**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**

По дисциплине «Функциональное программирование»

На тему:

«Стандартная библиотека Scala»

Выполнил:

студент группы БВТ1801

Задоркин Максим Александрович

Москва 2020

**Цель работы:** изучение стандартной библиотеки scala

**Задание:**

Решение задач из приложенных файлов

**Ход работы:**

**Adts**

import org.apache.logging.log4j.core.pattern.EqualsIgnoreCaseReplacementConverter

import scala.util.{Failure, Success, Try}

object Adts {

*// a) Дан List[Int], верните элемент с индексом n*

def GetNth(list: List[Int], n: Int): Option[Int] = {

List(1,2) **match** {

**case** head :: tail => Some(list(n))

**case** Nil => **null**

**case** l => Option(l(n))

}

}

*// примените функцию из пункта (a) здесь, не изменяйте сигнатуру*

def testGetNth(list: List[Int], n: Int): Option[Int] = GetNth(list, n)

*// b) Напишите функцию, увеличивающую число в два раза.*

def Double(n: Option[Int]): Option[Int] = {

Option(1) **match** {

**case** Some(a) => Option(a \* 2)

**case** None => **null**

}

}

*// примените функцию из пункта (b) здесь, не изменяйте сигнатуру*

def testDouble(n: Option[Int]): Option[Int] = Double(n)

*// c) Напишите функцию, проверяющую является ли число типа Int четным. Если так, верните Right. В противном случае, верните Left("Нечетное число.").*

def IsEven(n: Int): Either[String, Int] = Either.cond(n % 2 == 0, n, "Нечетное число") **match** {

**case** Left(i) => Left(i)

**case** Right(s) => Right(s)

}

*// примените функцию из пункта (c) здесь, не изменяйте сигнатуру*

def testIsEven(n: Int): Either[String, Int] = IsEven(n)

*// d) Напишите функцию, реализующую безопасное деление целых чисел. Верните Right с результатом или Left("Вы не можете делить на ноль.").*

def SafeDivide(a: Int, b: Int): Either[String, Int] = Try(a / b) **match** {

**case** Success(a) => Right(a)

**case** Failure(error) => Left("Вы не можете делить на ноль")

}

*// примените функцию из пункта (d) здесь, не изменяйте сигнатуру*

def testSafeDivide(a: Int, b: Int): Either[String, Int] = SafeDivide(a, b)

*// e) Обработайте исключения функции с побочным эффектом вернув 0.*

def GoodOldJava(impure: String => Int, str: String): Try[Int] =

Try(impure(str)).toEither **match** {

**case** Right(k) => Success(k)

**case** Left(l) => Success(0)

}

*// примените функцию из пункта (e) здесь, не изменяйте сигнатуру*

def testGoodOldJava(impure: String => Int, str: String): Try[Int] = GoodOldJava(impure, str)

}

**Maps**

object Maps {

case class User(name: String, age: Int)

*/\* a) В данной Seq[User] сгруппируйте пользователей по имени (`groupBy`) и вычислите средний возраст: `name -> averageAge`*

*\* Вы можете реализовать ваше решение в теле тестовой функции. Не изменяйте сигнатуру.*

*\*/*

def testGroupUsers(users: Seq[User]): Map[String, Int] = {

var map: Map[String, Int] = Map()

var name = users.groupBy(\_.name)

**for** (e <- name) {

var average = e.\_2.toBuffer.foldLeft[Int](0)((a, next) => a + next.age) /

e.\_2.toBuffer.size

map += (e.\_1 -> average)

}

map

}

*/\* b) Дана `Map[String, User]` состоящая из имен пользователей `User`, сколько имен пользователей, содержащихся в Map, содержат подстроку "Adam"?*

*\* Вы можете реализовать ваше решение в теле тестовой функции. Не изменяйте сигнатуру.*

