Лабораторная работа №1
«Введение в функциональное программирование на языке Scala»

Скоробогатов С.Ю.

18 февраля 2017 г.

1 Цель работы

Целью данной работы является ознакомление с программированием на языке Scala на основе чистых функций.

2 Исходные данные

Для выполнения лабораторной работы потребуется REPL-интерпретатор языка Scala, который можно установить через менеджер пакетов ОС Linux или вручную, скачав с сайта по ссылке: `http://www.scala-lang.org/download/`.

Для запуска REPL-интерпретатора нужно запустить команду

```
scala
```

Для выхода из REPL-интерпретатора нужно ввести команду «`:quit`» или использовать комбинацию клавиш Ctrl-D.

В командной строке интерпретатора сразу же после приглашения «`scala>`» можно вводить операторы языка Scala. При этом интерпретатор будет сразу же вычислять и выводить значения, связанные с переменными, или сообщать об ошибках в коде.

Например, после ввода оператора

```
scala> val x = 10*(2+3)
```

интерпретатор напечатает

```
x: Int = 50
```

Операторы, занимающие несколько строк, удобно вводить в специальном режиме, который инициируется командой «`:paste`». При этом для завершения ввода оператора нужно нажать Ctrl-D. Например,

```
scala> :paste
// Entering paste mode (ctrl-D to finish)
val sum: (Int => Boolean) => (List[Int] => Int) =
  p => {
    case Nil               => 0
    case x :: xs if (p(x)) => x + sum(p)(xs)
    case x :: xs           => sum(p)(xs)
  }
// Exiting paste mode, now interpreting.
sum: (Int => Boolean) => (List[Int] => Int) = <function1>
```

3 Задание

Выполнение лабораторной работы заключается в составлении и отладке с помощью REPL-интерпретатора одной из функций, приведённых в таблицах 1 и 2. При составлении функции запрещается использовать возможности Scala, выходящие за рамки функциональной парадигмы. Кроме того, запрещается применять функции стандартной библиотеки языка Scala.

Таблица 1: Варианты функций

1	Функция `fib: (Int, Int => Boolean) => List[Int]`, порождающая последовательность чисел Фибоначчи, не превышающих заданного целого числа и удовлетворяющих некоторому предикату.
2	Функция `powers: (List[Int], Int => Boolean) => List[Int]`, удаляющая из списка те числа, которые не являются заданными предикатом степенями числа 2.
3	Закаренная функция `power: Int => (Int => Int)`, выполняющая быстрое возведение числа в указанную степень (параметр функции – степень).
4	Функция `reverseP: (List[Int], Int => Boolean) => List[Int]`, выполняющая переворачивание списка целых чисел и удаление из него элементов, не удовлетворяющих предикату.
5	Функция `translate: (List[Int], Int => Int) => List[Int]`, порождающая список целых чисел, полученный из исходного списка путём применения к каждому его элементу функции перевода, принимающей `Int` и возвращающей `Int`.
6	Функция `filterIndexes: (List[Int], Int => Boolean) => List[Int]`, удаляющая из списка целых чисел те числа, номера которых в списке не удовлетворяют заданному предикату.
7	Функция `flatten: (List[List[Int]], Int => Boolean) => List[Int]`, выполняющая конкатенацию списков целых чисел, находящихся в списке списков целых чисел и имеющих длину, удовлетворяющую предикату.
8	Функция `split: (List[Int], Int => Boolean) => List[List[Int]]`, выполняющая разбиение последовательности целых чисел на подпоследовательности, разделённые числами, удовлетворяющими предикату.
9	Функция `trim: (List[Int], Int => Boolean) => List[Int]`, выполняющая удаление из списка подряд идущих нулей, количество которых удовлетворяет предикату.
10	Функция `filterBySum: (List[Int], Int => Boolean) => List[Int]`, выполняющая удаление из списка элементов, для которых сумма предыдущих элементов не удовлетворяет предикату.
11	Функция `kadane: List[Int] => (Int, Int)`, выполняющая поиск границ подпоследовательности с максимальной суммой (алгоритм Кадана).
12	Функция `partition: (List[Int], Int => Boolean) => (List[Int], List[Int])`, разделяющая элементы исходного списка на два списка в зависимости от того, удовлетворяют ли они предикату.
13	Функция `peaks: List[Int] => List[Int]`, формирующая список индексов пиков последовательности (пик – такой элемент, что соседние элементы его не превышают).
14	Закаренная функция `digits: Int => (Int => List[Int])`, выполняющая перевод числа в заданную систему счисления (параметр функции – основание системы счисления).

Таблица 2: Варианты функций

15	Функция `zipP: (List[Int], List[Int], (Int, Int) => Boolean) => List[(Int, Int)]`, превращающая два списка целых чисел в список пар целых чисел, в котором первый элемент пары принадлежит первому списку, а второй – второму, и оставляющая в списке только те пары, которые удовлетворяют предикату.
16	Закаренная функция `slices: Int => (List[Int] => List[List[Int]])`, выполняющая разбиение списка целых чисел на фрагменты указанной в качестве параметра функции длины.
17	Закаренная функция `frames: Int => (List[Int] => List[List[Int]])`, формирующая список, состоящий из всех подсписков списка целых чисел указанной в качестве параметра функции длины. Подсписком будем считать список, который можно получить удалением произвольного количества элементов от начала и от конца списка.
18	Закаренная функция `comb: Int => (List[Int] => List[List[Int]])`, формирующая список всех сочетаний элементов списка целых чисел. Размер сочетания передаётся через параметр функции.
19	Функция `coprimes: List[Int] => List[(Int, Int)]`, выполняющая поиск в списке целых чисел пар взаимно простых чисел. Функция должна возвращать список найденных пар, причём в каждой паре первое число должно быть меньше второго.
20	Закаренная функция `sorted: ((Int, Int) => Boolean) => (List[Int] => Boolean)`, принимающая функцию сравнения двух целых чисел и возвращающая функцию, определяющую, является ли список целых чисел отсортированным в соответствии с функцией сравнения.
21	Функция `uniq: List[Int] => (List[Int], Boolean)`, удаляющая дублирующиеся числа из списка отсортированных по возрастанию целых чисел. Функция возвращает пару, первым элементом которой является результирующий список, а вторым – признак успешного выполнения. Выполнение может быть неуспешным, если исходный список не отсортирован по возрастанию.
22	Функция `partition: (List[Int], Int) => (List[Int], List[Int])`, разделяющая список целых чисел на два списка: в первый список помещаются числа, которые меньше указанного числа, а во второй – числа, которые не меньше.
23	Функция `merge: (List[Int], List[Int]) => List[Int]`, выполняющая слияние двух отсортированных по возрастанию списков целых чисел в один отсортированный список.
24	Функция `sortedSeq: List[Int] => List[List[Int]]`, разбивающая список целых чисел на подсписки, в пределах которых числа либо не возрастают, либо не убывают.
25	Функция `sublists: (List[Int], Int) => List[List[Int]]`, разбивающая список целых чисел на подсписки, сумма элементов которых не превышает указанного числа.
26	
27	
28	