LAB SHEET - 2

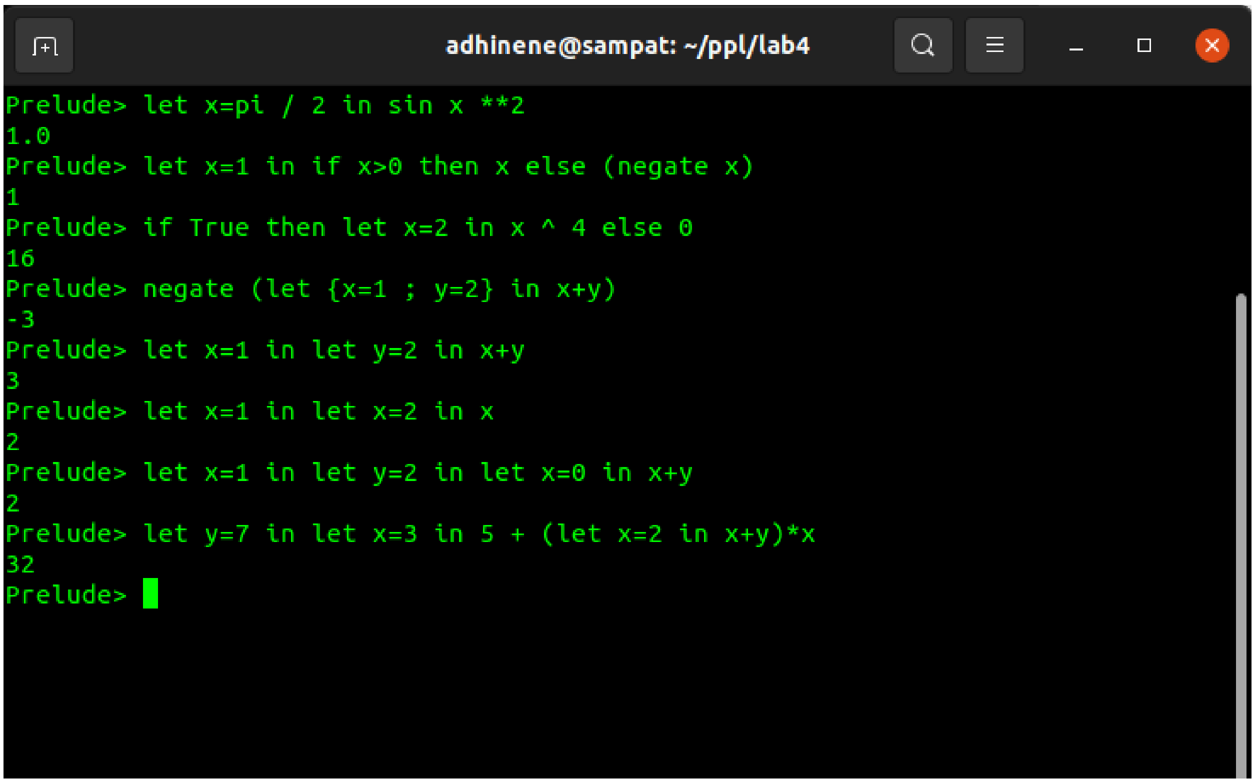
NAME : J.SAMPAT SRIVATSAV

ROLL NUMBER : AM.EN.U4CSE19125

BATCH : C.S.E - B

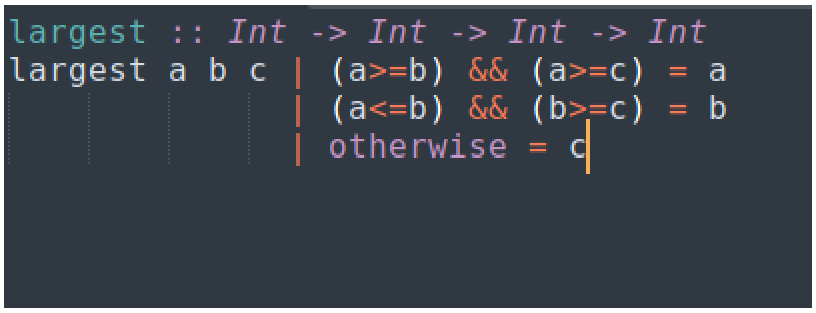
Try each expression in the terminal.

1. let x = pi / 2 in sin x ** 2
2. let x = 1 in if x > 0 then x else (negate x)
3. if True then let x = 2 in x ^ 4 else 0
4. negate (let { x = 1; y = 2 } in x + y)
5. let x = 1 in let y = 2 in x + y
6. let x = 1 in let x = 2 in x
7. let x = 1 in let y = 2 in let x = 0 in x + y
8. let y = 7 in let x = 3 in 5 + (let x = 2 in x + y) * x

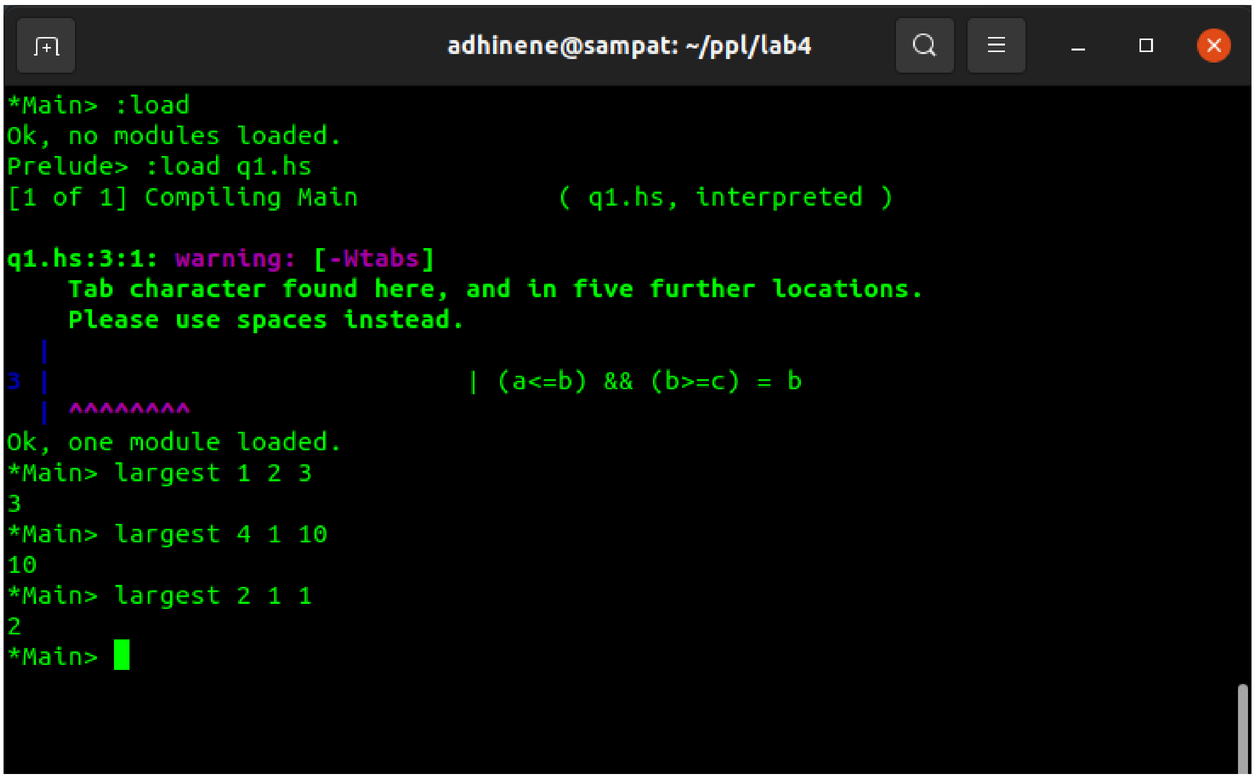


```
Prelude> let x=pi / 2 in sin x **2
1.0
Prelude> let x=1 in if x>0 then x else (negate x)
1
Prelude> if True then let x=2 in x ^ 4 else 0
16
Prelude> negate (let {x=1 ; y=2} in x+y)
-3
Prelude> let x=1 in let y=2 in x+y
3
Prelude> let x=1 in let x=2 in x
2
Prelude> let x=1 in let y=2 in let x=0 in x+y
2
Prelude> let y=7 in let x=3 in 5 + (let x=2 in x+y)*x
32
Prelude> 
```

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

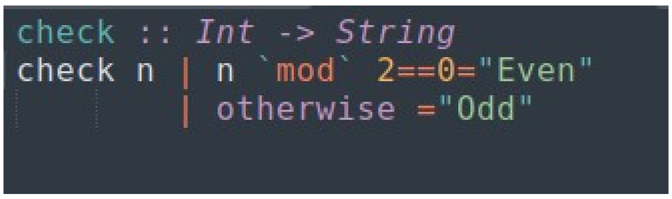


```haskell
largest :: Int -> Int -> Int -> Int
largest a b c | (a>=b) && (a>=c) = a
              | (a<=b) && (b>=c) = b
              | otherwise = c
```

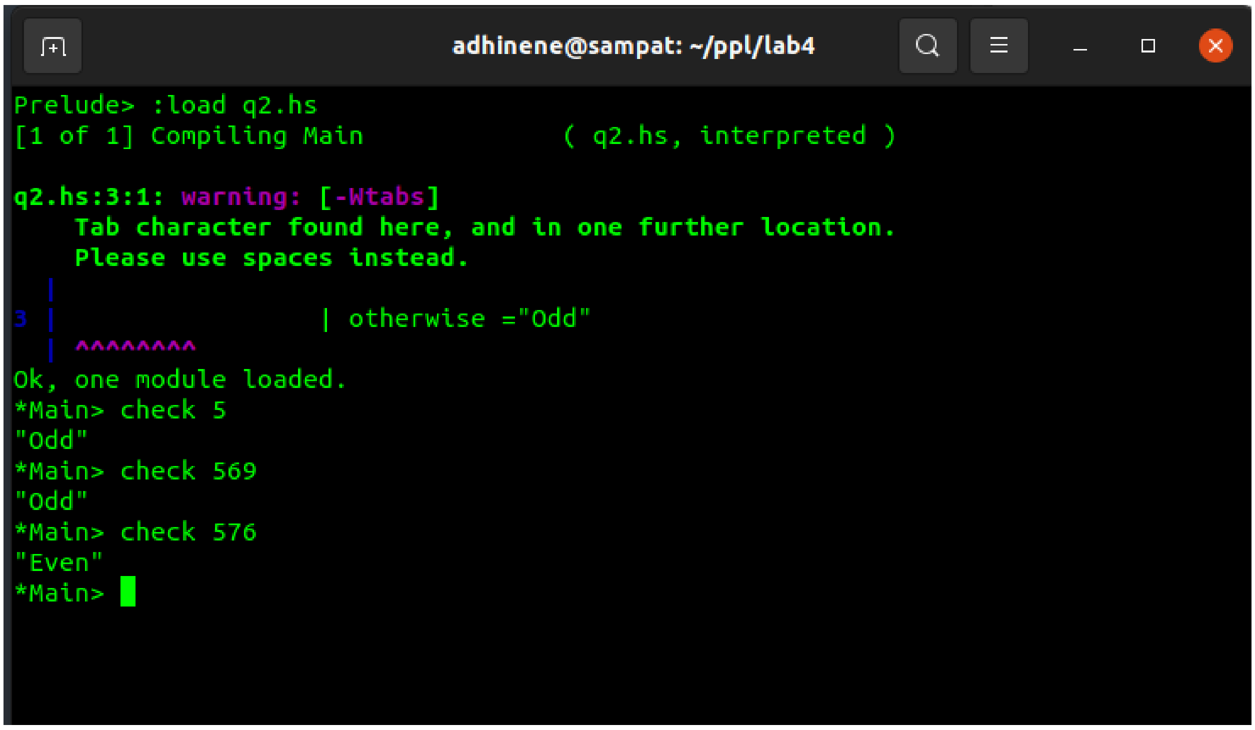


```
*Main> :load
Ok, no modules loaded.
Prelude> :load q1.hs
[1 of 1] Compiling Main             ( q1.hs, interpreted )

q1.hs:3:1: warning: [-Wtabs]
    Tab character found here, and in five further locations.
    Please use spaces instead.
  |
3 |                             | (a<=b) && (b>=c) = b
  | ^^^^^^^^
Ok, one module loaded.
*Main> largest 1 2 3
3
*Main> largest 4 1 10
10
*Main> largest 2 1 1
2
*Main> 
```