*\*/*

def testNumberFrodos(map: Map[String, User]): Int = {

var a: Int = 0

**for** (num <- map) **if** (num.\_2.name.indexOf("Adam") != (-1))

a += 1

a

}

*/\* c) Удалите всех пользователей возраст которых менее 35 лет.*

*\* Вы можете реализовать ваше решение в теле тестовой функции. Не изменяйте сигнатуру.*

*\*/*

def testUnderaged(map: Map[String, User]): Map[String, User] = {

map.filter(k => k.\_2.age >= 35)

}

}

**Sequence**

import scala.annotation.tailrec

object Sequence {

*/\* a) Найдите последний элемент Seq.*

*\**

*\*/*

def testLastElement[A](seq: Seq[A]): Option[A] = seq **match** {

**case** head +: tail => testLastElement(tail)

**case** last +: Nil => Option(last)

}

*/\* b) Объедините две Seqs (то есть Seq(1, 2) и Seq(3, 4) образуют Seq((1, 3), (2, 4))) - если Seq длиннее игнорируйте оставшиеся элементы.*

*\**

*\*/*

def testZip[A](a: Seq[A], b: Seq[A]): Seq[(A, A)] = {

def loop[A](a: Seq[A], b: Seq[A], c: Seq[(A, A)]): Seq[(A, A)] = a **match** {

**case** ahead +: atail => b **match** {

**case** blast +: Nil => c :+ (ahead, blast)

**case** bhead +: btail =>loop(atail, btail, c :+ (ahead,bhead))

}

**case** Nil => c

}

loop(a, b, Nil)

}

*/\* c) Проверьте, выполняется ли условие для всех элементов в Seq.*

*\**

*\*/*

def testForAll[A](seq: Seq[A])(cond: A => Boolean): Boolean = {

def loop[A](a: Seq[A])(cond: A => Boolean): Boolean = a **match** {

**case** head +: tail => **if** (cond(head)) loop(tail)(cond)

**else**

**false**

**case** Nil => **true**

}

loop(seq)(cond)

}

*/\* d) Проверьте, является ли Seq палиндромом*

*\**

*\*/*

def testPalindrom[A](seq: Seq[A]): Boolean = {

def loop[A](a: Seq[A], b: Seq[A]): Boolean = {

a **match** {

**case** head +: tail => loop(tail, b = head +: b)

**case** Nil => seq.equals(b)

}

}

loop(seq, Nil)

}

*/\* e) Реализуйте flatMap используя foldLeft.*

*\**

*\*/*

def testFlatMap[A, B](seq: Seq[A])(f: A => Seq[B]): Seq[B] = {

seq.foldLeft[Seq[B]](Nil)((a, next) => f(next).++: (a))

}

}

**Strings**

object Strings {

*/\* a) Преобразуйте все символы типа Char в верхний регистр (не используйте заглавные буквы).*

*\**

*\*/*

def testUppercase(str: String): String = str.toUpperCase()

*/\* b) Вставьте следующие значения в строку:*

*\* Hi my name is <name> and I am <age> years old.*

*\**

*\*/*

def testInterpolations(name: String, age: Int): String =

"Hi my name is %s and I am %d years old.".format(name, age)

*/\* c) Добавьте два числа в следующую строку:*

*\* Hi,*

*\* now follows a quite hard calculation. We try to add:*

*\* a := <value of a>*

*\* b := <value of b>*

*\**

*\* result is <a + b>*

*\**

*\**

*\*/*

def testComputation(a: Int, b: Int): String =

"Hi, \n now follows a quite hard calculation. We try ta add: \n\ta := %d \n\tb := %d\n\n\tresult is %d".format(a,b,a+b)

*/\* d) Если длина строки равна 2, верните всю строку, иначе верните первые два символа строки.*

*\*/*

def testTakeTwo(str: String): String = {

**if** (str.length == 2)

str

**else**

str.take(2)

}

}

**Выводы:**

В процессе лабораторной работы была изучена стандартная библиотека scala.