Funkcionalno programiranje

Vežbe

06 Liste

Zadatak 6.1

- Napisati implementaciju metode `:::` koja vrši konkatenaciju dve generičke liste.

- Analiza:
  - svi operatori koji se završavaju sa `:` grupišu se s desna ulevo
  - ```
    List(1, 2) ::: List(3, 4)
    = List(3, 4).:::(List(1, 2))
    = List(1, 2, 3, 4)
    ```
  - Dakle: parametar funkcije je zapravo prefiks rezultujuće liste

  - S obzirom na to da je lista kovarijantna, treba obezbediti da se u listu mogu nadovezivati liste nadtipova
    ```

Zadatak 6.1
Rešenje 1 – ručno napravljena lista

```
object lists {
  trait List[T] {
    def prazna: Boolean
    def glava: T
    def rep: List[T]
    def concat(l2 : List[T]) : List[T] = this match {
      case Nil() => l2
      case Elem(h, t) => Elem(h, t concat l2)
    }
    def :::(l2: List[T]) : List[T] = {
      l2.concat(this)
    }
  }

  case class Nil[T]() extends List[T] {
    def prazna = true
    def glava = throw new NoSuchElementException("Nil.glava")
    def rep = throw new NoSuchElementException("Nil.rep")
  }
```

Zadatak 6.1
Rešenje 1 – ručno napravljena lista

```
  case class Elem[T](val glava : T, val rep : List[T])
       extends List[T]  {
    def prazna = false
  }

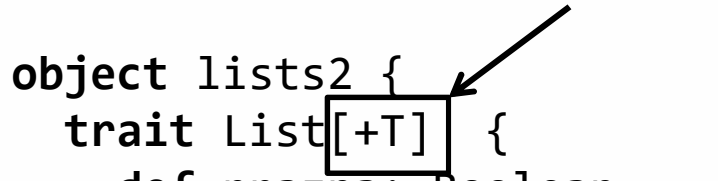
  val l1 = Elem(1, Nil()) //> l1:w06.lists.Elem[Int] = Elem(1,Nil())
  val l2 = Elem(2, Nil()) //> l2:w06.lists.Elem[Int] = Elem(2,Nil())

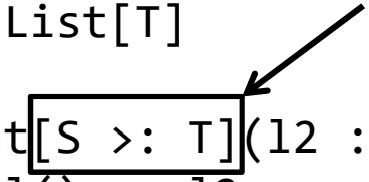
  l1.concat(l2)  //> res0:w06.lists.List[Int] = Elem(1,Elem(2,Nil()))

  l1 ::: l2      //> res1:w06.lists.List[Int] = Elem(1,Elem(2,Nil()))

  val l3 = Elem('a', Nil())
                 //> l3 : w06.lists.Elem[Char] = Elem(a,Nil())
  l1 ::: l3      // ????
}
```

Zadatak 6.1
Rešenje 2 – podrška za liste srodnih tipova





```
object lists2 {
  trait List[+T]  {
    def prazna: Boolean
    def glava: T
    def rep: List[T]

    def concat[S >: T](l2 : List[S]) : List[S] = this match {
      case Nil() => l2
      case Elem(h, t) => Elem(h, t concat l2)
    }
    def :::[S >: T](l2: List[S]) : List[S] =  {  l2 concat this  }
  }

  case class Nil[T]() extends List[T]  {
    def prazna = true
    def glava = throw new NoSuchElementException("Nil.glava")
    def rep = throw new NoSuchElementException("Nil.rep")
  }
```

Zadatak 6.1
Rešenje 2 – podrška za liste srodnih tipova

```
  case class Elem[T](val glava : T, val rep : List[T])
  extends List[T]  {    def prazna = false   }

  val l1 = Elem( 1, Nil() )
             //> l1:w06.lists2.Elem[Int] = Elem(1,Nil())
  val l2 = Elem( 'a', Nil() )
             //> l2:w06.lists2.Elem[Char] = Elem(a,Nil())

  l1 ::: l2  //> res1:w06.lists2.List[AnyVal] = Elem(1,Elem(a,Nil()))

  l2 ::: l1  //> res2:w06.lists2.List[AnyVal] = Elem(a,Elem(1,Nil()))

  val l3 = Elem( Nil(), Nil() )
  //> l3:w06.lists2.Elem[w06.lists2.Nil[Nothing]] = Elem(Nil(),Nil())

  l1 ::: l3
  //> res3: w06.lists2.List[Any] = Elem(1,Elem(Nil(),Nil()))
}
```

Zadatak 6.1
Rešenje 3 – `scala.List[T]`