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

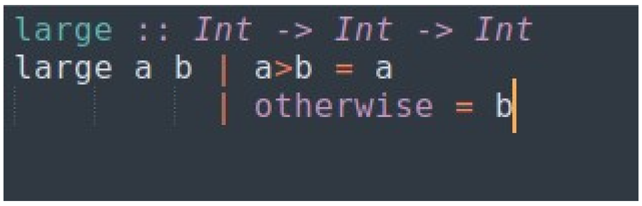


```haskell
check :: Int -> String
check n | n `mod` 2==0="Even"
        | otherwise ="Odd"
```

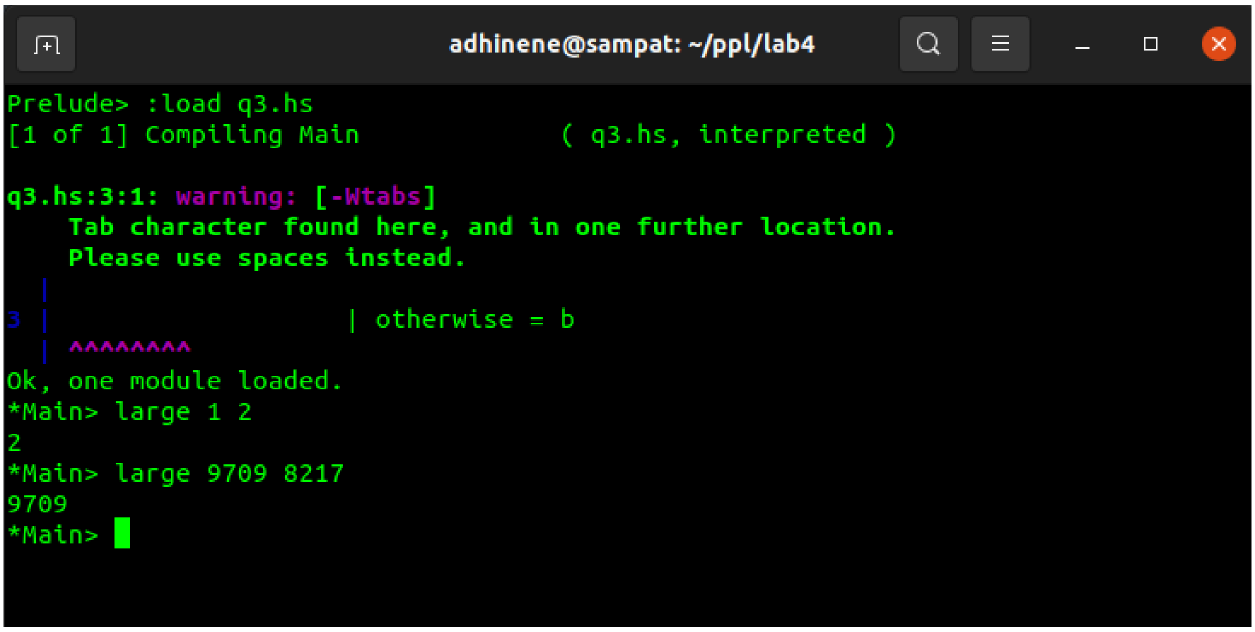


```
adhinene@sampat: ~/ppl/lab4
Prelude> :load q2.hs
[1 of 1] Compiling Main             ( q2.hs, interpreted )

q2.hs:3:1: warning: [-Wtabs]
    Tab character found here, and in one further location.
    Please use spaces instead.
  |
3 |                 | otherwise ="Odd"
  | ^^^^^^^^
Ok, one module loaded.
*Main> check 5
"Odd"
*Main> check 569
"Odd"
*Main> check 576
"Even"
*Main> 
```

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



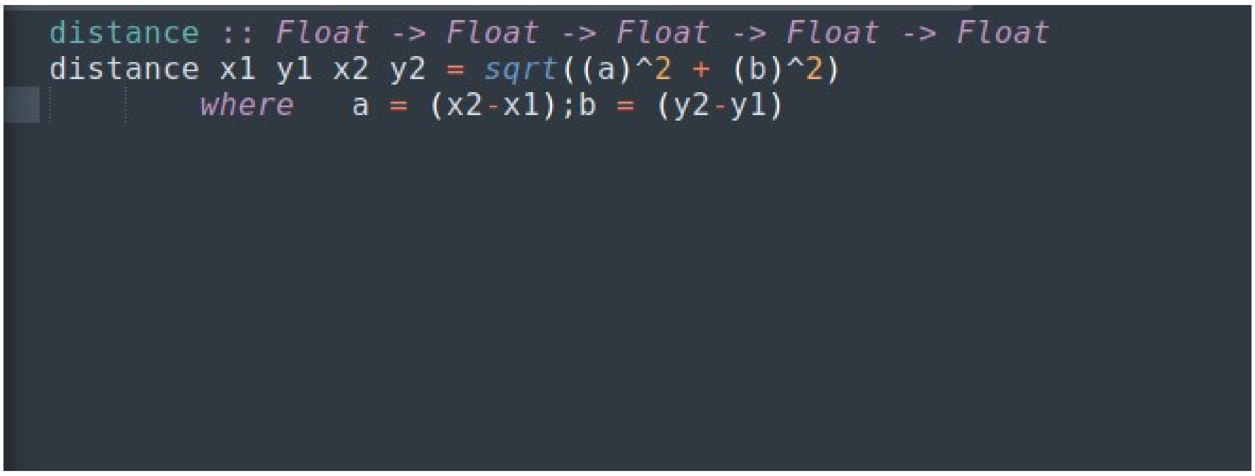
```haskell
large :: Int -> Int -> Int
large a b | a>b = a
          | otherwise = b
```



