**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 дисциплине «Функциональное программирование»

Выполнил:

студент группы БВТ1801

Алмамма Башар

Руководитель:

Мосева Марина Сергеевна

Москва 2020

Содержание

[Цель работы: 3](#_Toc42960231)

[Задача: 3](#_Toc42960232)

[Пример работы тестирующей программы: 3](#_Toc42960233)

[Покрытие: 4](#_Toc42960234)

[Код тестирующей программы: 4](#_Toc42960235)

[Для первой лабораторной работы: 4](#_Toc42960236)

[Для второй лабораторной работы: 6](#_Toc42960237)

[Для третьей лабораторной работы: 8](#_Toc42960238)

[Для четвертой лабораторной работы: 11](#_Toc42960239)

[Вывод: 11](#_Toc42960240)

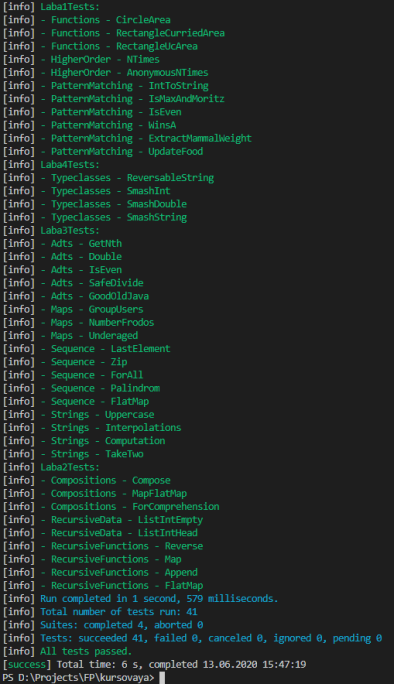
# Цель работы:

Изучить ScalaTest. Узнать, как создать проект Scala, использующий ScalaTest (одна из основных библиотек тестирования для проектов Scala). Скомпилировать, протестировать и запустить проект с SBT.

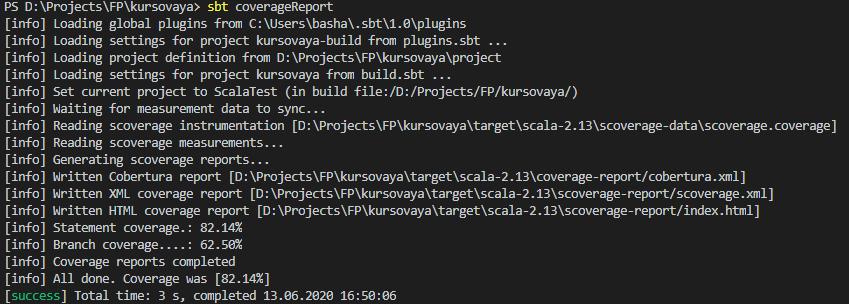
# Задача:

На основе ранее проделанных лабораторных работ создать проект и провести тесты для всех выполненных заданий. Так же проверить покрытие тестами кода.

# Пример работы тестирующей программы:



# Покрытие:



Результат Coverage Test = 82.14%

# Код тестирующей программы:

## Для первой лабораторной работы:

package laba1

import org.scalatest.FunSuite

class Laba1Tests extends FunSuite{

    test("Functions - CircleArea") {

        val circleArea = Functions.testCircle(3.0)

        assert(circleArea == 28.274333882308138)

    }

    test("Functions - RectangleCurriedArea") {

        val rectangleAreaCurried = Functions.testRectangleCurried(3.0, 4.0)

        assert(rectangleAreaCurried == 12.0)

    }

    test("Functions - RectangleUcArea") {

        val rectangleArea = Functions.testRectangleUc(3.0, 4.0)

        assert(rectangleArea == 12.0)

    }

    test("HigherOrder - NTimes") {

        val nTimes1 = HigherOrder.testNTimes(HigherOrder.plus, 3 ,2 ,5)

        assert(nTimes1 == 25)

        val nTimes2 = HigherOrder.testNTimes(HigherOrder.multiply, 3 ,2 ,5)

        assert(nTimes2 == 30)

    }

    test("HigherOrder - AnonymousNTimes") {

        val anonymousNTimes = HigherOrder.testAnonymousNTimes(3 ,2 ,5)

        assert(anonymousNTimes == 15)

    }

    test("PatternMatching - IntToString") {

        val intToString = PatternMatching.testIntToString(2)

        assert(intToString == "it is two")

    }

    test("PatternMatching - IsMaxAndMoritz") {

        assert(PatternMatching.testIsMaxAndMoritz("Max") == true)

        assert(PatternMatching.testIsMaxAndMoritz("max") == true)

        assert(PatternMatching.testIsMaxAndMoritz("Moritz") == true)

        assert(PatternMatching.testIsMaxAndMoritz("moritz") == true)

        assert(PatternMatching.testIsMaxAndMoritz("test") == false)

    }

    test("PatternMatching - IsEven") {

        val isEven = PatternMatching.testIsEven(2)

        assert(isEven == true)

    }

    test("PatternMatching - WinsA") {

        val winsA1 = PatternMatching.testWinsA(PatternMatching.Paper,PatternMatching.Scissor)

        assert(winsA1 == PatternMatching.Lose)

        val winsA2 = PatternMatching.testWinsA(PatternMatching.Rock,PatternMatching.Scissor)

        assert(winsA2 == PatternMatching.Win)

        val winsA3 = PatternMatching.testWinsA(PatternMatching.Scissor,PatternMatching.Scissor)

        assert(winsA3 == PatternMatching.Draw)

    }

    test("PatternMatching - ExtractMammalWeight") {

        val extractMammalWeight1 = PatternMatching.testExtractMammalWeight(PatternMatching.Mammal("mammal", PatternMatching.Meat, 100))

        assert(extractMammalWeight1 == 100)

        val extractMammalWeight2 = PatternMatching.testExtractMammalWeight(PatternMatching.Fish("fish", PatternMatching.Meat))

        assert(extractMammalWeight2 == -1)

    }

    test("PatternMatching - UpdateFood") {

        val updateFood1 = PatternMatching.testUpdateFood(PatternMatching.Fish("fish", PatternMatching.Meat))

        assert(updateFood1.food == PatternMatching.Plants)

        val updateFood2 = PatternMatching.testUpdateFood(PatternMatching.Mammal("mammal", PatternMatching.Meat, 100))

        assert(updateFood2.food == PatternMatching.Meat)

    }

}

## Для второй лабораторной работы:

package laba2

import org.scalatest.FunSuite

class Laba2Tests extends FunSuite{

    test("Compositions - Compose") {

        val f = (value: Int) => value \* 13

        val g = (value: Int) => value + 13

        val h = (value: Int) => value - 13

        val res = (value: Int) => value \* 13 + 13 - 13

        val compose = Compositions.testCompose(f)(g)(h)

        assert(compose(10) == res(10))

    }

    test("Compositions - MapFlatMap") {

        val f = (value:Int) => Some[Int](value)

        val g = (value:Int) => Some[Int](value)

        val h = (value: Int) => value - 13

        val res = Some[Int](-11)

        val mapFlatMap = Compositions.testMapFlatMap(f)(g)(h)

        assert(mapFlatMap(Some[Int](2)) == res)

    }

    test("Compositions - ForComprehension") {

        val f = (value:Int) => Some[Int](value)

        val g = (value:Int) => Some[Int](value)

        val h = (value: Int) => value - 13

        val res = Some[Int](-11)

        val forComprehension = Compositions.testForComprehension(f)(g)(h)

        assert(forComprehension(Some[Int](2)) == res)

    }

    test("RecursiveData - ListIntEmpty") {

        val list = Nil[Int]()

        val listIntEmpty = RecursiveData.testListIntEmpty(list)

        assert(listIntEmpty == true)

    }

    test("RecursiveData - ListIntHead") {

        val list = Cons[Int](1 , Nil[Int]())

        val listIntHead = RecursiveData.testListIntHead(list)

        assert(listIntHead == 1)

    }

    test("RecursiveFunctions - Reverse") {

        val list = Cons[Int](1 , Cons[Int](2 , Cons[Int](3 , Nil[Int]())))

        val list2 = Cons[Int](3 , Cons[Int](2 , Cons[Int](1 , Nil[Int]())))

        val reverse = RecursiveFunctions.testReverse(list)

        assert(reverse == list2)

    }

    test("RecursiveFunctions - Map") {

        val list = Cons[Int](1 , Cons[Int](2 , Cons[Int](3 , Nil[Int]())))

        val list2 = Cons[Int](2 , Cons[Int](4 , Cons[Int](6 , Nil[Int]())))

        val func = (value: Int) => value \* 2

        val map = RecursiveFunctions.testMap(list, func)

        assert(map == list2)

    }

    test("RecursiveFunctions - Append") {

        val list = Cons[Int](1 , Nil[Int]())

        val list2 = Cons[Int](2 , Cons[Int](3 , Nil[Int]()))

        val list3 = Cons[Int](1 , Cons[Int](2 , Cons[Int](3 , Nil[Int]())))

        val append = RecursiveFunctions.testAppend(list, list2)

        assert(append == list3)

    }

    test("RecursiveFunctions - FlatMap") {

        val func: List[Int] => List[Int] = \_ match{

            case Cons(head, tail) => Cons(head,tail)

            case Nil() => Nil()

        }

        val list = Cons[Cons[Int]](Cons[Int](1 , Nil[Int]()), Cons[Cons[Int]](Cons[Int](2 , Nil[Int]()), Nil[Cons[Int]]()))

        val list1 = Cons[Int](1 , Cons[Int](2 , Nil[Int]()))

        val flatMap = RecursiveFunctions.testFlatMap(list, func)

        assert(flatMap == list1)

    }

}

## Для третьей лабораторной работы:

package laba3

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

import org.scalatest.FunSuite

class Laba3Tests extends FunSuite{

    test("Adts - GetNth") {

        val list = List(1,5,8,9,10)

        val n = 3

        val res = Some(9)

        val getNth = Adts.testGetNth(list,n)

        assert(getNth == res)

    }

    test("Adts - Double") {

        val list = Option(2)

        val res = Option(4)

        val double = Adts.testDouble(list)

        assert(double == res)

    }

    test("Adts - IsEven") {

        val n1 = 13

        val res1 = Left("Нечетное число")

        val isEven1 = Adts.testIsEven(n1)

        assert(isEven1 == res1)

        val n2 = 12

        val res2 = Right(12)

        val isEven2 = Adts.testIsEven(n2)

        assert(isEven2 == res2)

    }

    test("Adts - SafeDivide") {

        val n1 = 20

        val n2 = 0

        val n3 = 1

        val res1 = Left("Вы не можете делить на ноль")

        val safeDivide1 = Adts.testSafeDivide(n1,n2)

        assert(safeDivide1 == res1)

        val res2 = Right(20)

        val safeDivide2 = Adts.testSafeDivide(n1,n3)

        assert(safeDivide2 == res2)

    }

    test("Adts - GoodOldJava") {

        val string = "test"

        val func = (s:String) => s.size

        val res = Success(4)

        val goodOldJava = Adts.testGoodOldJava(func,string)

        assert(goodOldJava == res)

    }

    test("Maps - GroupUsers") {

        val user1 = new Maps.User("Bashar", 15)

        val user2 = new Maps.User("Maxim", 20)

        val user3 = new Maps.User("Dima", 20)

        val seq = Seq(user1,user2,user3)

        val res = Map(("Bashar", 18),("Maxim", 18),("Dima", 18))

        val groupUsers = Maps.testGroupUsers(seq)

        assert(groupUsers == res)

    }

    test("Maps - NumberFrodos") {

        val user1 = new Maps.User("AdamBashar", 20)

        val user2 = new Maps.User("Maxim", 20)

        val user3 = new Maps.User("AdamDima", 20)

        val map = Map(("AdamBash", user1),("AdamMaxim", user2),("Dima", user3))

        val res = 2

        val numberFrodos = Maps.testNumberFrodos(map)

        assert(numberFrodos == res)

    }

    test("Maps - Underaged") {

        val user1 = new Maps.User("Bashar", 40)

        val user2 = new Maps.User("Maxim", 30)

        val user3 = new Maps.User("Dima", 37)

        val map = Map(("Bash", user1),("Maxim", user2),("Dima", user3))

        val res = Map(("Bash", user1),("Dima", user3))

        val underaged = Maps.testUnderaged(map)

        assert(underaged == res)

    }

    test("Sequence - LastElement") {

        val seq = List(1,2,3)

        val res = Option(3)

        val lastElement = Sequence.testLastElement(seq)

        assert(lastElement == res)

    }

    test("Sequence - