Федеральное агентство связи

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и информационных технологий

Отчет по лабораторной работе №6

по дисциплине «Функциональное программирование»

Выполнил: студент группы БВТ1801

Клюшкин Дмитрий Алексеевич

Руководитель:

Мосева Марина Сергеевна

Москва 2020

RecursiveData

a) Реализуйте функцию, определяющую является ли пустым `List[Int]`.

```
def a(list: List[Int]): Boolean = {
  list match {
    case Cons(head, tail) => false
    case Nil() => true
  }
}
```

b) Реализуйте функцию, которая получает head `List[Int]`или возвращает -1 в случае если он пустой.

```
def b(list: List[Int]): Int = {
  list match {
    case Cons(head, tail) => head
    case Nil() => -1
  }
}
```

c) Можно ли изменить `List[A]` так чтобы гарантировать что он не является пустым?

Вместо Nil[A]() использовать Nil[A](head: A)

d) Реализуйте универсальное дерево (Tree) которое хранит значения в виде листьев и состоит из:

* node - левое и правое дерево (Tree)
* leaf - переменная типа A

```
sealed trait Tree[A]
case class TCons[A](leaf: A, node: (Tree[A],Tree[A])) extends Tree[A]
case class TNil[A]() extends Tree[A]
```

RecursiveFunctions

Метод вывода значений List[A]:

```
def vprint[A](as: List[A]): Unit ={
  @tailrec
  def loop(rem: List[A]): Int = rem match {
    case Cons(head, tail) => {print(head+" "); loop(tail)}
    case Nil()=>0
  }
  loop(as)
  println()
}
```

Данные для проверки работы методов:

```
var a:List[Int]=Cons(4,Cons(2,Cons(1,Cons(5,Nil()))))
var b:List[Int]=Cons(8,Cons(9,Cons(7,Cons(6,Nil()))))
var c:List[List[Int]]=Cons(Cons(1,Cons(2,Nil())),
Cons(Cons(3,Cons(4,Nil())),Nil()))
```

a) Напишите функцию которая записывает в обратном порядке список:

```
def reverse[A](list: List[A]): List[A] = {
  @tailrec
  def loop(rem: List[A], as:List[A]): List[A] = rem match {
    case Cons(head, tail) => loop(tail, Cons(head,as))
    case Nil() => as
  }

  loop(list, Nil())
}
```

```
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
Исходный лист: 4 2 1 5
Полученный лист: 5 1 2 4

Process finished with exit code 0
```

b) Напишите функцию, которая применяет функцию к каждому значению списка:

```
val InDo: (Int) => Double = _*1

def map[A, B](list: List[A])(f: A => B): List[B]={
  @tailrec
  def loop(rem: List[A], as:List[B], f: A => B): List[B] = rem match {
    case Cons(head, tail) =>loop(tail, Cons(f(head),as),f)
    case Nil() => reverse(as)
  }
  loop(list, Nil(),f)
}
```

```
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
Исходный лист: 4 2 1 5
Полученный лист: 4.0 2.0 1.0 5.0

Process finished with exit code 0
```

c) Напишите функцию, которая присоединяет один список к другому:

```
def append[A](l: List[A], r: List[A]): List[A]={
  @tailrec
  def loop(ll: List[A], rr:List[A], as:List[A]): List[A] = ll match {
    case Cons(head, tail) => loop(tail,rr, Cons(head,as))
    case Nil() => rr match {
      case Cons(head, tail) => loop(ll,tail, Cons(head,as))
      case Nil() => reverse(as)
    }
  }
  loop(l,r, Nil())
}
```

```
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
Исходный лист:
4 2 1 5
8 9 7 6
Полученный лист: 4 2 1 5 8 9 7 6

Process finished with exit code 0
```

d) Напишите функцию, которая применяет функцию к каждому значению списка: она получает функцию, которая создает новый List[B] для каждого элемента типа A в списке. Поэтому вы создаете List[List[B]].

```
val InDoB: List[Int] => List[Int] = _ match{
    case Cons(head, tail) => Cons(head,tail)
    case Nil() => Nil()
  }

def flatMap[A, B](list: List[A])(f: A => List[B]): List[B]={
  @tailrec
  def loop(rem: List[A], as:List[B], f: A => List[B]): List[B] = {
    rem match {
      case Cons(head, tail) => as match {
        case Cons(ahead, ateil)=> loop(tail,append(as,f(head)),f)
        case Nil()=> loop(tail,f(head),f)
      }
      case Nil() => as
    }
  }
  loop(list, Nil() ,f)
}
```

```
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
Исходный лист:
Cons(Cons(1,Cons(2,Nil())),Cons(Cons(3,Cons(4,Nil())),Nil()))
Полученный лист: Cons(1,Cons(2,Cons(3,Cons(4,Nil()))))

Process finished with exit code 0
```

e) Вопрос: Возможно ли написать функцию с хвостовой рекурсией для `Tree`s? Если нет, почему?

Если дерево имеет отсортированный вид, то можно реализовать поиск с хвостовой рекурсией. В других случаях хвостовая рекурсия не доступна из-за множественного выбора.

Compositions

a) Используйте данные функции. Вы можете реализовать свое решение прямо в тестовой функции.

```
def testCompose[A, B, C, D](f: A => B)
                           (g: B => C)
                           (h: C => D): A => D = h compose g compose f
```

b) Напишите функции с использованием `map` и `flatMap`. Вы можете реализовать свое решение прямо в тестовой функции.

```
def testMapFlatMap[A, B, C, D](f: A => Option[B])
                              (g: B => Option[C])
                              (h: C => D): Option[A] => Option[D] =
                                                  _.flatMap(f).flatMap(g).map(h)
```

c) Напишите функцию используя for. Вы можете реализовать свое решение прямо в тестовой функции.

```
def testForComprehension[A, B, C, D](f: A => Option[B])
                                    (g: B => Option[C])
                                    (h: C => D): Option[A] => Option[D] = ???
```

Не понятно. Хотелось бы иметь наглядный пример.