CMSC330 Practice Problems 7

1. Polymorphism
   Consider the following Java classes:

   ```java
   class A { public void a() { … } }
   class B extends A { public void b() { … } }
   class C extends B { public void c() { … } }
   ```

   Explain why the following code is or is not legal

   a. int count(Set<A> s) { … } … count(new TreeSet<A>( ));
   b. int count(Set<A> s) { … } … count(new TreeSet<B>( ));
   c. int count(Set s) { … } … count(new TreeSet<A>( ));
   d. int count(Set<? extends A> s) { … } … count(new TreeSet<A>( ));
   e. int count(Set<? extends B> s) { … } … count(new TreeSet<A>( ));
   f. int count(Set<? extends B> s) { … } … count(new TreeSet<A>( ));
   g. int count(Set<? super B> s) { for (A x : s) x.a(); … }
   h. int count(Set<? super B> s) { for (C x : s) x.c(); … }
   i. int count(Set<? super B> s) { for (A x : s) x.a(); … }
   j. int count(Set<? super B> s) { for (C x : s) x.c(); … }

2. Function arguments
   For each code, explain whether g is an upward or downward funarg.

   a. let f x = let g y = x + y in let app a b = a b in app g 1 ;;
   b. let f x = let g y = x + y in let app a b = a b in app g 1 ;;

3. Static vs. Dynamic Scoping
   Consider the following OCaml code.

   ```ocaml
   let app f w = let x = 1 in f w ;; // value of x determined here
   // for dynamic scoping
   let add x y = let incr z = z+x in app incr y;; // value of x determined here
   // for static scoping
   (add 2 3) ;;
   ```

   c. What is the order of invocation for the functions app, add, and incr when evaluating
      the expression (add 2 3)?
   d. What value is returned by (add 2 3) with static scoping? Explain.
   e. What value is returned by (add 2 3) with dynamic scoping? Explain.
4. Parameter passing
Consider the following C code.

```c
int i = 2;
void foo(int f, int g) {
    f = f - i;
    g = f;
}
int main() {
    int a[] = {2, 0, 1};
    foo(i, a[i]);
    printf("%d %d %d %d\n", i, a[0], a[1], a[2]);
}
```

a. Give the output if C uses call-by-value
b. Give the output if C uses call-by-reference
c. Give the output if C uses call-by-name

5. Lazy evaluation
Given the following OCaml code.

```ocaml
let doIf p x = if p then x else 0 ;;
let rec loop n = loop n ;;
dolif false (loop 0) ;;
```

a. What is the result of evaluating the dolIf expression if OCaml uses call-by-value?
b. What is the result of evaluating the dolIf expression if OCaml uses call-by-name?
c. Rewrite the code (using thunks) so that the result of evaluating the dolIf expression is the same as if OCaml used call-by-name, even though OCaml uses call-by-value.