EECS 356: Programming Language Concepts, Written Exercise 2
due Monday, March 21, 2016

Problem 1: Consider the following Java code (assuming Java allows methods to be declared inside other methods:

```
public class AClass {
  private static int a = 10;
  private static int b = 20;

  public static int bmethod(int y) {
    int b = 30;

    public static int dmethod(int z) {
       int a = z;
       return a + cmethod(z);
    }

    return a + b + dmethod(y);
  }

  public static int cmethod(int x) {
    int a = 40;
    if (x == 0) {
      return a + b;
    }
    else {
      return bmethod(x-1) + a + b;
    }
  }

  public static void main (String[] args) {
    int b = 2;
    System.out.println(cmethod(b) + a + b);
  }
}
```

What is the value printed by the `System.out.println` statement when the `main` method is run if:
a) Java uses static scoping
b) Java uses dynamic scoping
Be sure to trace your reasoning and justify your answer!

Problem 2: Consider the following program written in C/Java. What is the contents of `list` if we use:
a) call-by-value
b) call-by-reference
c) call-by-value-result
d) call-by-name ?
If there is more than one possible answer, list all possibilities. Be sure to trace your reasoning and justify your answer!

```
void incrementAll (int val1, int val2, int val3) {
  val1 += 1;
  val2 += 1;
  val3 += 1;
}

void main() {
  int save = 1;
  int list[] = {1, 2, 3, 4, 5};
  incrementAll(save, list[save], list[list[save]]);
}
```

Problem 3: (Exercise 7.2 of the Scott book)
In the following code, which of the variables will a compiler consider to have compatible types under
a) structural equivalence
b) strict name equivalence
c) loose name equivalence ?

```
type T = array[1..10] of integer
     S = T
var A : T
var B : T
var C : S
var D : array[1..10] of integer
```

Problem 4: (Exercise 7.25 of the Scott book)
In Example 7.88 we noted that functional languages can safely use reference counts since the lack of an assignment statement prevents them from introducing circularity. This isn't strictly true; constructs like Scheme's `letrec` can also be used to make cycles, so long as uses of cicularly define names are hidden inside lambda expressions in each definition

```
(define foo
    (lambda ()
      (letrec ((a (lambda (f) (if f 0 b)))
               (b (lambda (f) (if f 1 c)))
               (c (lambda (f) (if f 2 a))))
        a)))
```

Each of the functions a, b, and c contais a reference to the next:

```
> ((foo) #t)
0
> (((foo) #f) #t)
1
> ((((foo) #f) #f) #t)
2
> (((((foo) #f) #f) #f) #t)
0
```

How might you address this circularity without giving up on reference counts?

Problem 5: One reason tombstones are rarely used is that each tombstone must persist to prevent the dangling pointer problem. We may be able to solve this problem by implementing garbage collection on the tombstones. If we do so, should we use reference counters or mark-and-sweep? Either argue that one method is superior to the other or argue that both methods are equally good (or equally poor). (Hint: why would a language need to use tombstones?)