**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 курсовой работе**

по предмету «Функциональное программирование»

Выполнил: студент группы

Митрохин Ярослав Игоревич

Руководитель:

Мосева Марина Сергеевна

Москва 2020

*Цель работы:* реализовать проект, состоящий из выполненных лабораторных, реализовать в рамках проекта тестирование каждой из лабораторных по отдельности и совместно всего проекта.

*Ход работы:*

*Код тест-классов:*

*TestFunctions:*

package exercise1  
import org.scalatest.FunSuite  
class TestFunctions extends FunSuite{  
 test("testCircle: 2")  
 {  
 val p = Functions.*testCircle*(2)  
 *assert*(p == 4\*Math.*PI*)  
 }  
 test("testRectangleCurried: 2 and 3")  
 {  
 val p = Functions.*testRectangleCurried*(2,3)  
 *assert*(p == 6)  
 }  
 test("testRectangleUc: 2 and 3")  
 {  
 val p = Functions.*testRectangleUc*(2,3)  
 *assert*(p == 6)  
 }  
}

*TestHiOrder:*

package exercise1  
import org.scalatest.FunSuite  
class TestHiOrder extends FunSuite {  
 test("testNTimes: + 1,2,3")  
 {  
 val plus: (Int, Int) => Int = \_+\_  
 val p = HiOrder.*testNTimes*(plus,1,2,3)  
 *assert*(p == 9)  
 }  
 test("testAnonymousNTimes: 3,2,1")  
 {  
 val p = HiOrder.*testAnonymousNTimes*(3,2,1)  
 *assert*(p == 3)  
 }  
}

*TestPatterns:*

package exercise1  
import org.scalatest.FunSuite  
class TestPatterns extends FunSuite {  
 test("testIntToString: 1")  
 {  
 val p = Patterns.*testToString*(1)  
 *assert*(p == "it is one")  
 }  
 test("testIntToString: 2")  
 {  
 val p = Patterns.*testToString*(2)  
 *assert*(p == "it is two")  
 }  
 test("testIntToString: 3")  
 {  
 val p = Patterns.*testToString*(3)  
 *assert*(p == "it is three")  
 }  
 test("testIntToString: 4")  
 {  
 val p = Patterns.*testToString*(4)  
 *assert*(p == "what's that")  
 }  
 test("testIsMaxAndMoritz: max")  
 {  
 val p = Patterns.*testIsMaxAndMoritz*("max")  
 *assert*(p==true)  
 }  
 test("testIsMaxAndMoritz: Max")  
 {  
 val p = Patterns.*testIsMaxAndMoritz*("Max")  
 *assert*(p==true)  
 }  
 test("testIsMaxAndMoritz: moritz")  
 {  
 val p = Patterns.*testIsMaxAndMoritz*("moritz")  
 *assert*(p==true)  
 }  
 test("testIsMaxAndMoritz: morit"){  
 val p = Patterns.*testIsMaxAndMoritz*("morit")  
 *assert*(p == false)  
 }  
 test("testIsEven: 1"){  
 val p = Patterns. *testIsEven*(4)  
 *assert*(p == true)  
 }  
 test("testIsEven: 3"){  
 val p = Patterns. *testIsEven*(3)  
 *assert*(p == false)  
 }  
 test("testWinsA: Rock, Rock"){  
 val p = Patterns.*testWinsA*(Patterns.Rock, Patterns.Rock)  
 *assert*(p == Patterns.Draw)  
 }  
 test("testWinsA: Paper, Rock"){  
 val p = Patterns.*testWinsA*(Patterns.Paper, Patterns.Rock)  
 *assert*(p == Patterns.Win)  
 }  
 test("testWinsA: Scissor, Rock"){  
 val p = Patterns.*testWinsA*(Patterns.Scissor, Patterns.Rock)  
 *assert*(p == Patterns.Lose)  
 }  
 test("testExtractMammalWeight:KIT and Plants"){  
 val p = Patterns.*testExtractMammalWeight*(Patterns.*Mammal*("KIT",Patterns.Plants,2))  
 *assert*(p==2)  
 }  
 test("testUpdateFood: Fish and Plants"){  
 val p = Patterns.*testUpdateFood*(Patterns.*Fish*("Fish",Patterns.Meat))  
 *assert*(p == Patterns.*Fish*("Fish",Patterns.Plants))}}

*TestComposition:*

package exercise2  
import org.scalatest.FunSuite  
class TestCompositions extends FunSuite{  
 val *sw*:(Double)=>Int = \_.toInt  
 val *sw1*:(Int)=>Byte = \_.toByte  
 val *sw2*:(Byte)=>String = \_.toString  
 test("testCompose: 1134 to String ")  
 {  
 val p = Compositions.*testCompose*(*sw*)(*sw1*)(*sw2*)  
 *assert*(p(1134) == "110")  
 }  
 test("testMapFlatMap: double -> int -> byte -> string"){  
 val A1: (Double) => Int = \_.toInt  
 val B1: (Int) => Byte = \_.toByte  
 val C1: (Byte) => String = \_.toString  
  
 val A: (Double) => Option[Int] = *Some*(\_).map(A1)  
 val B: (Int) => Option[Byte] = *Some*(\_).map(B1)  
 val C: (Byte) => String = \_.toString  
 val p = Compositions.*testMapFlatMap*(A)(B)(C)  
 *assert*(p(*Some*(1134)).toString == (*Some*(110).toString))  
 }  
  
