Name_____

Net-ID_____

Prelim One Solution

CS211 — Fall 2006

Closed book; closed notes; no calculators. Write your name and Net-ID above. Write your name clearly on ***each*** page of this exam. For partial credit, you must show your work.

Do not begin until we instruct you to do so. You have 90 minutes to complete the problems.

Name_____

1. (*20 points; 2 each*) True or false?

a) **T** F The statement "**`String s;`**" implies that the *static type* of **`s`** is **`String`**.

b) T **F** The modifier **`private`** signifies that the method or variable can only be accessed from within its class and from within its subclasses.

c) **T** F An abstract class can implement interfaces and can include static methods and constructors.

d) **T** F If class **`A`** inherits from abstract class **`B`**, and **`B`** implements an interface **`C`**, then a cast from an object of type **`A`** to an object of type **`C`** is an instance of *upcasting*. For example,

```
A a;
...
C c = (C) a;
```

e) T **F** In Java, a variable's *static type* is always different from its *dynamic type*.

f) T **F** The term *weak induction* is used to indicate a type of error common in induction proofs.

g) **T** F In Java, objects are stored in the heap.

h) **T** F If a variable is declared to hold a primitive type, then the variable cannot hold the value **`null`**.

i) T **F** During a Java program's execution, there is at most one *stack frame* per method.

j) T **F** Any binary tree with ≤ 7 nodes has at most 3 levels.

Name_____

2. (*Induction; 20 points; 10 each*)

Make sure you clearly show (i) your base case or cases, (ii) your induction hypothesis, (iii) the inductive step, and (iv) your conclusion.

2a. Use induction to prove that $n! > 2^n$, for $n \geq 4$.

Base Case: $4! = 24 > 2^4 = 16$

Induction Hypothesis: $k! > 2^k$ for some $k \geq 4$.

Inductive Step: $(k+1)! = (k+1)k! > (k+1)\ 2^k$ by the IH.
Also by the IH, $k > 3$; thus, $(k+1) > 4 > 2$.
Using this, we have $(k+1)! > (k+1)\ 2^k > 2*2^k = 2^{k+1}$.

Conclusion: $n! > 2^n$, for $n \geq 4$.

2b. Use induction to prove that a binary tree of height h has $\leq 2^{h+1} - 1$ nodes. (Note that the *height of a tree* is defined to be the *number of links* on the longest path from root to leaf.)

Base Case: A binary tree of height zero has one node; $h = 1 \leq 2^{0+1} - 1 = 1$.

Induction Hypothesis: (This proof is easiest if we use strong induction.)
Assume a binary tree of height $\leq k$ has $\leq 2^{k+1} - 1$ nodes.

Inductive Step: Consider a tree T of height k+1.
T has two subtrees, both of height $\leq k$. (At least one of them must have height k, but we don't need this information for the proof.)
By the IH, each of these subtrees has $\leq 2^{k+1} - 1$ nodes.
The total number of nodes in tree T is 1 for root, plus the number in the left subtree, plus the number in the right subtree.
This sum is $\leq 1 + 2^{k+1} - 1 + 2^{k+1} - 1 = 2^{k+2} - 1$.

Conclusion: A binary tree of height h has $\leq 2^{h+1} - 1$ nodes.

This proof can be done using weak induction, but you need to change the statement you are trying to prove into "A binary tree of height $\leq h$ has $\leq 2^{h+1} - 1$ nodes.

Name_____

3. (*Lists; 10 points*)
Write a static method called **`reverse`** that takes (the head of) a linked list using nodes of type **`ListNode`** and returns (the head of) a new, circular, doubly-linked list with the same values, but in *reverse order*. The new list should use nodes of type **`CircularListNode`**.

Do not create any additional classes beyond **`ListNode`**, **`CircularListNode`**, and the **`Problem3`** class that holds your method. You may use helper methods, but do not modify **`ListNode`** or **`CircularListNode`**. Do not modify the method header for method **`reverse`**.

```
class ListNode {
     public int data;
     public ListNode next;
}
class CircularListNode {
     public int data;
     public CircularListNode next;
     public CircularListNode previous;
}
class Problem3 {
     public static CircularListNode reverse (ListNode n) {
          // YOUR CODE //

          if (n == null) return null;
          CircularListNode last = new CircularListNode();
          last.data = n.data;
          current = last;
          for (n = n.next; n != null; n = n.next) {
               temp = current;
               current = new CircularListNode();
               current.data = n.data;
               current.next = temp;
               temp.previous = current;
          }
          // Link current and last nodes to make circular list
          current.previous = last;
          last.next = current;
          return current;




     }    // end method
}         // end class
```

Name_____

4. (*Inheritance; 20 points; 2 each*)
Consider the following class declarations.

```
class A {
      int val;
      public A() {}
      public A(int _val) { val = _val; }
      public void output() { System.out.println(88); }
  }
public class B extends A {
      int val;
      public B(int _val) { val = _val; }
      public B(int v1, int v2) { super(v1 + v2); }
      public void output() { System.out.println(val); }

      public static void main(String[] args) {
          A p0 = new A(3);
          A p1 = new B(5);
          A p2 = new B(6, 7);
          B p3 = new B(8, 9);
          // Code snippet goes HERE
      }
}
```

For each of the following snippets of Java code, assume that the snippet is inserted as indicated into the main method of class **B** and determine whether the code will produce *no error*, a *run-time error*, or a *compile-time error*. For a snippet that produces output, determine the *output* as well. Write your answer in the blank (i.e., **//_____**) provided. (Assume that each snippet is tested independently of the others.)

```
p0.output(1);                    //___Compile-Time Error

System.out.println(p1.val);      //___0

p2 = p3;                         //___No Error

p3 = p2;                         //___Compile-Time Error

p3 = (B)p0;                      //___Run-time Error

p1.output();                     //___5

p3.output();                     //___0

((A)p3).output();                //___0

B p4 = new B();                  //___Compile-Time Error

B p5 = new A(2);                 //___No Error
```

Name_____

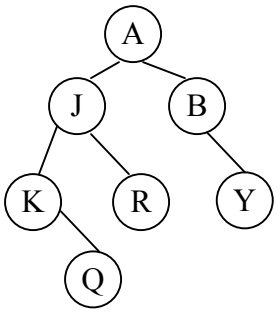
5. (*Trees; 20 points; 10, 5, 5*)

5 (a, b, c). Show the preorder, inorder, and postorder traversals for the binary tree, below:

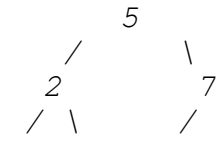
a. preorder: *AJKQRBY*

b. inorder: *KQJRABY*

c. postorder: *QKRJYBA*

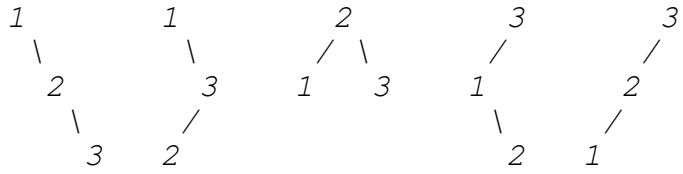


5d. Insert the following values into a *binary search tree* (BST), inserting in the specified order: 5, 2, 3, 7, 6, 1. What is the sum of the leaf nodes in the resulting tree? (It is possible to receive partial credit, but only if you show your work.)



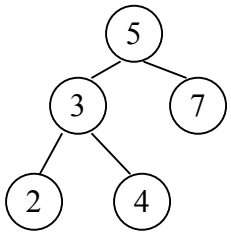
Leaf sum = 1 + 3 + 6 = 10.

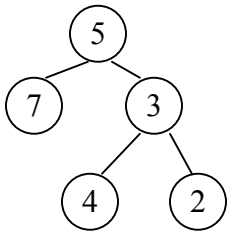
5e. Draw all possible binary search trees (BSTs) with the nodal values 1, 2, and 3.



Name_____

6. (*Recursion; 10 points*)
Write a Java method **`mirror`** that returns (the root of) a *mirrored binary tree*. For instance, given (the root of) the tree on the left, below, the method should return (the root of) the tree on the right, below. Note that your method must be *non-destructive*: the original tree should not be modified.





Write your code in the space provided. Do **not** modify the method header, add new methods/ constructors, add instance/class variables, or write additional classes. Your method *must use recursion* in a useful way.

```
class Node {
    public int val;           // data value stored at the node
    public Node left, right;  // left and right children,
                              // null if non-existent

    /*
     * Given an input tree rooted at Node n,
     * return the root of the mirrored tree.
     */
    public static Node mirror (Node n) {
          // --- YOUR CODE ---

        if (node == null) return null;
        Node m = new Node();
        m.val = n.val;
        m.left = mirror(n.right);
        m.right = mirror(n.left);
        return m;









    } // method mirror
} // class Node
```