Recursive Functions

1. Factorial of a number

factorial :: Int -> Int

factorial 0 = 1

factorial n = n \* factorial (n-1)

main = do

let result = factorial 5

print result

1. Fibonacci Series

fib :: Int -> Int

fib 0 = 0

fib 1 = 1

fib n = fib (n-1) + fib (n-2)

main = do

let result = fib 10

print result

1. Sum of elements in a list

sumList :: [Int] -> Int

sumList [] = 0

sumList (x:xs) = x + sumList xs

main = do

let result = sumList [1,2,3,4,5]

print result

1. Sum of Natural Numbers

sumOfNaturalNumbers :: Integer -> Integer

sumOfNaturalNumbers 0 = 0

sumOfNaturalNumbers n = n + sumOfNaturalNumbers (n - 1)

main :: IO ()

main = do

let n = 10

print (sumOfNaturalNumbers n)

Filter

filter: Take only elements of a list that meet some condition.

Example:

Expression Result

filter even [1..8] [2,4,6,8]

filter isPrime [2..10] [2,3,5,7]

1. Write a Haskell script to find the product of the numbers in a list.

productList :: [Int] -> Int

productList [] = 1

productList (x:xs) = x \* (productList xs)

main ::IO()

main = do

putStrLn "Enter list elements : "

n <- getLine

let arr = map read (words n) :: [Int]

let ans = productList arr

putStrLn $ "The product of all elements in the list is " ++ show ans

1. Write a Haskell script to reverse the elements in a list using recursion.

rev :: [String] -> [String]

rev (x:xs) = do

if null xs

then x:xs

else

rev xs ++ [x]

main :: IO()

main = do

putStrLn "Enter the list elements : "

a <- getLine

let arr = ( words a ) :: [String]

let b = rev arr

putStrLn ("Reversed list : " ++ show b )

1. Write a Haskell script to illustrate the use of zip function on multiple lists in Haskell.

main :: IO ()

main = do

let a = [1,2,3]::[Int]

let b = ["Jan","feb","march"]::[String]

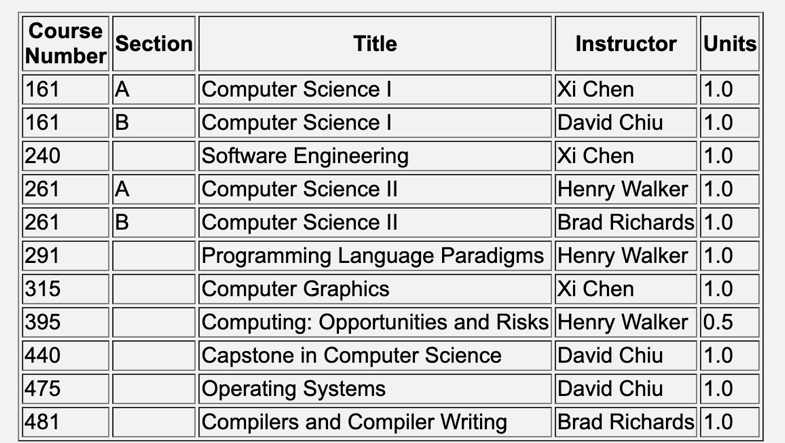
putStrLn $ "\nArray 1 : " ++ show a

putStrLn $ "Array 2 : " ++ show b

putStrLn $ "Zip of both the lists: " ++ show (zip a b)

1. Program based on tuples

An abbreviated version of this semester's computer science offerings is shown in the following table:



1. Translate this table into a list of courses, where each course is represented by a Haskell tuple with five components.
2. Write a Haskell function that has two parameters:

a course number (an integer)

the list of courses from part a.

and returns a list of all course tuples with the given course number.

1. Implement quicksort based on recursive functions.

quick :: (Ord a)=> [a] -> [a]

quick [] = []

quick (x:xs) =

let smallerSorted = quick [a | a <- xs, a<=x]

biggerSorted = quick [a | a <- xs, a>x]

in smallerSorted ++ [x] ++ biggerSorted

main::IO()

main = do

putStrLn "Enter the list elements : "

a <- getLine

let arr = map read ( words a ) :: [Int]

let b = quick arr

putStrLn ("Quicksorted list : " ++ show b )

1. Illustrate the use of filter functions.

isEven n = n `mod` 2 == 0

isOdd n = n `mod` 2 == 1

main::IO()

main = do

let arr = [1,2,3,4,5,6,7,8,9,10]

putStrLn $ "list = " ++ show arr

let evenArr = filter isEven arr

putStrLn $ "even numbers = " ++ show evenArr

let oddArr = filter isOdd arr

putStrLn $ "odd numbers = " ++ show oddArr