  
 //test testForComprehension  
 test("testForComprehension: double -> int -> byte -> string"){  
 val A1: (Double) => Int = \_.toInt  
 val B1: (Int) => Byte = \_.toByte  
  
 val A: (Double) => Option[Int] = *Some*(\_).map(A1)  
 val B: (Int) => Option[Byte] = *Some*(\_).map(B1)  
 val C: (Byte) => String = \_.toString  
 val p = Compositions.*testMapFlatMap*(A)(B)(C)  
 *assert*(p(*Some*(1134)).toString == (*Some*(110).toString))  
 }  
}

*TestRecursiveData:*

package exercise2  
import org.scalatest.FunSuite  
class TestRecursiveData extends FunSuite{  
 val *a* = *Cons*(1, *Cons*(2, *Cons*(3, *Cons*(4, *Nil*()))))  
 test("testListIntEmpty: Nil() ")  
 {  
 val p = RecursiveData.*testListIntEmpty*(*Nil*())  
 *assert*(p == true)  
 }  
 test("testListIntHead: 1 2 3 4 ")  
 {  
 val p = RecursiveData.*testListIntEmpty*(*Nil*())  
 *assert*(p == true)  
 }  
}

*TestRecursiveFunc:*

package exercise2  
import org.scalatest.FunSuite  
class TestRecursiveFunc extends FunSuite{  
 val *a* = *Cons*(1, *Cons*(2, *Cons*(3, *Cons*(4, *Nil*()))))  
 val *b* = *Cons*(4, *Cons*(3, *Cons*(2, *Cons*(1, *Nil*()))))  
 val *c* = *Cons*(*Cons*(4, *Cons*(2, *Nil*())), *Cons*(*Cons*(5, *Cons*(4, *Nil*())), *Nil*()))  
 val *de* = (l: List[Int]) => l  
 val *sw* = (x: Int) => x.toDouble  
 test("testReverse: 1 2 3 4 ")  
 {  
 val p = RecursiveFunctions.*testReverse*(*a*)  
 *assert*(p == *Cons*(4, *Cons*(3, *Cons*(2, *Cons*(1, *Nil*())))))  
 }  
 test("testMap: 1 2 3 4 ")  
 {  
 val p = RecursiveFunctions.*testMap*(*a*,*sw*)  
 *assert*(p == *Cons*(1.0, *Cons*(2.0, *Cons*(3.0, *Cons*(4.0, *Nil*())))))  
 }  
 test("testAppend: 1 2 3 4 and 4 3 2 1 ")  
 {  
 val p = RecursiveFunctions.*testAppend*(*a*,*b*)  
 *assert*(p == *Cons*(1, *Cons*(2, *Cons*(3, *Cons*(4, *Cons*(4, *Cons*(3,*Cons*(2,*Cons*(1, *Nil*())))))))))  
 }  
 test("testFlatMap: [4,2] and [5,4] ")  
 {  
 val p = RecursiveFunctions.*testFlatMap*(*c*,*de*)  
 *assert*(p == *Cons*(2, *Cons*(4, *Cons*(4, *Cons*(5, *Nil*())))))  
 }  
}

*TestAdts:*

package exercise3  
import org.scalatest.FunSuite  
  
import scala.util.Success  
class TestAdts extends FunSuite{  
  
 test("testGetNth ")  
 {  
 val p = Adts.*testGetNth*(*List*(1,9,3,2,1,4),2)  
 *assert*(p == Some(3))  
 }  
 test("testDouble: 5")  
 {  
 val p = Adts.*testDouble*(Some(5))  
 *assert*(p == Some(10))  
 }  
 test("testIsEven: 4")  
 {  
 val p = Adts.*testIsEven*(4)  
 *assert*(p == *Right*(4))  
 }  
 test("testIsEven: 7")  
 {  
 val p = Adts.*testIsEven*(7)  
 *assert*(p == *Left*("Нечетное число"))  
 }  
 test("testSafeDivide: 5 на 0")  
 {  
 val p = Adts.*testSafeDivide*(5,0)  
 *assert*(p == *Left*("Вы не можете делить на ноль."))  
 }  
 test("testGoodOldJava: 2153")  
 {  
 var parse = (x: String) => Integer.*parseInt*(x)  
 val p = Adts.*testGoodOldJava*(parse,"2153")  
 *assert*(p == Success(2153) )  
 }  
}

