Name: Beh Shu Ao

Student ID: 33354723

Student Email: sab1e22@soton.ac.uk

COMP2209 Programming III Coursework 3

```haskell
-- {-# LANGUAGE LambdaCase #-}
-- {-# OPTIONS_GHC -Wno-overlapping-patterns #-}
import Data.Data (ConIndex)
import Control.Monad.ST.Lazy (strictToLazyST)
import System.Posix.Internals (set_console_buffering)

-- A1:: Type for Expressions
type Environment = [(String, Expr)]
data Expr = Var String | Lam String Expr | App Expr Expr | Closure String Expr
Environment | Zero | Succ Expr deriving (Eq,Show,Read)

-- A2: Types for Frames and Continuations
data Frame = MissingLeftFrame Expr |MissingRightFrame Expr Environment |
MissingSuccFrame  deriving (Eq, Show, Read)
type Continuation = [Frame]
type Configuration =(Expr,Environment, Continuation)

-- A1:: Lookup Function
lookupFunction :: String -> Environment -> Expr
lookupFunction _ [] = Var "Nothing"
lookupFunction var ((name, expr) : env) | var == name = expr
                                | otherwise = lookupFunction var env

-- A3: Evaluation for configuration
eval1 :: Configuration -> Configuration
eval1 (Var x,env,k)= (lookupFunction x env, env,k) -- R1: a simple lookup
eval1 (App e1 e2, env, k) = (e1,env,MissingRightFrame e2 env : k)  -- R2:
pluck leftside and put rightside in the continuation with the environment
eval1 (Lam x e1, env, k) =(Closure x e1 env,env,k) -- R3:wrap it in a closure
with a copy of the current environment
eval1 (v,env,MissingRightFrame e2 env' : k) = (e2,env',MissingLeftFrame v:k) -
- R4
eval1 (v,env,MissingLeftFrame (Closure x e1 env'):k) =(e1,extendEnv x v
env',k) --R4
eval1 (v,env,MissingLeftFrame (Var x):k) =(Var x,extendEnv x v env,k) -- R5

eval1 (Succ e, env, k) = (e, env, MissingSuccFrame: k) -- R4: evaluation of
Succ complete
eval1 (Zero, env, MissingSuccFrame: k) = (Zero, env, k) -- R5: Zero is a
terminated value for Succ
```

```haskell
eval1 _ = error "No applicable reduction rule"

-- Helper function to extend the environment
extendEnv :: String -> Expr -> Environment -> Environment
extendEnv x expr env = (x, expr) : env

-- Initial configuration setup
initialConfig :: Expr -> Configuration
initialConfig expr = (expr,[],[])

-- A4: Eval function using a call-by-value strategy
eval :: Expr -> Expr
eval expr =  eval' (initialConfig expr)
  where
    eval' :: Configuration -> Expr
    eval' config =
      case eval1 config of
        newConfig@(App _ _ ,_,_) -> eval' newConfig --Case with a Application
        newConfig@(Var _,[],[]) -> eval' newConfig -- Case with a Variable
        newConfig@(Closure x e1 env, _ ,[])-> Lam x e1 -- Only Closure and the
continuation is empty means it is terminated
        newConfig@(Zero,_,[]) -> Zero -- Only Zero and the continuation is
empty means it is terminated
        newConfig@(Succ e,_,[])->Succ e
        newConfig -> eval' newConfig -- Catch all other case

main :: IO ()
main = do
  -- A5: Test Case for the output
  let expr =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")))

  putStrLn "Eval Result (Assessment 5):"
  print $ eval expr -- expected Output: Lam "x" (Lam "y" (Var" "x"))

  putStrLn "------------------------------------------------------------"
  putStrLn "Eval Simple Test1 (LookupFunction): "
  print $ eval1 (Var "x",[("x", Var "x"), ("y", Lam "z" (Var "z"))],[]) --
(Var "x",[("x",Var "x"),("y",Lam "z" (Var "z"))],[])

  putStrLn "------------------------------------------------------------"
```

```
  putStrLn "Eval Simple Test2 (Application): "
  print $ eval1  (App (Var "a") (Var "b"),[],[]) --(Var
"a",[],[MissingRightFrame (Var "b") []])

  putStrLn "------------------------------------------------------"
  putStrLn "Eval Simple Test3 (Lambda): "
  print $ eval1 (Lam "x" (Lam "y" (Var "y")),[],[]) --(Closure "x" (Lam "y"
(Var "y")) [],[],[])

  putStrLn "------------------------------------------------------"
  putStrLn "Eval Simple Test4 (Missing Right Frame):"
  print $ eval1 (Closure "a" (Var "a") [],[],[MissingRightFrame (Var "b") []])
--(Var "b",[],[MissingLeftFrame (Closure "a" (Var "a") [])])

  putStrLn "------------------------------------------------------"
  putStrLn "Eval Simple Test5 (Missing Left Frame in Closure Type):"
  print $ eval1 (Closure "b" (Var "b") [] ,[],[MissingLeftFrame (Closure "a"
(Var "a") [])]) --(Var "a",[("a",Closure "b" (Var "b") [])],[])

  putStrLn "------------------------------------------------------"
  -- A6: Test Case for Zero and Succ e
  let expr2 = App (Lam "x" (Succ (Succ Zero))) (Succ (Succ (Succ Zero)))
  putStrLn "Eval Result (Succ and Zero):"
  print $ eval expr2

  putStrLn "------------------------------------------------------"
  let testExpr = App (Lam "x" Zero) (Succ (Succ (Var "x")))
  putStrLn "Eval Result (Zero Test Case):"
  print $ eval testExpr
```

Output running in Command Prompt:

```
C:\Users\BehShuAo\Desktop\BehShuAo\CW3\33354723_CW3>ghci CW3.hs
Loaded package environment from C:\Users\BehShuAo\AppData\Roaming\ghc\x86_64-mingw32-9.4.7\environments\default
GHCi, version 9.4.7: https://www.haskell.org/ghc/  :? for help
[1 of 2] Compiling Main             ( CW3.hs, interpreted )
Ok, one module loaded.
ghci> main
Eval Result (Assessment 5):
Lam "x" (Lam "y" (Var "x"))
-------------------------------------------------------
Eval Simple Test1 (LookupFunction):
(Var "x",[("x",Var "x"),("y",Lam "z" (Var "z"))],[])
-------------------------------------------------------
Eval Simple Test2 (Application):
(Var "a",[],[MissingRightFrame (Var "b") []])
-------------------------------------------------------
Eval Simple Test3 (Lambda):
(Closure "x" (Lam "y" (Var "y")) [],[],[])
-------------------------------------------------------
Eval Simple Test4 (Missing Right Frame):
(Var "b",[],[MissingLeftFrame (Closure "a" (Var "a") [])])
-------------------------------------------------------
Eval Simple Test5 (Missing Left Frame in Closure Type):
(Var "a",[("a",Closure "b" (Var "b") [])],[])
-------------------------------------------------------
Eval Result (Succ and Zero):
Succ (Succ Zero)
-------------------------------------------------------
Eval Result (Zero Test Case):
Zero
```