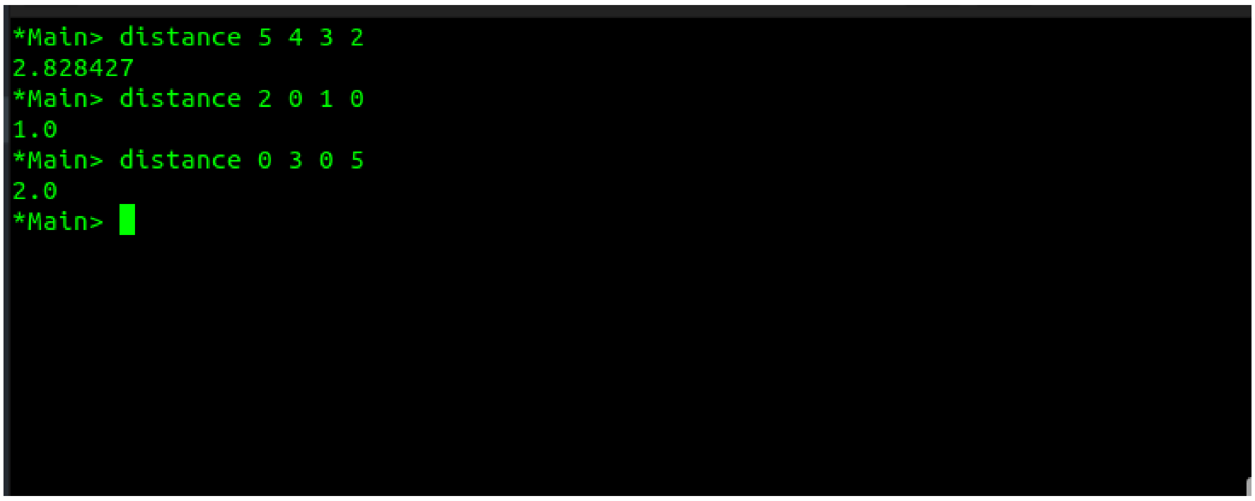
```
adhinene@sampat: ~/ppl/lab4
Prelude> :load q3.hs
[1 of 1] Compiling Main             ( q3.hs, interpreted )

q3.hs:3:1: warning: [-Wtabs]
    Tab character found here, and in one further location.
    Please use spaces instead.
  |
3 |                  | otherwise = b
  | ^^^^^^^^
Ok, one module loaded.
*Main> large 1 2
2
*Main> large 9709 8217
9709
*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]**

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



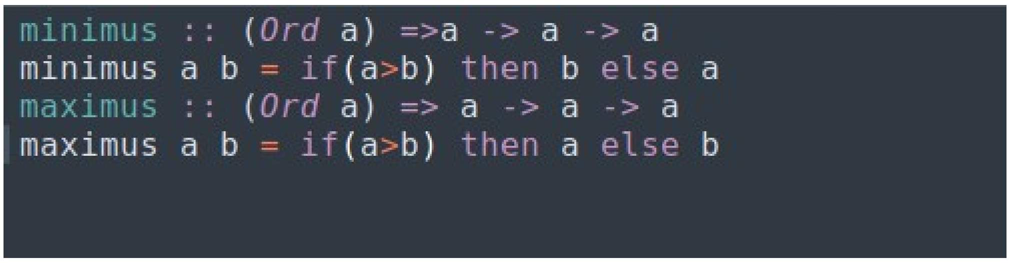
```
distance :: Float -> Float -> Float -> Float -> Float
distance x1 y1 x2 y2 = sqrt((a)^2 + (b)^2)
          where   a = (x2-x1);b = (y2-y1)