*TestMaps:*

package exercise3  
import exercise3.Maps.User  
import org.scalatest.FunSuite  
class TestMaps extends FunSuite{  
 test("testGroupUsers ")  
 {  
 val p = Maps.*testGroupUsers*(*Seq*(*User*("YarikAdam",20),*User*("Adam",26),*User*("Adam",65)))  
 *assert*(p == *Map*(("YarikAdam",20),("Adam",45)))  
 }  
 test("testNumberFrodos")  
 {  
 val p = Maps.*testNumberFrodos*(*Map*("1"->*User*("Adam",34),"2"->*User*("Yir",35),"3"->*User*("Mir",36)))  
 *assert*(p == 1)  
 }  
 test("testUnderaged")  
 {  
 val p = Maps.*testUnderaged*(*Map*("1"->*User*("Adam",34),"2"->*User*("Yir",36),"3"->*User*("Mir",36)))  
 *assert*(p == *Map*("2"->*User*("Yir",36),"3"->*User*("Mir",36)))  
 }  
}

*TestSequence:*

package exercise3  
import org.scalatest.FunSuite  
class TestSequence extends FunSuite{  
 test("testLastElement 5 2 6 1 ")  
 {  
 val p = Sequence.*testLastElement*(*Seq*(5,2,6,1))  
 *assert*(p == Some(1))  
 }  
 test("testZip: 1 2 3 and 3 2 1")  
 {  
 val p = Sequence.*testZip*(*Seq*(1,2,3),*Seq*(3,2,1))  
 *assert*(p == *Seq*((1,3),(2,2),(3,1)))  
 }  
 test("testForAll: 2,4,6")  
 {  
 val cond = (i: Int) => i>2  
 val p = Sequence.*testForAll*(*Seq*(2,4,6))(cond)  
 *assert*(p == false)  
 }  
 test("testForAll: 2,4,6,8")  
 {  
 val cond = (i: Int) => i>1  
 val p = Sequence.*testForAll*(*Seq*(2,4,6,8))(cond)  
 *assert*(p == true)  
 }  
 test("testPalindrom: 2,4,4,2")  
 {  
 val p = Sequence.*testPalindrom*(*Seq*(2,4,4,2))  
 *assert*(p == true)  
 }  
 test("testFlatMap: ")  
 {  
 val A = (i: Seq[Int])=> i  
 val p = Sequence.*testFlatMap*(*Seq*(*Seq*(2,3),*Seq*(4,5)))(A)  
 *assert*(p == *Seq*(4,5,2,3))  
 }  
}

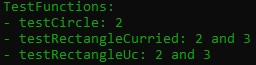
*TestString:*

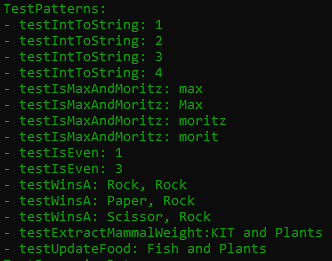
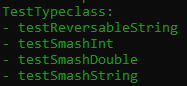
package exercise3  
import org.scalatest.FunSuite  
class TestString extends FunSuite{  
 test("testUppercase"){  
 val p = Strings.*testUppercase*("wfie")  
 *assert*(p == "WFIE")  
 }  
 test("testInterpolations"){  
 val p = Strings.*testInterpolations*("Vova",12)  
 *assert*(p == "Hi my name is Vova and I am 12 years old.")  
 }  
 test("testComputation"){  
 val p = Strings.*testComputation*(4,5)  
 *assert*(p == "Hi,\nnow follows a quite hard calculation. We try to add:\n\ta := 4\n\tb := 5\n\n\tresult is 9")  
 }  
 test("testTakeTwo: 2"){  
 val p = Strings.*testTaleTwo*("KO")  
 *assert*(p == "KO")  
 }  
 test("testTakeTwo: 5"){  
 val p = Strings.*testTaleTwo*("Hello")  
 *assert*(p == "He")  
 }  
}

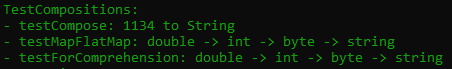
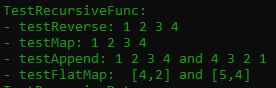
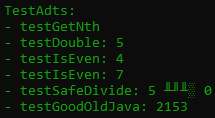
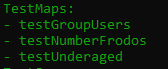
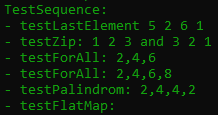
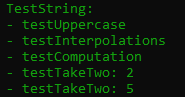
*TestTypeclasses:*

package Lab\_ex4  
import org.scalatest.FunSuite  
class TestTypeclass extends FunSuite {  
  
 test("testReversableString"){  
 val p = Typeclasses.*textReversableString*("kvak")  
 *assert*(p == "kavk")  
 }  
 test("testSmashInt"){  
 val p = Typeclasses.*testSmashInt*(6,3)  
 *assert*(p == 9)  
 }  
 test("testSmashDouble"){  
 val p = Typeclasses.*testSmashDouble*(3.0,3.0)  
 *assert*(p == 9.0)  
 }  
 test("testSmashString"){  
 val p = Typeclasses.*testSmashString*("He", "llo")  
 *assert*(p == "Hello")  
 }  
}

*Результаты:*

** **

** **

** **    ** 

Ошибка при неверном тесте:

**

*Выводы:* реализовал проект, состоящий из выполненных лабораторных, реализовал в рамках проекта тестирование каждой из лабораторных по отдельности и совместно всего проекта.