Basic Programs in Haskell

```
{- Comment -}

1. Hello.hs

main = putStrLn "Hello, World!" {- "do" not required for single line code block -}

{- o/p  Hello, World!
-}
```

****

Computational Programs in Haskell

```
2. arith.hs

main = do  {- "do" required for multiline line code block -}
        let x = 10
        let y = 3
        putStr"The addition of 10 and 3 is : "
        print(x + y)
        putStr"The subtraction of 10 and 3 is : "
        print(x - y)
        putStr"The product of 10 and 3 is : "
        print(x * y)
        putStr"The division of 10 and 3 is : "
        print(x / y)

{- o/p  The addition of 10 and 3 is : 13.0
        The subtraction of 10 and 3 is : 7.0
        The product of 10 and 3 is : 30.0
        The division of 10 and 3 is :  3.3333333333333335
-}
```

```
3. areatri.hs

main = do
        putStrLn("Enter the magnitude of 3 sides of a triangle")
        ain <- getLine
        let a = (read ain :: Float)
        bin <- getLine
        let b = (read bin :: Float)
        cin <- getLine
        let c = (read cin :: Float)
        let s = (a+b+c)/2
        let ar = sqrt(s*(s-a)*(s-b)*(s-c))
        putStr("Area in sq units is : ")
        print(ar)

{- o/p  Enter the magnitude of 3 sides of a triangle
        5
        6
        3
        Area in sq units is : 7.483315
-}
```

```
4. fact::Int->Int {- Declaration of factorial function -}
{- | operator used for binding condition -}
fact n|n==0=1 {- Definition of factorial function for termination of recursion -}
fact n|n/=0=n*fact(n-1) {- Definition of factorial function for general case -}

main=do
        num <- getLine
        let n = (read num :: Int)
        print(fact n)

{- o/p  5
        120
-}
```

****

5. joinlist.hs

```
main = print([1,2,3,0] ++ [5,9,3])

{-  o/p [1,2,3,0,5,9,3]
-}
```

---

6. Summation.hs

```
main = print(sum [1,2,3,0,-5,9,-3.6])

{- o/p 6.4
-}
```

---

7. mult.hs {- Two definitions for recursion -}

```
mult [] = 1  {- Definitions for termination of recursion -}
mult (n:ns) = n * mult ns {- Multiply head with result of multiplication of tail -}

main = print(mult [1,2,3,-5,9,-3.6])

{-  o/p 972.0
-}
```

---

8. sqrlst.hs

```
sqr [] = []
sqr (x:xs) = [x*x] ++ sqr xs

main = print( sqr [1,5,10] )

{- o/p [1,25,100]
-}
```

---

9. prod2lst.hs

```
prodtwo [] [] z = reverse z
prodtwo (n:ns) (m:ms) z = prodtwo ns ms ((n*m):z)

main = print(prodtwo [1,2,3] [-5,9,-3] [])

{- o/p [-5,18,-9]
-}
```

---

10. lengthlst.hs

```
len [] = 0
len (x:xs) = 1 + len xs

main = print( len [1,2,3,4,-6,0,-2] )

{- o/p 7
-}
```

****

Computational Programs in Haskell

11. and1.hs

```
and1::Bool->Bool->Bool

and1 x y|x==True && y==True=True
        |otherwise=False

main = do
          putStr( "False AND False = " )
          print( and1 False False )
          putStr( "False AND True  = " )
          print( and1 False True )
          putStr( "True  AND False = " )
          print( and1 True False )
          putStr( "True  AND True  = " )
          print( and1 True True )

{- o/p  False AND False = False
          False AND True  = False
          True  AND False = False
          True  AND True  = True
-}
```

---

12. or1.hs

```
or1::Bool->Bool->Bool


or1 x y|x==False && y==False = False
       |otherwise = True

main = do
          putStr( "False OR False = " )
          print( or1 False False )
          putStr( "False OR True  = " )
          print( or1 False True )
          putStr( "True  OR False = " )
          print( or1 True False )
          putStr( "True  OR True  = " )
          print( or1 True True )

{- o/p  False OR False = False
          False OR True  = True
          True  OR False = True
          True  OR True  = True
-}
```

---

13. xor1.hs

```
xor1::Bool->Bool->Bool


xor1 x y|x==True && y== False=True
        |x==False && y==True=True
        |otherwise=False

main = do
          putStr( "False XOR False = " )
          print( xor1 False False )
          putStr( "False XOR True  = " )
          print( xor1 False True )
          putStr( "True  XOR False = " )
          print( xor1 True False )
          putStr( "True  XOR True  = " )
          print( xor1 True True )


{- o/p  False XOR False = False
          False XOR True  = True
          True  XOR False = True
          True  XOR True  = False
-}
```