```



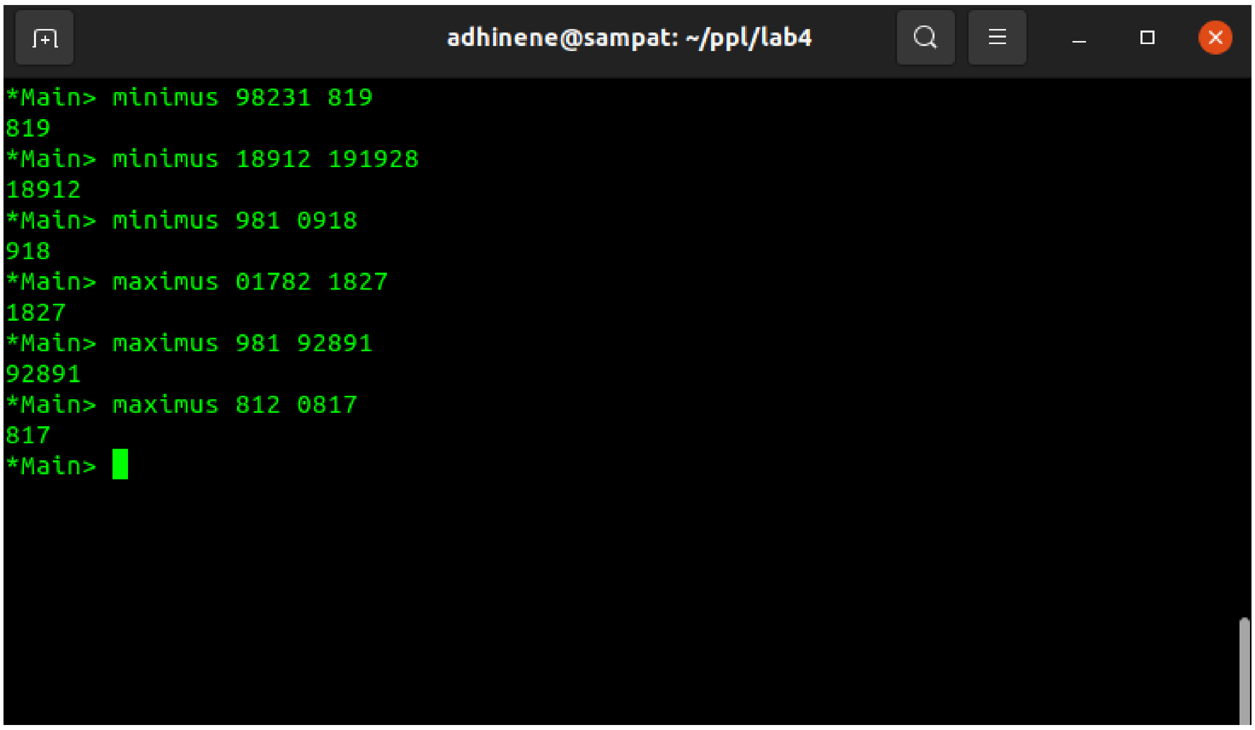
```
*Main> distance 5 4 3 2
2.828427
*Main> distance 2 0 1 0
1.0
*Main> distance 0 3 0 5
2.0
*Main>
```

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



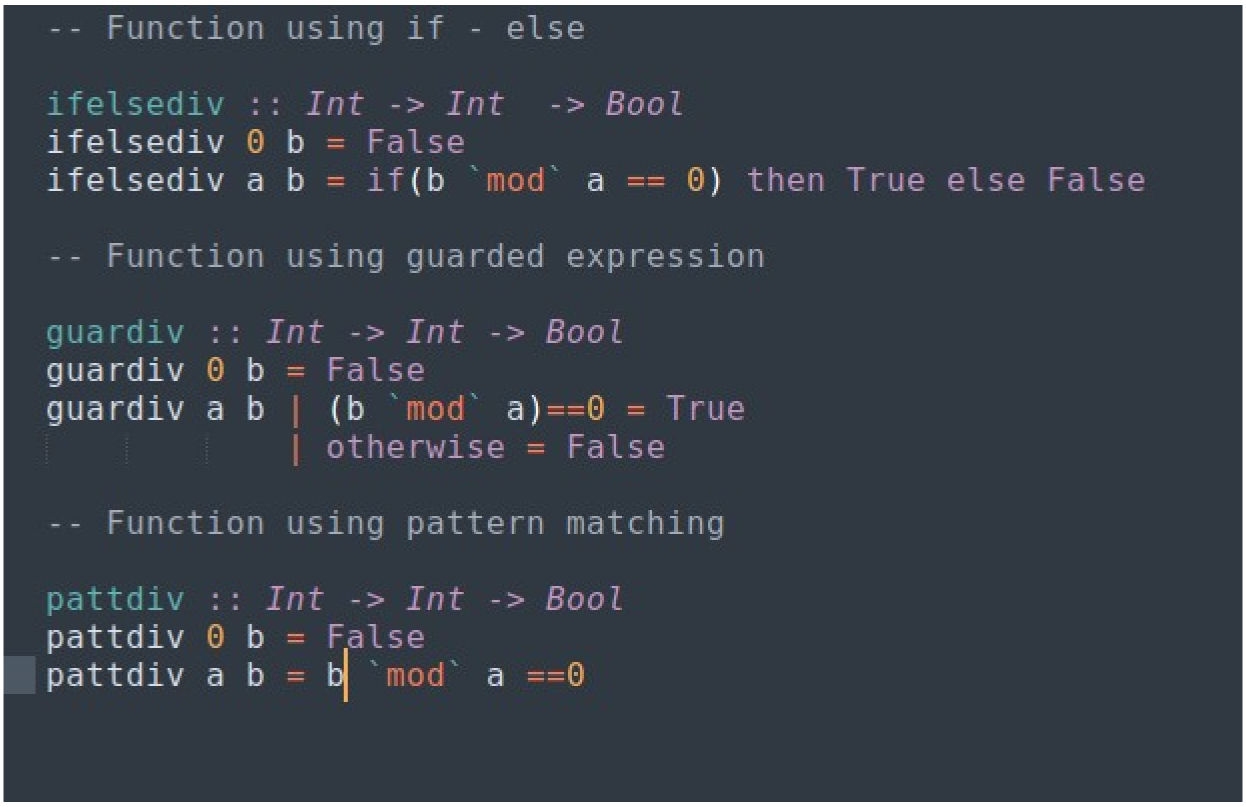
```
minimus :: (Ord a) =>a -> a -> a
minimus a b = if(a>b) then b else a
maximus :: (Ord a) => a -> a -> a
maximus a b = if(a>b) then a else b
```



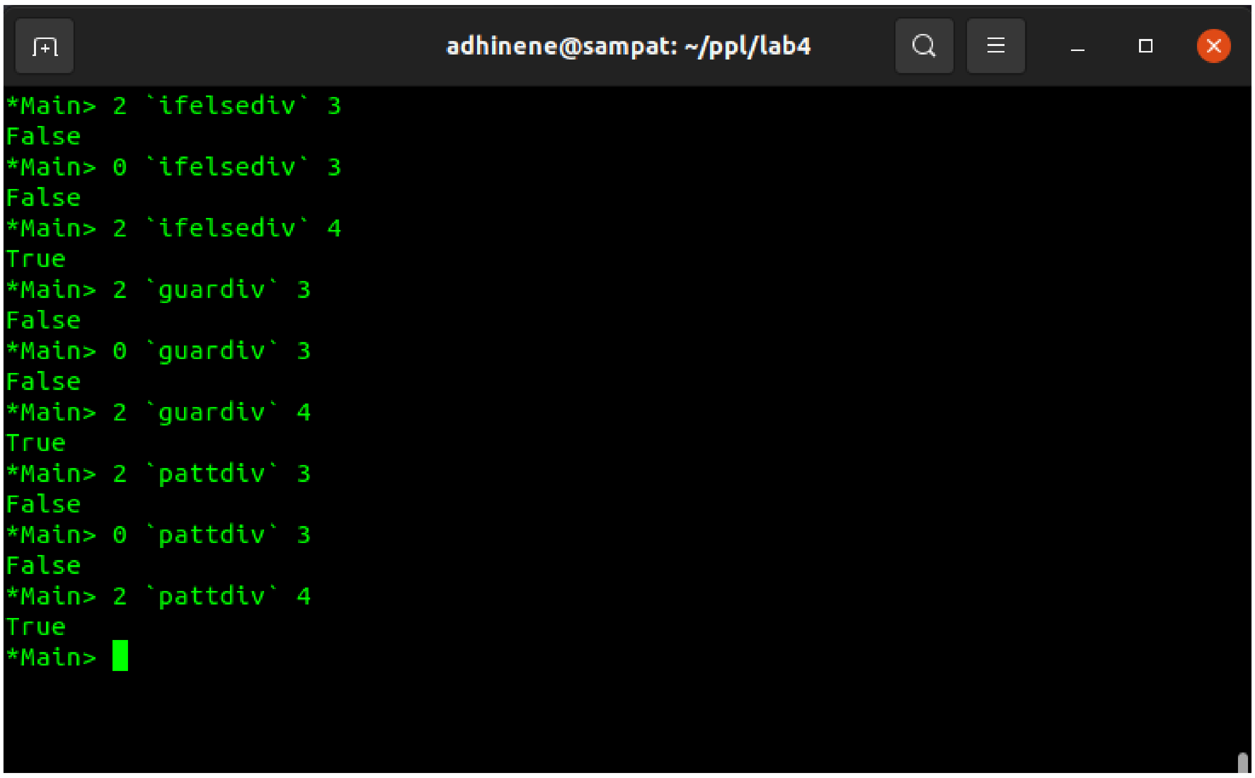
```
*Main> minimus 98231 819
819
*Main> minimus 18912 191928
18912
*Main> minimus 981 0918
918
*Main> maximus 01782 1827
1827
*Main> maximus 981 92891
92891
*Main> maximus 812 0817
817
*Main> 
```

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
```



