CSCI 3155: Principles of Programming Languages
Spring 2014: January 28, 2014 **Quiz 1**

Name: ____________________

This quiz has 3 questions for a total of 20 points. The duration of this quiz is 20 minutes.

1. Consider the following function in Scala:

```
1   def f(n: Int): Boolean = {
2
3     require(n >= 0)
4
5     def g(n: Int): Boolean = n match {
6
7       case 0 => false
8
9       case n => f(n - 1)
10
11    }
12
13    n match {
14
15      case 0 => true
16
17      case _ => g(n - 1)
18
19    }
20
21  }
```

(a) $\boxed{\text{3 points}}$ **Scoping**. For each variable use site in the above code, draw an arrow from the use site to the binding site? **Hint**: There a total of 7 use sites.

> **Solution:**
>
> - Use of n at 3, bound at 1.
> - Use of n at 5, bound at 5.
> - Use of f at 9, bound at 1.
> - Use of n at 9, bound at 9.
> - Use of n at 13, bound at 1.
> - Use of g at 17, bound at 5.
> - Use of n at 17, bound at 1.

(b) $\boxed{\text{3 points}}$ **Evaluation**. What does f(3) evaluate to? Please show the important steps of evaluation using the "e steps e' in 0 or more steps" relation from the course notes:

$e \longrightarrow^* e'$. For this question, we will consider function call expressions (e.g., `f(3)` and `g(2)`) and the final value important. For example, the first step is `f(3)` $\longrightarrow^*$ `g(2)`. You are welcome to chain the step relation (e.g., writing $e \longrightarrow^* e' \longrightarrow^* e'' \longrightarrow^* \cdots \longrightarrow^* v$ where the e's are function call expressions and v is the final value).

Solution: `f(3)` $\longrightarrow^*$ `g(2)` $\longrightarrow^*$ `f(1)` $\longrightarrow^*$ `g(0)` $\longrightarrow^*$ **false**

(c) 2 points Explain briefly in one sentence what function `f` computes?

Solution: Function `f` returns **true** if and only if its argument is an even natural number.

2. 6 points **Recursion**. Implement a recursive function that computes x^n (i.e., raises `x` to the `n`th power). It does not need to be particularly efficient.

```
def pow(x: Int, n: Int): Int =
```

Solution:

```
def pow(x: Int, n: Int): Int = n match {
  case 0 => 1
  case _ => x * pow(x, n - 1)
}
```

This implementation is $O(\mathtt{n})$, which is not the most efficient one.

3. 6 points **Recursive Data Structures**. Consider the following recursive data type `SearchTree` from Lab 1:

```
sealed abstract class SearchTree
case object Empty extends SearchTree
case class Node(l: SearchTree, d: Int, r: SearchTree) extends SearchTree
```

Implement a function `sum` recursively that sums up all of the data values in all nodes of tree `t`. For extra credit, re-implement `sum` using an accumulator parameter (note, your implementation is not likely to be tail recursive, which is ok).

```
def sum(t: SearchTree): Int =
```

Solution: The following solution is probably the most straightforward one:

Text

```
def sum(t: SearchTree): Int = t match {
  case Empty => 0
  case Node(l, d, r) => sum(l) + d + sum(r)
}
```

Here, we implement `sum` using a helper function with an accumulator parameter. Note that this implementation is not tail recursive because there is work to be done after return from the `loop(l, acc)` recursive call.

```
def sum(t: SearchTree): Int = {
  def loop(t: SearchTree, acc: Int): Int = t match {
    case Empty => acc
    case Node(l, d, r) => loop(r, d + loop(l, acc))
  }
  loop(t, 0)
}
```

In case you are curious, it is possible to implement `sum` tail recursively using what is called a *continuation*:

```
def sum(t: SearchTree): Int = {
  def loop(t: SearchTree, k: Int => Int): Int = t match {
    case Empty => k(0)
    case Node(l, d, r) => loop(l, a => d + loop(r, k))
  }
  loop(t, a => a)
}
```

The continuation parameter `k` is like an accumulator the accumulates the operations that need to be done to compute the sum. We have not (yet) discussed continuations, so you are not expected to have been able to come up with this solution. It is, however, informative to trace through an evaluation using this solution to see how it works.