Київський національний університет ім. Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

Звіт

з лабораторної роботи №6

з дисципліни

«Функціональне програмування»

Виконав:

студент групи ІПЗ-41

Ляшуга Микола

## 2019

## **Завдання**

Написати програму символьного диференціювання (по одній змінній) алгебраїчних виразів, представлених у формі правильних префіксних виразів, які містять усі арифметичні операції та вказані у варіанті математичні функції, ch(x)

**Виконання:**

module Calculus where

  import Data.Maybe

  import Data.Char

  data UnOp = Neg | Sin | Cos | Log | Ch | Sh

            deriving (Eq, Ord)

  instance Show UnOp where

   show Neg = "-"

   show Cos = "cos"

   show Sin = "sin"

   show Log = "log"

   show Ch = "ch"

   show Sh = "sh"

  --adfsadfasdfsafsafd

  data BinOp = Add | Mul | Div

             deriving (Eq, Ord)

  instance Show BinOp where

    show Add = "+"

    show Mul = "\*"

    show Div = "/"

  data Exp = Val Double | Id String | UnApp UnOp Exp | BinApp BinOp Exp Exp

           deriving (Eq, Ord, Show)

  type Env = [(String, Double)]

  unOpMap :: [(UnOp, Double -> Double)]

  binOpMap :: [(BinOp, Double -> Double -> Double)]

  unOpMap = [(Neg, (\*) (-1)), (Sin, sin), (Cos, cos), (Log, log), (Ch, ch), (Sh, sh)]

  binOpMap = [(Add, (+)), (Mul, (\*)), (Div, (/))]

  lookUp :: Eq a => a -> [(a, b)] -> b

  lookUp search table

   = fromJust (lookup search table)

  eval :: Exp -> Env -> Double

  eval (Val val) \_ = val

  eval (Id var) table = lookUp var table

  eval (UnApp op exp) table

   = (lookUp op unOpMap) (eval exp table)

  eval (BinApp op exp exp') table

   = (lookUp op binOpMap) (eval exp table) (eval exp' table)

  diff :: Exp -> String -> Exp

  diff (Val val) \_ = Val 0

  diff (Id var) var'

   | var == var' = Val 1

   | otherwise = Val 0

  diff (BinApp op exp exp') var

   | op == Add = BinApp Add (diff exp var) (diff exp' var)

   | op == Mul = BinApp Add uv' u'v

   | op == Div = BinApp Div (BinApp Add u'v (UnApp Neg uv')) (BinApp Mul exp' exp')

     where

      uv' = (BinApp Mul exp (diff exp' var))

      u'v = (BinApp Mul (diff exp var) exp')

  diff (UnApp op exp) var

   | op == Neg = UnApp Neg (diff exp var)

   | op == Sin = BinApp Mul (UnApp Cos exp) (diff exp var)

   | op == Cos = UnApp Neg (BinApp Mul (UnApp Sin exp) (diff exp var))

   | op == Log = BinApp Div (diff exp var) exp

   | op == Ch = BinApp Mul (UnApp Sh exp) (diff exp var)

   | op == Sh = BinApp Mul (UnApp Ch exp) (diff exp var)

  maclaurin :: Exp -> Double -> Int -> Double

  maclaurin exp val n

   = sum (map (maclaurinEval val) (zipWith3 combine3 differentials factorials powers))

     where

      factorials = map fromIntegral (scanl (\*) 1 [1..(n-1)])

      differentials = take n (iterate ((flip diff) "x") exp)

      powers = [0..(n-1)]

      combine3 :: a -> b -> c -> (a, b, c)

      combine3 e1 e2 e3 = (e1, e2, e3)

      maclaurinEval :: Double -> (Exp, Double, Int) -> Double

      maclaurinEval val (exp, fact, pow)

       = eval exp [("x", 0)] / fact \* (val ^ pow)

  showExp :: Exp -> String

  showExp (Val val) = show val

  showExp (Id var) = var

  showExp (UnApp op exp) = show op ++ "(" ++ showExp exp ++ ")"

  showExp (BinApp op exp exp') = "(" ++ showExp exp ++ show op ++ showExp exp' ++ ")"

  isInt :: Double -> Bool

  isInt n

   | n < 1 = n == 0

   | otherwise = isInt (n - 1)

   ch x = ((exp x) + (exp ((-1) \* x))) / 2

   sh x = ((exp x) - (exp ((-1) \* x))) / 2

  ---------------------------------------------------------------------------

  -- Test cases from the spec.

  e1, e2, e3, e4, e5, e6 :: Exp

  -- > 5\*x

  e1 = BinApp Mul (Val 5.0) (Id "x")

  -- > x\*x + y - 7

  e2 = BinApp Add (BinApp Add (BinApp Mul (Id "x") (Id "x")) (Id "y"))

                  (UnApp Neg (Val 7.0))

  -- > x-y^2/(4\*x\*y-y^2)::Exp

  e3 = BinApp Add (Id "x")

              (UnApp Neg (BinApp Div (BinApp Mul (Id "y") (Id "y"))

              (BinApp Add (BinApp Mul (BinApp Mul (Val 4.0) (Id "x")) (Id "y"))

                          (UnApp Neg (BinApp Mul (Id "y") (Id "y"))))))

  -- > -cos x::Exp

  e4 = UnApp Neg (UnApp Cos (Id "x"))

  e41 = UnApp Neg (UnApp Ch (Id "x"))

  -- > sin (1+log(2\*x))::Exp

  e5 = UnApp Sin (BinApp Add (Val 1.0)

                             (UnApp Log (BinApp Mul (Val 2.0) (Id "x"))))

  -- > log(3\*x^2+2)::Exp

  e6 = UnApp Log (BinApp Add (BinApp Mul (Val 3.0) (BinApp Mul (Id "x") (Id "x")))

                             (Val 2.0))

  ----------------------------------------------------------------------

  -- EXTENSION: Uncomment and complete these...

  -- instance Num Exp where

  -- instance Fractional Exp where

  -- instance Floating Exp where

  -- instance (Eq a, Num a) => Num (Maybe a) where

  -- instance (Eq a, Fractional a) => Fractional (Maybe a) where

  -- diff2 :: Exp -> String -> Maybe Exp

  -- The following makes it much easier to input expressions, e.g. sin x, log(x\*x) etc.

  x, y :: Exp

  x = Id "x"

  y = Id "y"

**Результати:**



**Висновок**

Программа функціонує за планом, результат вірний.