PAIR PROGRAMMING
M.DARSHAN - CB.EN.U4CSE19126
MATHAN KARTHICK - CB.EN.U4CCSE19130

PROBLEM STATEMENT - IMPLEMENT A LINKED LIST USING HASKELL AND SCALA

HASKELL CODE:

```
data MyList a = Cons a (MyList a)
              | MyNil deriving (Show, Eq)

{-
data - polymorpic userdefined variable
Cons - Constructor name
-}

myHead :: MyList a -> a
myHead l = case l of
      Cons a _ -> a

myTail :: MyList a -> MyList a
myTail MyNil = MyNil
myTail l = case l of
      Cons _ a -> a

myIndex :: Int -> MyList a -> a
myIndex 0 xs = myHead xs
myIndex x xs = myHead (myIndexTail x xs)
   where
      myIndexTail 0 xs = xs
      myIndexTail i xs = myIndexTail (i-1) (myTail xs)

myLength :: MyList a -> Int
myLength MyNil = 0
myLength xs = 1 + (myLength (myTail xs))

myLast :: MyList a -> a
myLast (Cons a MyNil) = a
myLast l = myLast (myTail l)

myInsert :: a -> MyList a -> MyList a
myInsert x xs = Cons x xs
```

```
myConcat :: MyList a -> MyList a -> MyList a
myConcat (Cons a MyNil) bs = Cons a bs
myConcat as bs = myInsert (myHead as) (myConcat (myTail as) bs)

myAppend :: a -> MyList a -> MyList a
myAppend x (Cons a MyNil) = Cons a (Cons x MyNil)
myAppend x xs = myInsert (myHead xs) (myAppend x (myTail xs))

myToList :: MyList a -> [a]
myToList MyNil = []
myToList (Cons a l) = a:(myToList l)

myFromList :: [a] -> MyList a
myFromList [] = MyNil
myFromList l = Cons (head l) (myFromList (tail l))

myMapList :: (t -> a) -> MyList t -> MyList a
myMapList f (Cons x MyNil) = Cons (f x) MyNil
myMapList f l = Cons (f (myHead l)) (myMapList f (myTail l))
```

OUTPUT:

```
PS D:\PPL\Assignment-3\Project> ghci
GHCi, version 9.2.1: https://www.haskell.org/ghc/  :? for help
ghci> :l ll.hs
[1 of 1] Compiling Main             ( ll.hs, interpreted )
Ok, one module loaded.
ghci> mylist = (Cons 10 (Cons 99 (Cons 11 (Cons 1 MyNil))))
ghci> myHead mylist
10
ghci> myTail mylist
Cons 99 (Cons 11 (Cons 1 MyNil))
ghci> myIndex 3 mylist
1
ghci> myLength mylist
4
ghci> myLast mylist
1
ghci> myInsert 50 mylist
Cons 50 (Cons 10 (Cons 99 (Cons 11 (Cons 1 MyNil))))
ghci> myAppend 16 mylist
Cons 10 (Cons 99 (Cons 11 (Cons 1 (Cons 16 MyNil))))
ghci> myToList mylist
[10,99,11,1]
```

```
ghci> myMapList (\x->x+100) mylist
Cons 110 (Cons 199 (Cons 111 (Cons 101 MyNil)))
```

SCALA CODE:

App.scala:

```
import scala.util.control.Breaks

case class LinkedList[T]() {
  var head: Node[T] = null;

  def push(data: T) = {
    head match {
      case null => head = new Node(data, null)

      case _ => {
        var last: Node[T] = head;

        while (last.next != null) {
          last = last.next;
        }

        last.next = new Node[T](data, null);
      }
    }
  }

//higher order function since it uses another function
//currying is done in order to reduce complexity
  def append(data:T) = {
    push(data);
  }

  def prepend(data:T): Unit = {
    val tempHead:Node[T]  = new Node[T](data,head);
    head = tempHead;
  }

  def print() = {
    if (head != null) {
      head.printList();
    }
    println();
  }
```

```
def delete(deleteItem: T) = {
  var previousNode: Node[T] = head
  var currentNode: Node[T] = head

  val loopBreak = new Breaks;

  loopBreak.breakable {
    while (currentNode != null) {
      if (currentNode.data.equals(deleteItem)) {
        if (currentNode.equals(previousNode)) {
          head = currentNode.next
        } else {
          previousNode.next = currentNode.next;

        }
        loopBreak.break();
      } else {
        previousNode = currentNode;
        currentNode = currentNode.next;
      }
    }
  }

}

def reverse(): Unit = {
  var previous:Node[T] = null;
  var current:Node[T] = head;
  var next:Node[T] = null;

  while (current != null) {
    next = current.next;
    current.next = previous;
    previous = current;
    current = next;
  }
  head = previous;
}

def getDataByIndex(index: Int): T = {
  var currentNode = head
  var currentIndex = 0;
```

```
    while (!currentIndex.equals(index)) {
      currentNode = currentNode.next
      currentIndex += 1;
    }

    currentNode.data;
  }

}

sealed case class Node[T](var data: T, var next: Node[T]) {
  def getData: T = this.data

  def getNext: Node[T] = this.next;

  def printList(): Unit = {
    print(data)

    if (next != null) {
      print(",")
      next.printList();
    }

  }

}
```

main.scala

```
object Main extends App {

  var list: LinkedList[Int] = new LinkedList();

  list.push(1);
  list.push(2);
  list.push(3);
  list.print()
  list.delete(1);
  list.print();
  list.reverse();
  list.print();
  println(list.getDataByIndex(1));
  list.prepend(23);
  list.print();
```

```
  list.reverse();
  list.print();

}
```

OUTPUT: