COMP2209-PROGRAMMING III

COUSEWORK III

Here is the list of the assessments that you will solve in coursework III. Your solution for the given assessment is due by the given handin date but may be submitted at any point before that.

Repeated multiple submissions are allowed before the deadline. You will be given feedback on how well your solutions perform against our test suit. Marks will only be given to your final submission after the deadline. The coursework III comprises 15% of the overall assessment for the module.

You are required to build an interpreter for the lambda-calculus by using a machine-based implementation. By the end of the assessment you should be familiar with the CEK machine model and how to vary it to include new features.

Throughout this sheet we will be working with the data type for lambda calculus abstract syntax trees.

```
data Expr = Var String | Lam String Expr | App Expr Expr
   deriving (Eq,Show,Read)
```

We are going to implement the CEK machine for Call-by-Value lambda calculus as presented in the lecture. In order to check your implementation you may like to use substitution based lambda calculus interpreter.
(For more detail regarding CEK Machine: https://en.wikipedia.org/wiki/CEK_Machine)

ASSESSMENT A1

First define a type synonym for environments named Environment. An environment consists of associations between variable names and expressions. Now extend the Expr datatype given above with an extra constructor to allow for a representation of closure expressions. That is, modify

```
data Expr = Var String | Lam String Expr | App Expr Expr | ???
   deriving (Eq,Show,Read)
```

with an extra clause. Recall that closures comprise a pair of a lambda expression and an environment but to specify this here it is slightly easier to give the lambda expression broken down in to the string variable name and an expression for its body. With this in mind, the new constructor should have three parameters.

In the way that we will use environments, they will always associate variable names to closures that the variable is bound to in that environment. To access the closure associated with a variable we will need a `lookup' function. The type of this should be

```
lookup :: String -> Environment -> Maybe Expr
```

and, given the string name of a variable and an environment, it should return the closure that the variable is bound to in that environment. Think about what should you return in case the variable has no binding in the environment. There is a built-in function lookup that may be of use to you or define your own directly.
(3 marks)

ASSESSMENT A2

Define suitable types for Frames and Continuations. Look at the grammar for Frames - the data type should resemble that. Given these we can define a type for what we shall call a *configuration*. A configuration is the triple consisting of the current Control, Environment, and Continuation.
(3 marks)

ASSESSMENT A3

Implement a function called eval1 that takes a configuration and returns the configuration given by executing one step of the machine by applying the rules R1-R5. Pattern matching is very much your friend when it comes to defining this function. Effectively each rule becomes one case of the eval1 function. You will need a case of the function that deals with successful termination and another that catches any reduction errors. You will find it helpful to define some additional functions for manipulating closures and environments also.
(3 marks)

ASSESSMENT A4

We should be able to put this all together now to implement a function called eval that takes a lambda expression and evaluates it using a Call-by-Value strategy until termination. The function should return the terminated expression (if it terminates). You will need to think what the initial configuration should be and how to recognise termination.
(3 marks)

ASSESSMENT A5

An example lambda calculus term is given by

```
App (Lam "v"
      (App
        (App (Var "v") (Lam "z" (Var "z")))
        (App (Lam "v" (App
                         (App (Var "v") (Lam "x" (Lam "y" (Var "x"))))
                         (Lam "z" (Var "z")))
              )
              (Lam "x" (Lam "y" (Var "x")))
```

```
        )
      )
    )
(Lam "x" (Lam "y" (Var "y")))
```

It should evaluate to Lam "x" (Lam "y" (Var "x")). Test your implementation on this term to check that it does. Build some other simple terms to test with.
(3 marks)

ASSESSMENT A6

This exercise is a little harder and is for those who like a challenge.

Extend the lambda calculus with natural numbers by providing terms Zero and Succ e to represent natural numbers. Do this by modifiying the data type above for Expr above to include Zero and Succ Expr. Extend the CEK machine implementation above to cater for this extended langauge. Hint: you will need to identify what the terminated values of the language are, modify frames accordingly and rewrite the machine rules R1-R5.
(5 marks)

-- End of File --