```
sealed abstract class List[+A] extends AbstractSeq[A]
                              with LinearSeq[A]
                              with Product
                              with GenericTraversableTemplate[A, List]
                              with LinearSeqOptimized[A, List[A]]
                              with Serializable {
  ...
  def isEmpty: Boolean
  def head: A
  def tail: List[A]

  def ::[B >: A] (x: B): List[B] =
    new scala.collection.immutable.::(x, this)

  def :::[B >: A](prefix: List[B]): List[B] =
    if (isEmpty) prefix
    else if (prefix.isEmpty) this
    else (new ListBuffer[B] ++= prefix).prependToList(this)
  ...
```

Zadatak 6.2

- Napisati implementaciju funkcije `take(List[T], n)`, čiji rezultat je nova lista formirana od prvih n elemenata generičke liste.Ako je n veće od dužine liste, rezultat je sama lista.

Zadatak 6.2

```
def take[T](l : List[T], n : Int) : List[T] =
{
  def takea[T](l : List[T], n : Int) : List[T] =
    if( n > 0 )  l.head :: takea(l.tail, n-1)
    else Nil

  if( n > l.length ) l
  else takea(l, n)
}

val lista = 1 :: 2 :: 3 :: 5 :: 4 :: 8 :: 6 :: Nil

take(lista, 4)              //> res1: List[Int] = List(1, 2, 3, 5)
```

Zadatak 6.3

- Napisati implementaciju funkcije `drop(List[T], n)`, čiji rezultat je nova lista formirana izostavljajući prvih n elemenata generičke liste. Ako je n veće od dužine liste, rezultat je prazna lista.

Zadatak 6.3

```
def drop[T](l : List[T], n : Int) : List[T] =
{
  def dropa[T](l : List[T], n : Int) : List[T] =
      if( n > 0 ) dropa(l.tail, n-1 )
      else l

  if( n > l.length ) List()
  else  dropa(l, n)
}

val lista = 1 :: 2 :: 3 :: 5 :: 4 :: 8 :: 6 :: Nil

drop(lista, 3)            //> res0: List[Int] = List(5, 4, 8, 6)
```

Zadatak 6.4

- Napisati implementaciju funkcije `splitAt(List[T], n)`, čiji rezultat je par lista. Prvi element para je lista formirana izostavljajući prvih n elemenata ulazne liste. Drugi element para je lista koja se sastoji od preostalih elemenata ulazne liste.

Zadatak 6.4

```
def splitAtEasy[T](l : List[T], n : Int) : (List[T], List[T]) =
{

  (take(l, n), drop(l, n))

}

val lista = 1 :: 2 :: 3 :: 5 :: 4 :: 8 :: 6 :: Nil

splitAtEasy(lista, 3)
//> res2: (List[Int], List[Int]) = (List(1, 2, 3),List(5, 4, 8, 6))
```

Zadatak 6.4

```
def splitAt[T](l : List[T], n : Int) : (List[T], List[T]) =
{
  var a = l;

  def takefront(l : List[T], n : Int) : List[T] =
    if( n > 0 ) l.head :: takefront(l.tail, n-1)
    else {
      a = l
      Nil
    }

  if( n > l.length ) (l, Nil)
  else (takefront(l,n), a)
}

val lista = 1 :: 2 :: 3 :: 5 :: 4 :: 8 :: 6 :: Nil

splitAt(lista, 3)
//> res3: (List[Int], List[Int]) = (List(1, 2, 3),List(5, 4, 8, 6))
```

Zadatak 6.4

Implementacija u `scala.collection.immutable`

```
override def splitAt(n: Int): (List[A], List[A]) = {
  val b = new ListBuffer[A]
  var i = 0
  var these = this
  while (!these.isEmpty && i < n) {
    i += 1
    b += these.head
    these = these.tail
  }
  (b.toList, these)
}
```

Zadatak 6.5

- Implementirati metodu apply() koja vraća n-ti element generičke liste.

```
def apply[T](l : List[T], n : Int) : T =
  drop(l, n).head
//> apply: [T](l: List[T], n: Int)T


// Implementacija iz crte LinearSeqOptimized
def apply(n: Int): A = {
  val rest = drop(n)

  if (n < 0 || rest.isEmpty)
    throw new IndexOutOfBoundsException("" + n)

  rest.head
}
```

Zadatak 6.6

- Napisati funkciju za sortiranje generičke liste primenom tehnike uparivanja obrazaca.

Zadatak 6.6
Rešenje 1

```
def msort[T](cmp : (T,T) => Boolean)(l : List[T]) : List[T] =  {
  val n = l.length / 2
  if( n == 0 ) l
  else  {
    def merge(l1 : List[T], l2 : List[T]) : List[T] = l1 match  {
      case List() => l2
      case h :: t => l2 match {
            case List() => l1
            case h2 :: t2 => if( cmp(h, h2) ) h :: merge(t, l2)
                       else h2 :: merge(l1, t2)
          }
    }
    merge(msort(cmp)(l take n), msort(cmp)(l drop n))
  }
}

msort((x:Int,y:Int) => x < y) (List(3, 5, 1, 2, 7, 4))
//> res7: List[Int] = List(1, 2, 3, 4, 5, 7)
```

Zadatak 6.6
Rešenje 2