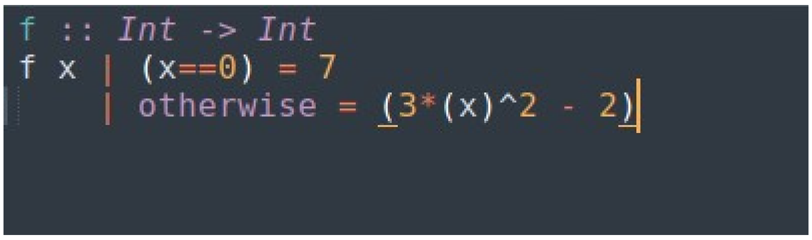
```
-- Function using if - else

ifelsediv :: Int -> Int  -> Bool
ifelsediv 0 b = False
ifelsediv a b = if(b `mod` a == 0) then True else False

-- Function using guarded expression

guardiv :: Int -> Int -> Bool
guardiv 0 b = False
guardiv a b | (b `mod` a)==0 = True
            | otherwise = False

-- Function using pattern matching

pattdiv :: Int -> Int -> Bool
pattdiv 0 b = False
pattdiv a b = b `mod` a ==0
```



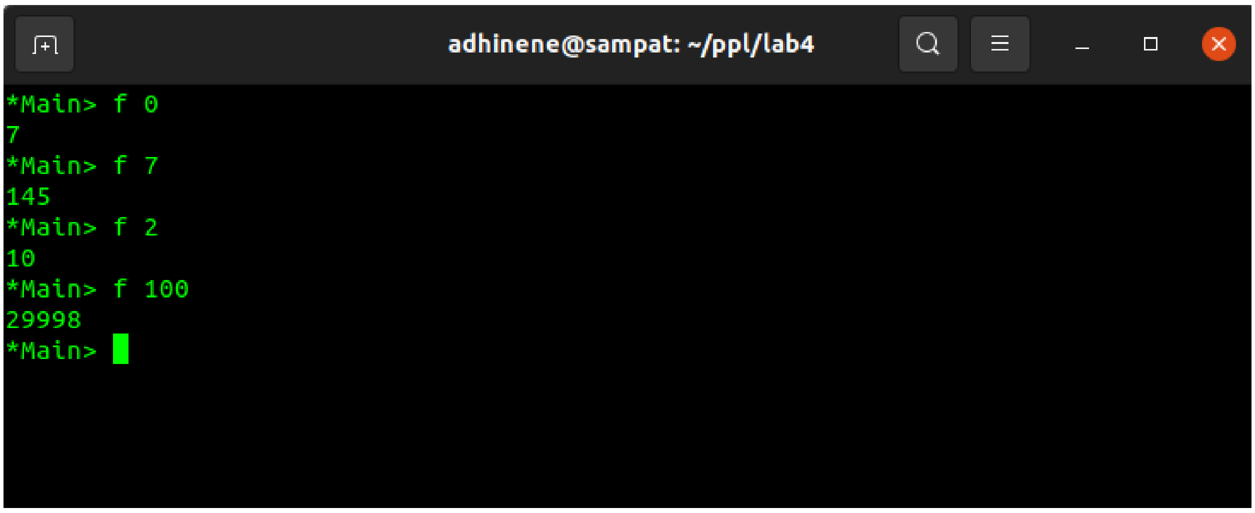
```
*Main> 2 `ifelsediv` 3
False
*Main> 0 `ifelsediv` 3
False
*Main> 2 `ifelsediv` 4
True
*Main> 2 `guardiv` 3
False
*Main> 0 `guardiv` 3
False
*Main> 2 `guardiv` 4
True
*Main> 2 `pattdiv` 3
False
*Main> 0 `pattdiv` 3
False
*Main> 2 `pattdiv` 4
True
*Main>
```

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

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



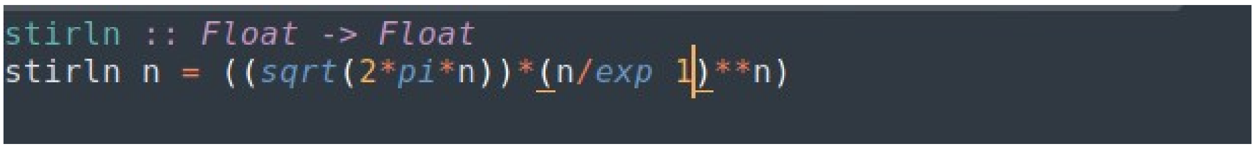
```haskell
f :: Int -> Int
f x | (x==0) = 7
    | otherwise = (3*(x)^2 - 2)
```



