CSci 340, Spring 2003, Project 5

This project is due *Friday, April 25* at 11:20am.

The “`getcs 340 5`” command will fetch the file `lmb.hs`, which defines a type for representing lambda expressions.

```
data LambdaExpr = Lmb String LambdaExpr
    | LambdaExpr :@ LambdaExpr | Id String
```

This says that lambda expressions can be built using three different data constructors: `Lmb` representing a function, `:@` representing a function application, and `Id` representing a variable.[1] We could use this data type to represent the lambda expression $(\lambda x. * \; x \; x) \; 2$ as follows.

```
(Lmb "x" (Id "*" :@ Id "x" :@ Id "x")) :@ Id "2"
```

In fact, the provided code defines `test0` to be exactly this value.

The provided code also places LambdaExpr in the Show class. Thus, if you type “`test0`” after loading `lmb.hs` in Hugs, the interpreter use the provided `show` function to display “`(\x.* x x) 2`”.

1. Define a function called *`subst`* for replacing all free instances of a variable with an expression.[2]

   ```
   subst :: String -> LambdaExpr -> LambdaExpr -> LambdaExpr
   ```

   This represents a function taking three arguments, a string id, a lambda expression val, and a lambda expression $expr$. It should return a new lambda expression based on $expr$, except with each free occurrence of id replaced by val. For example, the expression

   ```
   subst "x" (Id "2") (Id "*" :@ Id "x" :@ Id "y")
   ```

   should return “`* 2 y`” (as given by the `show` function).

   Your `subst` function should be careful not to replace bound variable instances. Consider the following.

   ```
   subst "x" (Id "2") (Lmb "x" (Id "*" :@ Id "x" :@ Id "y"))
   ```

   In this case, it should return “`\x.* x y`”, not “`\x.* 2 y`”. The reason is that x is bound as a parameter in the expression $\lambda x. * \; x \; y$, and so it should not be replaced.

2. Define a function called *`simplify`* for simplifying a lambda expression into normal form.

   ```
   simplify :: LambdaExpr -> LambdaExpr
   ```

   For example, given the `test0` definition above, the expression “`simplify test0`” should return “`* 2 2`”.

   Your function should perform all β-reductions possible. If I were to type “`simplify (Lmb "z" test0)`”, your function should return “`\z.* 2 2`”. (Whether it simplifies lazily or eagerly is not important; do not bother with η-reductions.)

For both functions, feel free to define other functions that you find useful. You may use the `subst` function in defining `simplify`.

Submit your solution using “`handincs 340 5`”. Also, submit a paper copy of the functions you wrote; in this paper copy, *do not* include the code I included.

[1] The `:@` constructor is an *infix data constructor*, described at the top of page 11 of the Haskell handout.

[2] A *free* variable is a variable that is not bound. A *bound* variable is one whose purpose is defined within the expression. For example, in the expression $\lambda x.f \; x$, the x variable is bound (it identifies a parameter value), but f is free.