CSDS 345: Programming Language Concepts, Written Exercise 2
due Monday, April 19, 2021

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:
Does Java use strict name equivalence, loose name equivalence, or structural equivalence when determining if two primitive types are compatible and/or equivalent? What about for non-primitive types? Give examples to justify your answer.

Problem 3: Consider the following program written in a C-like language. What is the contents of `values` if we use:
a) call-by-value
b) call-by-value-result
c) call-by-reference
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!

```
type coord = struct {int x; int y; int z};

function rotate(coord[] a, coord b) {
   if (a[0].x >= a[1].x)
      b.y = a[0].y;
   else
      b.y = a[1].y;

   if (a[0].y > a[1].y)
      b.z = a[0].z;
   else
      b.z = a[1].z;

   if (a[0].z <= a[1].z)
      b.x = a[0].x;
   else
      b.x = a[1].x;
}

void main() {
  coord[] values = { {0, 1, 0}, {1, 0, 1} };
  rotate(values, values[values[0].z]);
  print(values);
}
```

Problem 4: 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 and how would programmers use this feature of the language?)

Problem 5:
We saw in lecture that some of the type inference rules have direct logical equivalences. For example, reasoning about functional composition is equivalent to the logical hypothetical syllogism rule.

Consider the following functions:

```
int g(String s) {
double f(int x, int y) {
```

(Although the functions are written in Java-style, this is an arbitrary functional language.)

Let function `h` be defined as

```
h(s1, s2) = f(g(s1), g(s2))
```

Use only Curry-ing, "hypothetical syllogism" and "implication introduction" to prove that `h` has type String $\rightarrow$ String $\rightarrow$ double.