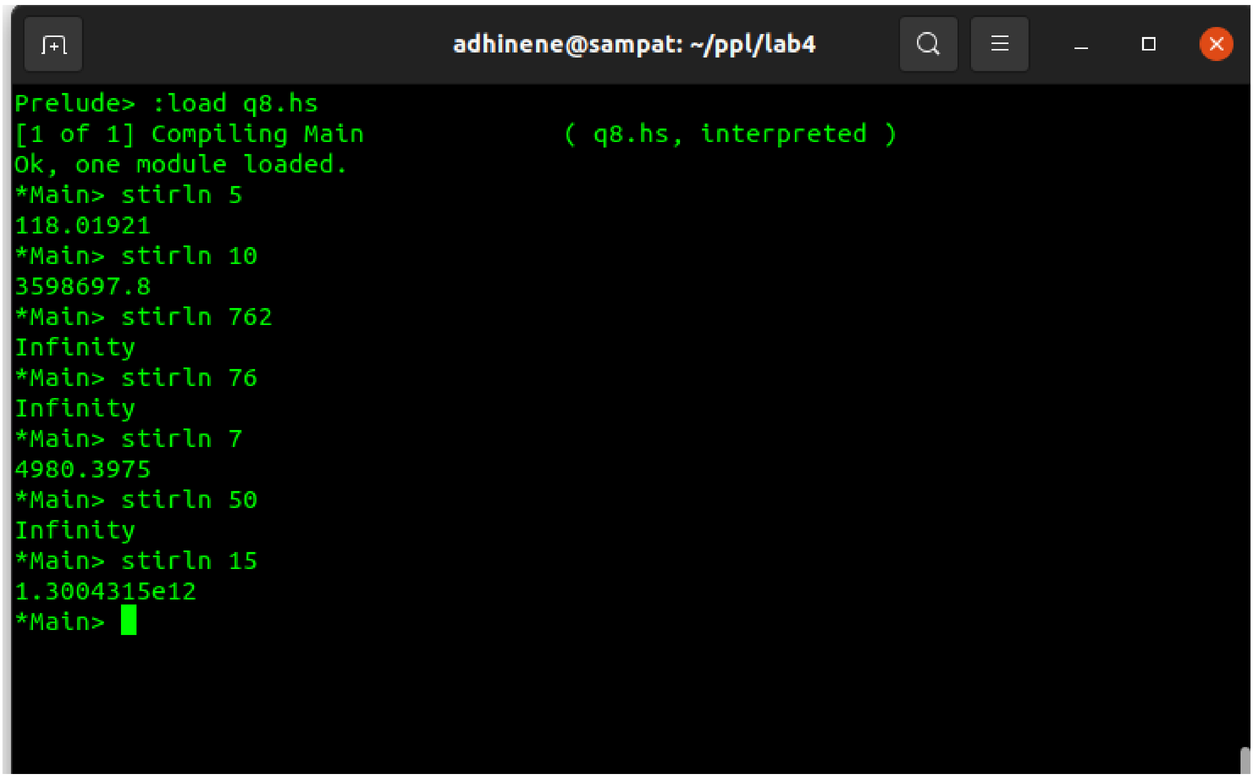
```
*Main> f 0
7
*Main> f 7
145
*Main> f 2
10
*Main> f 100
29998
*Main>
```

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

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

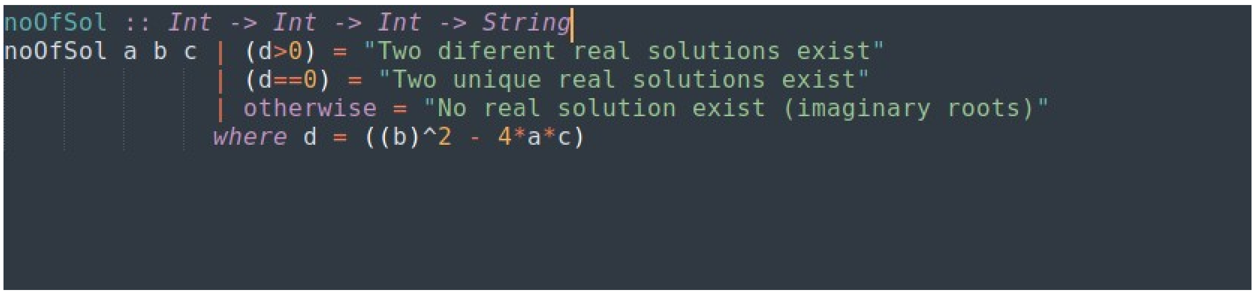


```haskell
stirln :: Float -> Float
stirln n = ((sqrt(2*pi*n))*(n/exp 1)**n)
```

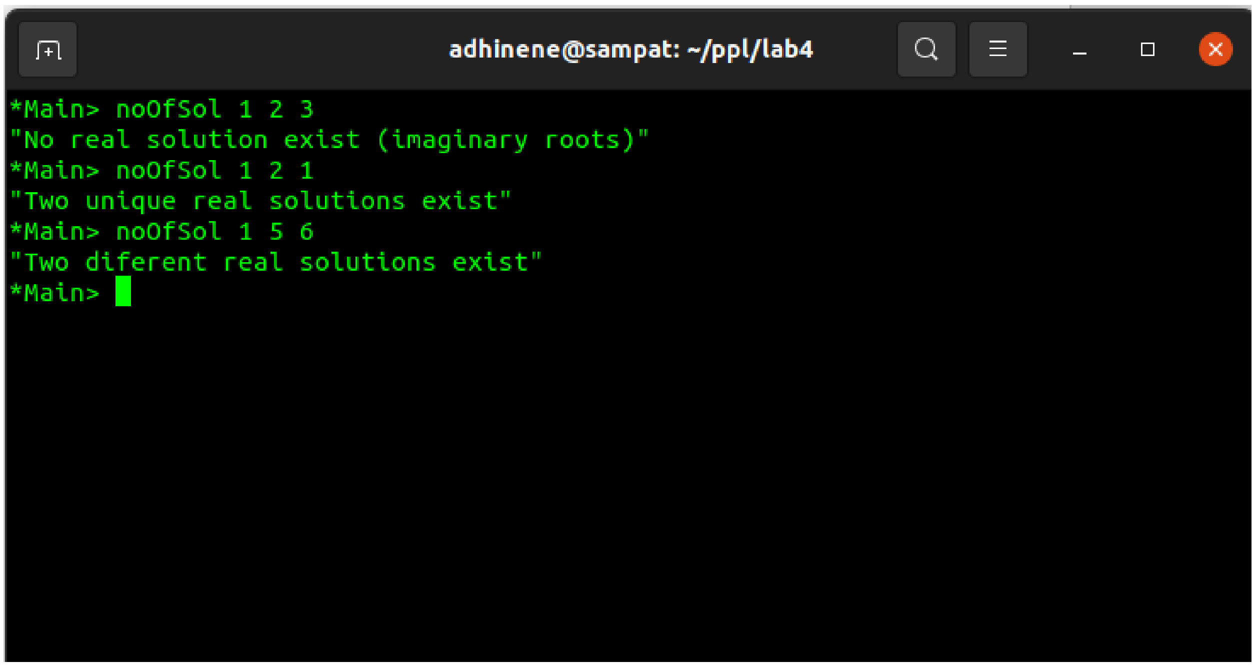


```
Prelude> :load q8.hs
[1 of 1] Compiling Main             ( q8.hs, interpreted )
Ok, one module loaded.
*Main> stirln 5
118.01921
*Main> stirln 10
3598697.8
*Main> stirln 762
Infinity
*Main> stirln 76
Infinity
*Main> stirln 7
4980.3975
*Main> stirln 50
Infinity
*Main> stirln 15
1.3004315e12
*Main>
```

