CSCI 3330 Comparative Languages
Joy Reed
Spring 2013

Programming Assignment 3: A Functional Interpreter
Due Date: Thursday April 18, 2013

The purpose of this assignment is to use recursively defined data structures and functions for a naturally-recursive problem.

Design and implement an interpreter for a baby imperative language, called BIL. A program in BIL consists of a set of declarations, followed by a set of assignment statements. Our interpreter will take an abstract syntax tree for a BIL program as input, “executes” it by building a state of (variable, value) pairs, and completes by returning the final state. I have created a starter script called adt_2013.hs.

A simple concrete program in BIL:

```
x; y; z;
x = 1;
y = 2;
z = 3;
x = x*x + y +z;
```

For simplicity, the only data type is integer. You will not have to parse the concrete programs; rather, you will create your own abstract syntax trees by hand.

Requirements:

0. i. Explicitly declare all types for your Haskell functions in your source, i.e., preface all of your function definitions with their types.
   ii. Provide corresponding concrete version of all BIL declarations and statements as comments in your Haskell script.

1. Use the starter file to define a constant called smallBil that represents the program:

```
x; y;
x = 1;
inc x;
y = x+4;
```

2. Define the abstract syntax below as Haskell data types. Build and print out in some fashion test programs. Comment the script with the concrete syntax of the programs.

```
Program = Prog Decl*  Stmt*
Decl = Variable Ident
Stmt = Assign Ident Expr
Expr =    Plus Expr Expr
        | Times Expr Expr
        | Inc Ident
        | Var Ident
        | Val Int
```

3. We will not be concerned with concrete syntax in this project, other than as documentation. Rather, the interpreter will execute an abstract syntax structure, returning a "state" of variables mapped to integers. Implement such an interpreter.

   Some function signatures and types you might find useful. The grand finale is *exec*.

```
type State = [(Ident,Val)]        for type Ident = String
exec :: Program -> State          -- executing program returns a final state
getVal :: Ident -> State -> Int   -- return the integer binding of a variable
eval :: Expr -> State -> Int      -- evaluate an expression
elab :: Declaration -> State -> State -- add a declaration to a state
```

4. Expand the language and the interpreter to include a while statement.

Extra Credit:

Expand the language abstract syntax with block declarations/statements for nesting scopes.

Deliverables: Submit as hard copy. *The hard copy descriptions should be stand alone and completely demonstrate that you met the requirements.*

1. Title page with your name, date, course number, instructor, and assignment number. 1 page max.

2. Copy of the assignment.

3. Table of Contents 1 page max.

3. Work completed: Specify clearly which requirement you completed or partially completed. ½ page max.

4. Source (properly commented) and screen shots (effectively annotated where useful).

Source files: Source codes should be submitted as .hs file or windows zipped directory.

Points will be deducted for sloppy and careless presentation.

30% -- Completion of Requirements.
30% -- Quality of Solution
10% -- Quality of demonstrated test cases
20% -- Clarity and effectiveness of presentation