Recitation 1 : Lambda Calculus

1. Solve the following lambda expressions until they are irreducible. The “lambda” symbol is denoted by “L”.

   a. (Lx.Ly.x) x y

   b. (Lx.xx) (Lx.xx)

   c. ((Lxyz.y (w y x)) (Lsz.z)) # comment: Lxyz is just a shorthand for Lx.Ly.Lz

   d. (Lx. (Ly. y x) (Lz. x z)) (Ly. y y)

2. Let *f* be a function of the type A -> B, and *g* be a function of the type B -> C. The composition of f and g, will be a function of the type A -> C. This composition is read as “g of f”. For example, if we let *f* denote the unary increment of an integer, and *g* denote division by 2, then the composition g of f applied to 3 will yield 2 (i.e., 3+1, then divided by 2). Verify that the lambda expression Lf.Lg.Lx (f (g x)) is the composition “g of f” as explained above.

3. Show that the lambda expression ((Lx.Ly.y) y) (Lx.x a) reduces to a.

4. Lambda calculus is the base of any and all functional languages. A major aspect of any and all functional languages is the use of recursion. Some problems in computer science are primarily, or can only be thought of in recursive terms, such as traversing a tree. In other cases, you can take an iterative problem and turn it into a recursive problem. Think of the idea of multiplication between positive numbers as repeatedly adding a number to itself n times, e.g.

```
int mult(x, y) {
     int temp = x;
     for (int i = 1; i<y; i++) {
         temp+=x;
     }
     return temp;
}
```

   Now try to do this recursively. Feel free to add as many helper functions as you want, but you must only use addition and subtraction to get your goal. No multiplication must be used, and remember: no need to worry about negative numbers.