```
import math.Ordering
object MergeSort {
  def msort[T](list: List[T])(implicit ord: Ordering[T]): List[T] = {
    val mid = list.length / 2
    if (mid == 0) list
    else {
      def merge(xs: List[T], ys: List[T]): List[T] = (xs, ys) match {
        case (Nil, ys) => ys
        case (xs, Nil) => xs
        case (x :: xs1, y :: ys1) =>
          if (ord.lt(x, y)) x :: merge(xs1, ys)
          else y :: merge(xs, ys1)
      }
      val (first, second) = list.splitAt(mid)
      merge(msort(first), msort(second))
    }
  }
  println(msort(List(2, 1, 5, 2, 76, 7, 2)))
  //> List(1, 2, 2, 2, 5, 7, 76)
}
```

Zadatak 6.7

- Napisati funkciju `map` koja od zadate generičke liste formira novu listu primenom funkcije preslikavanja na njene elemente

```
def map[S, D](f : S => D)(l : List[S]) =
{
  for(e <- l)
    yield f(e)
}

map( (x : Int) => x + 1 )(List(1, 2, 3))
//> res8: List[Int] = List(2, 3, 4)
```

Zadatak 6.7

Implementacija u `scala.collection.immutable`

```
final override def map[B, That](f: A => B)
      (implicit bf: CanBuildFrom[List[A], B, That]): That = {
  if (bf eq List.ReusableCBF) {
    if (this eq Nil) Nil.asInstanceOf[That] else {
      val h = new ::[B](f(head), Nil)
      var t: ::[B] = h
      var rest = tail
      while (rest ne Nil) {
        val nx = new ::(f(rest.head), Nil)
        t.tl = nx
        t = nx
        rest = rest.tail
      }
      h.asInstanceOf[That]
    }
  }
  else super.map(f)
}
```

Zadatak 6.8

- Napisati funkciju koja od generičke liste proizvoljnih elemenata pravi njenu linearizaciju.

Primer:

```
List( List(List(4,5),List(5,6)), List(List(7,8)), 9)
```

pretvara u:

```
List(4, 5, 5, 6, 7, 8, 9)
```

Zadatak 6.8

- Generička klasa List poseduje metodu `flatten` koja obavlja sličan posao.

Primer:

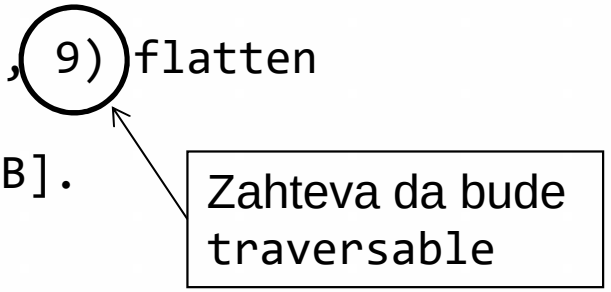
```
List(List(4,5), List(5,6), List(7,8)) flatten
//> res6: List[Int] = List(4, 5, 5, 6, 7, 8)
```

Ali:

```
List(List(List(4,5),List(5,6)), List(List(7,8))) flatten
//> res7: List[List[Int]] = List(List(4, 5), List(5, 6), List(7, 8))
```

Greška:

```
List(List(List(4,5),List(5,6)), List(List(7,8)), 9) flatten
//> No implicit view available from
//| Any => scala.collection.GenTraversableOnce[B].
```



Zadatak 6.8

```
def flatten(l : List[Any]) : List[Any] = l match {
  case Nil => Nil
  case (h : List[_]) :: t => flatten(h) ::: flatten(t)
  case h :: t => h :: flatten(t)
}
```

```
val test2 = List(List(List(4,5),List(5,6)), List(List(7,8)), 9)
//> test2  : List[Any] = List(List(List(4, 5), List(5, 6)),
//| List(List(7, 8)), 9)

flatten(test2)
//> res5: List[Any] = List(4, 5, 5, 6, 7, 8, 9)
```

Zadatak 6.9

- Napisati funkciju reverse, linearne vremenske složenosti, koja obrće redosled elemenata.

```
def reverse[A](l : List[A]) : List[A] = l
match {
  case List() => List()
  case List(x) => List(x)
  case h :: t => concat(reverse(t), List(h))
}

def concat[A](l1 : List[A], l2 :
List[A]) : List[A] = l1 match {
  case List() => l2
  case h :: t => h :: concat(t, l2)
}
```

Kvadratna složenost:
-listu obrće element po element
-konkatenacija ima linearnu složenost

Zadatak 6.9

```
def reverse[T](l : List[T]) : List[T] =
{
  (List[T]() /: l)( (left, right) => right :: left )
}

reverse( List(1, 2, 3) )
//> res13: List[Int] = List(3, 2, 1)
```