a NumberFormatException is thrown.

Question 3. Recursive functions (25 points). A subsequence of a string is obtained by deleting 0 or more characters from the string. For example, "Cornell" contains these subsequences (and more): "", "C", "Crnll", "oe". Note that "eo" is not a subsequence.

A *common subsequence* of two strings is a subsequence that appears in both. For example, some of the common subsequences of the two strings "recursion" and "iteration" are: "ero", "ion", and "rion". Note that the empty string "" is always a common subsequence.

Write a *recursive* function lcs(s1, s2) that returns a longest common subsequence of the two strings s1 and s2. If there are several, return any one of them. Treat lower and uppercase letters as different characters, e.g. 'A' and 'a' are different.

Please be sure that your recursive function looks at *all* common subsequences. Do not worry about efficiency; just write the simplest possible function you can think of.

Here are a few examples of what function lcs(s1, s2) returns on different inputs:

lcs("bright","ithaca") is "it"
lcs("bright","Ithaca") is "t"
lcs("abcde","xyz") is ""
lcs("recursion","iteration") is "erion"

s.length() is the length of String s
s.charAt(k) is the character at index k.
s.substring(k) is the substring consisting of the chars s[k], s[k+1], s[k+2], ….

Question 4. Classes (20 points). This question is designed to test your understanding of interfaces, classes, and subclasses. Shown to the right are two interfaces: Personal and Business.

```
public interface Personal {
    public String getName();
    public int getAge();
}

public interface Business {
    public String getAddress();
    public int getEarnings();
}
```

(a) Write the definition a class Celebrity that implements interface Personal. An instance Celebrity has a name (a String) and an age (an **int**). The class should have a suitable constructor and any other methods that must be present for this class. There is no need to put comments on the fields, but please put in method specifications.

(b) Write the definition of a subclass Actor of Celebrity. Since actors need to maintain a business profile, this class will have to implement interface Business. An instance of this Actor has a name, age, address (String) and earnings (**int**). The class should have a suitable constructor and any other methods that must be present for this class. For full credit, there should be no duplicate code in classes Celebrity and Actor. There is no need to put comments on the fields, but please put in method specifications.

Question 5. Interfaces (25 points). To the right is Java interface Iterator with two of its three functions. You will not be using or implementing function remove().

Class Process is defined as follows. We show **only** what you need to know for this question.

```
public class Process {
    public …                ; // info about the process
    public int priority;      // priority of the process

    /** Constructor: instance of Process with priority p */
    public Process(int p)
       { priority= p; }
}
```

```
public interface Iterator {
  /** = there exists another
        item in the iteration */
  public boolean hasNext();

  /** = the next item in the itera-
    tion. If there is no next item,
    throw a
    NoSuchElementException */
  public Object next();
}
```

If Process P's priority is greater than Q's priority, we say P has higher priority than Q.

Each instance of class Scheduler (shown below) implements a simplified model of a process scheduler found in operating systems. It has two arrays, userA and userB, which contain the Processes being run by users A and B respectively. Each array is already sorted by the priority of the Processes it contains, in descending order (from high to low). Not all methods of the Scheduler class are shown, since you will not need them.

Method iterator (see below) returns an Iterator of *all* the Processes currently stored in the instance. It does this by returning the value of the expression

new ProcessIterator()

Write class ProcessIterator as an inner class of class Scheduler (on the next page). Each call to function next() in class ProcessIterator should return the Process that has the highest priority among the as-yet-unenumerated Processes. If there is more than one such Process, any one of them can be returned. Do not create any new data structure. Do not do any sorting.

```
public class Scheduler {

    private Process[] userA;   // userA[0..sizeA-1] contains user A's processes, in
    private int sizeA;         // order of priority
    private Process[] userB;   // userB[0..sizeB-1] contains user B's processes, in
    private int sizeB;         // order of priority

    /** = an Iterator of the Processes in this instance, in order of priority */
    public Iterator iterator() {
        return new ProcessIterator();
    }
```

—— Write inner class ProcessIterator on this page ——

Question 0	_____/2
Question 1	_____/18
Question 2	_____/15
Question 3	_____/25
Question 4	_____/15
Question 5	_____/25
Total	_____/100

}