19CSE313-Principles of Programming Languages

Lab Exercise-4

Done By:

Adharsh S Mathew
CSE-D
AM.EN.U4CSE19302

1.Define a function to find the largest of 3 numbers using if expression.

```haskell
-- 1
max' :: Int->Int->Int->Int
max' x y z = if(x>=y) && (x>=z) then x
             else if (y>=z) then y else z
```

```
(base) adharsh@adharsh-Inspiron-5570:~/Github/Haskell/Lab 4$ ghci
GHCi, version 8.6.5: http://www.haskell.org/ghc/  :? for help
Prelude> :load lab4.hs
[1 of 1] Compiling Main             ( lab4.hs, interpreted )
Ok, one module loaded.
*Main> max' 2 3 4
4
*Main> max' 2 5 4
5
*Main>
```

2. Define a function of type : **Int -> String** which reads a number and returns whether "even" or "odd".

```haskell
-- 2
even' :: Int->String
even' x = if x `mod` 2 == 0 then "even" else "odd"
```

```
*Main> :reload
[1 of 1] Compiling Main             ( lab4.hs, interpreted )
Ok, one module loaded.
*Main> even' 2
"even"
*Main> even' 3
"odd"
*Main>
```

3. Using **Guards**, determine the largest of two numbers.

```haskell
-- 3
max2 :: Int-> Int-> Int
max2 x y |(x > y) = x
         | otherwise = y
```

```
*Main> :reload
[1 of 1] Compiling Main             ( lab4.hs, interpreted )
Ok, one module loaded.
*Main> max2 5 6
6
*Main> max2 8 7
8
*Main>
```

4. Define a function distance to find the distance between two points in a xy-plane. Let P = (x_1, y_1) and Q = (x_2, y_2) , **[use where expression]**

$$PQ = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
-- 4
distance :: (Float,Float) -> (Float,Float) ->Float
distance (x1,y1) (x2,y2) = sqrt(a^2 + b^2)
                         where
                             a = (x2-x1)
                             b = (y2-y1)
```

```
*Main> :reload
[1 of 1] Compiling Main             ( lab4.hs, interpreted )
Ok, one module loaded.
*Main> distance (2,3) (4,5)
2.828427
*Main> distance (-25,10) (7.5,69)
67.359116
*Main>
```

5. Define the function **min** and **max** which work with values of arbitrary type, so long as this type is an instance of the **Ord** class. Check this function, by passing different type of values, like int, float, char, string.

min :: (Ord a)) a -> a -> a

```
-- 5

mIN :: Ord p => p -> p -> p
mIN x y = if(x<y) then x else y
mAX :: Ord p => p -> p -> p
mAX x y = if(x>y) then x else y
```

```
*Main> :reload
[1 of 1] Compiling Main
Ok, one module loaded.
*Main> mIN 5 10
5
*Main> mIN 7.89 7.90
7.89
*Main> mIN "a" "b"
"a"
*Main> mAX 5 10
10
*Main> mAX 7.89 7.90
7.9
*Main> mAX "a" "b"
"b"
*Main>
```

6. Define a function **divides**, **divides :: Int -> Int -> Bool** which, verifies whether the first argument divides the second one. Define this function using if expression, guarded expression and multiple definition using pattern matching

```
> 2 `divides` 3
False
> 0 `divides` 3
False
> 2 `divides` 4
True
```

```haskell
-- 6
divides :: Int->Int->Bool
divides x y = if(mod y x) == 0 then True else False
```

```
*Main> :reload
[1 of 1] Compiling Main             ( lab4.hs, interpreted )
Ok, one module loaded.
*Main> divides 2 4
True
*Main> divides 4 8
True
*Main> divides 2 3
False
*Main>
```

7. Implement a function in Haskell for the following mathematic function defined as, **[use pattern matching]**

$$f(x) = \begin{cases} 7 & \text{if } x = 0 \\ 3x^2 - 2 & \text{otherwise} \end{cases}$$

```haskell
-- 7
f :: Int-> Int
f 0 = 7
f x = 3*(x^2) - 2
```

```
*Main> :reload
[1 of 1] Compiling Main
Ok, one module loaded.
*Main> f 0
7
*Main> f 1
1
*Main> f 2
10
*Main>
```

8. Define a function to implement **Stirling's formula**

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

```haskell
--8
stirling ::Float  -> Float
stirling n = sqrt(2*pi*n) * ((n/exp(1))**n)
```

```
*Main> :reload
[1 of 1] Compiling Main
Ok, one module loaded.
*Main> stirling 5.0
118.01921
*Main> stirling 7.0
4980.3975
*Main>
```

9. Define a function **noOfSol** of type **:: Int -> Int -> Int -> String**, to find the number of solution of a quadratic equation.

```haskell
-- 9
noOfSol :: Int->Int->Int->String
noOfSol a b c
        | d  > 0 = "2 Real Solutions"
        | d == 0 = "1 Real Solution"
        | d <  0 = "No/ 2 Imaginnary Solutions"
        where
            d = (b^(2 :: Integer)) - (4 *a * c)
```

```
*Main> :reload
[1 of 1] Compiling Main
Ok, one module loaded.
*Main> noOfSol 3 (-10) (-25)
"2 Real Solutions"
*Main> noOfSol 2 (-4) 7
"No/ 2 Imaginnary Solutions"
*Main> noOfSol (-3) (-24) (-48)
"1 Real Solution"
*Main>
```

10. Define a function **rootsOfQuadraticEqu** of appropriate type, to find the two roots of a Quadratic equation. Given **a**, **b** and **c**, find the roots $\mathbf{x_1}$ and $\mathbf{x_2}$.

```haskell
-- 10
rootsOfQuadraticEqu :: Float -> Float -> Float-> (Float,Float)
rootsOfQuadraticEqu a b c
        | d < 0 = (0.0,0.0)
        | otherwise = (x1,x2)
        where
            x1 = ((-b) + d)/(2*a)
            x2 = ((-b) - d)/(2*a)
            d = sqrt((b^(2 :: Integer)) - (4 *a * c))
```

```
*Main> :reload
Ok, one module loaded.
*Main> rootsOfQuadraticEqu 3 (-10) (-25)
(5.0,-1.6666666)
*Main> rootsOfQuadraticEqu 2 (-4) 7
(NaN,NaN)
*Main> rootsOfQuadraticEqu (-3) (-24) (-48)
(-4.0,-4.0)
*Main>
```