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

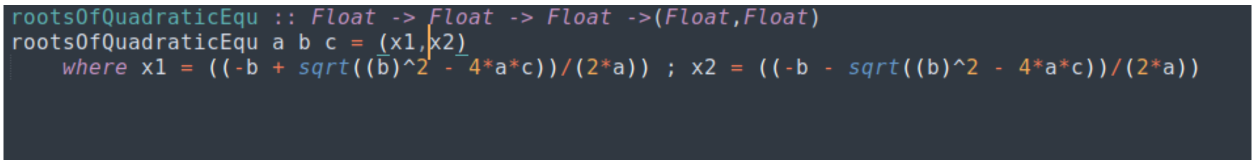


```haskell
noOfSol :: Int -> Int -> Int -> String
noOfSol a b c | (d>0) = "Two diferent real solutions exist"
              | (d==0) = "Two unique real solutions exist"
              | otherwise = "No real solution exist (imaginary roots)"
             where d = ((b)^2 - 4*a*c)
```

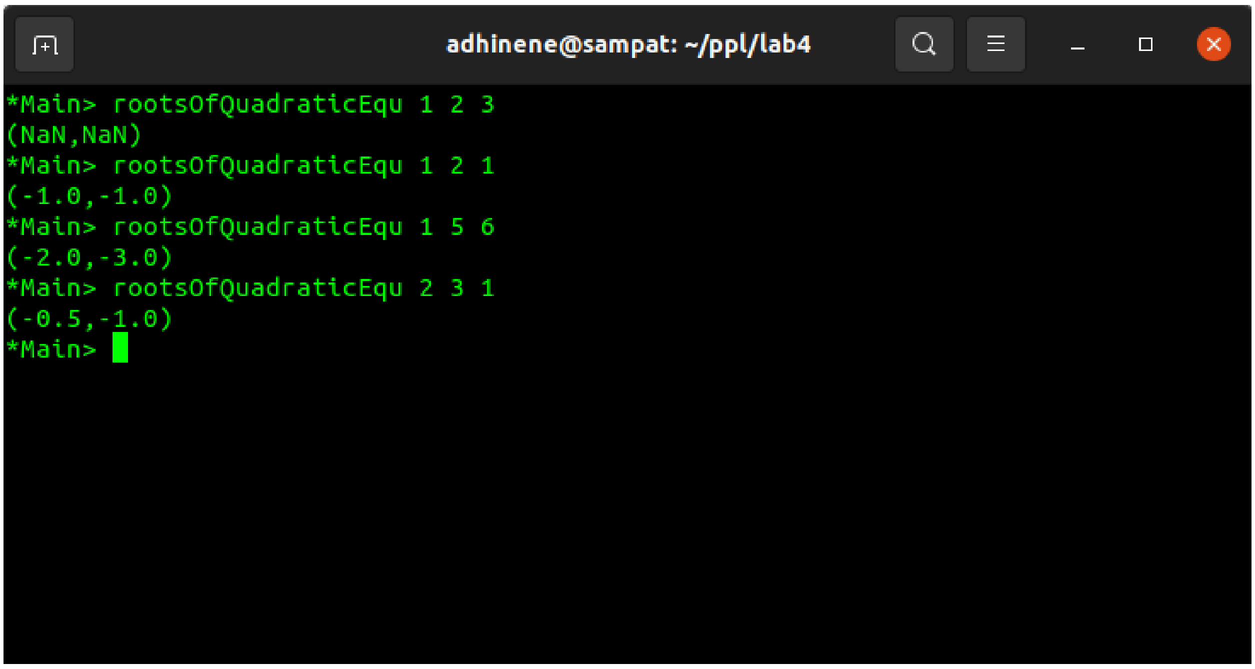


```
adhinene@sampat: ~/ppl/lab4
*Main> noOfSol 1 2 3
"No real solution exist (imaginary roots)"
*Main> noOfSol 1 2 1
"Two unique real solutions exist"
*Main> noOfSol 1 5 6
"Two diferent real solutions exist"
*Main>
```

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
rootsOfQuadraticEqu :: Float -> Float -> Float ->(Float,Float)
rootsOfQuadraticEqu a b c = (x1,x2)
    where x1 = ((-b + sqrt((b)^2 - 4*a*c))/(2*a)) ; x2 = ((-b - sqrt((b)^2 - 4*a*c))/(2*a))
```



```
adhinene@sampat: ~/ppl/lab4
*Main> rootsOfQuadraticEqu 1 2 3
(NaN,NaN)
*Main> rootsOfQuadraticEqu 1 2 1
(-1.0,-1.0)
*Main> rootsOfQuadraticEqu 1 5 6
(-2.0,-3.0)
*Main> rootsOfQuadraticEqu 2 3 1
(-0.5,-1.0)
*Main>
```