CS211 Spring 2007
Prelim 1
March 8, 2007

Solutions

Instructions

Write your name and Cornell netid above. There are 6 questions on 6 numbered pages. Check now that you have all the pages. Write your answers in the boxes provided. Use the back of the pages for workspace. Ambiguous answers will be considered incorrect. The exam is closed book and closed notes. Do not begin until instructed. You have 90 minutes. **Good luck!**

	1	2	3	4	5	6	Σ
Score	/15	/24	/10	/15	/21	/15	/100
Grader							

1. (16 points) A node of a binary tree is *nearly balanced* if either (i) the node is a leaf, (ii) the node has one child and that child is a leaf, or (iii) the node has two children and the heights of its left and right subtrees differ by no more than 1. A binary tree is *nearly balanced* if all its nodes are nearly balanced. Prove by induction that the number of nodes in a nearly balanced binary tree of height k is at least f_k, the kth Fibonacci number. Recall $f_0 = 0$, $f_1 = 1$, and $f_k = f_{k-1} + f_{k-2}$ for $k \geq 2$.

   Label clearly the parts of your proof. In the induction step, state the induction hypothesis clearly and indicate exactly where in your argument it is used.

   Hint. You will probably need strong induction, and there will probably be two base cases.

   By strong induction on k.

   There are 2 base cases, $k = 0$ and $k = 1$. For $k = 0$, the only tree of height 0 is a leaf, which has 1 node, and $1 > 0 = f_0$. For $k = 1$, the smallest nearly balanced tree of height 1 is a node with a single child, which has 2 nodes, and $2 > 1 = f_1$. That's it for the basis.

   Now let T be a nearly balanced tree of height $k \geq 2$. The strong induction hypothesis states that the theorem holds for all nearly balanced trees of height strictly less than k. At least one of T's subtrees (call it A) must have height exactly $k - 1$. The other subtree (call it B) must exist, otherwise condition (ii) in the definition of nearly balanced would fail, and B must have height at least $k - 2$ by condition (iii). By the strong induction hypothesis, A must have at least f_{k-1} nodes and B must have at least f_{k-2} nodes, thus T has at least $f_{k-1} + f_{k-2} + 1 > f_k$ nodes.

2. (16 points) Say what is printed by the following Java program.

```
public class Test {
   public static void main(String[] args) {
      A a = new A();
      System.out.println(a.x);
      System.out.println(a.m());
      B b = new B();
      System.out.println(b.x);
      System.out.println(b.m());
      a = new B();
      System.out.println(a.x);
      System.out.println(a.m());
      b = (B)a;
      System.out.println(b.x);
      System.out.println(b.m());
   }
}

class A {
   int x;
   A() { this(1); }
   A(int x) { this.x = x; }
   int m() { return x; }
}

class B extends A {
   int x;
   B() { this(2); }
   B(int x) {
      super(x+1);
      this.x = super.x + 1;
   }
   int m() { return x; }
}
```

```
1
1
4
4
3
4
4
4
```

3. (16 points)

   (a) (8 points) Suppose you had a method `m()` of a class `A` inherited by a subclass `B`. Say you would like to know how many times during the run of the program `m()` is called as a member of an object of dynamic type `B` and how many times it is called as a member of an object of dynamic type `A`. Is it possible to modify the method `m()` and the class `A` to keep track of this information? If so, show how. If not, say why not.

```
class A {
   static int aCount = 0;
   static int bCount = 0;
   void m() {
      if (this instanceof B) bCount++;
      else aCount++;
   }
}
class B extends A {}
```

   (b) (8 points) A binary tree is *full* if every node has either 0 or 2 children. Suppose you had the following definition of a `TreeNode` class, and wanted to modify it to enforce fullness. You would like to make sure that no tree can ever be constructed that is not full (including later modification), while still allowing the left and right children of the node to be read and changed. Say briefly (in words) how the class could be modified to accomplish this.

```
class TreeNode {
   public Object datum;
   public TreeNode left, right;

   public TreeNode(Object datum, TreeNode left, TreeNode right) {
      this.left = left;
      this.right = right;
      this.datum = datum;
   }

   public TreeNode(Object datum) {
      this(datum, null, null);
   }
}
```

Make the constructor with 3 arguments check that both left and right are null or both are nonnull. Make the left and right fields private instead of public and access them with public getters and setters. Have just one setter that sets both left and right simultaneously, checking that both arguments are null or both are nonnull.

4. (16 points) True or false?

(a) T [✓ F] A successful cast `(Integer)x` changes the dynamic type of the object occupying `x` to `Integer`.

(b) T [✓ F] The cast `(Integer)x` changes the static type of the variable `x` to `Integer`.

(c) [✓ T] F A call to `super` or `this` in a constructor must occur first.

(d) [✓ T] F A `static` field is shared by all objects of the class.

(e) [✓ T] F A non-abstract subclass of an abstract class `A` must implement all methods declared abstract in `A`.

(f) [✓ T] F Abstract classes and interfaces cannot be instantiated.

(g) [✓ T] F `int[]` is a reference type.

(h) T [✓ F] Local variables of a method can be declared `static` provided the method is declared `static`.

(i) T [✓ F] Shadowing and overriding are the same thing.

(j) [✓ T] F Any inductive proof using weak induction can also be done using strong induction.

(k) [✓ T] F The `main` method must be static.

(l) T [✓ F] A method in an abstract class must have an empty body.

(m) T [✓ F] Uncaught `RuntimeException`s must be declared in the method header.

Questions (n)–(p) refer to the following grammar for numbers in scientific notation. Spaces are not significant. `<sci>` is the start symbol of the grammar.

```
<nz_digit>  →  1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
   <digit>  →  0 | <nz_digit>
     <int>  →  <digit> | <digit><int>
     <exp>  →  E+<int> | E−<int>
     <sci>  →  <nz_digit>.<int><exp> | −<nz_digit>.<int><exp> | 0.0
```

(n) [✓ T] F The following is a sentence: `1.0618E-15`

(o) T [✓ F] The following is a sentence: `9.0`

(p) [✓ T] F The number of possible sentences is infinite.

5. (16 points) Construct a binary tree with nodes A–I having the following traversals. Nodes may have 0, 1, or 2 children.

   Preorder: B A D C E F G I H
   Inorder: A F E G C D I B H

```
      B
     / \
    A   H
     \
      D
     / \
    C   I
   /
  E
 / \
F   G
```

6. (20 points) Say we are given `null`-terminated integer lists built from `ListCells`

```
ListCell {
   int datum;
   ListCell next;
}
```

We would like to compare two lists lexicographically, that is, in dictionary order. The definition of lexicographic order is as follows.

(a) Find the first cell position where the data in two lists differ, if such a position exists. The list with the smaller datum in that position is lexicographically smaller. (If we were talking about words in the dictionary, this would be like "aardvark" < "abacus".)

(b) If no such position exists and one list is longer than the other, that is, if the shorter list is a prefix of the longer, then the shorter is lexicographically smaller. (Like "cat" < "catnip".)

(c) Otherwise the lists are equal.

Write a *recursive* method `lex` that compares two lists lexicographically. It should take two lists and return a negative number if the first is lexicographically less than the second, 0 if they are equal, and a positive number if the second is lexicographically less than the first.

```
public static int lex(ListCell c1, ListCell c2) {
   if (c1 == null && c2 == null) return 0;
   if (c1 == null) return -1;
   if (c2 == null) return 1;
   if (c1.datum < c2.datum) return -1;
   if (c1.datum > c2.datum) return 1;
   return lex(c1.next, c2.next);
}
```

END OF EXAM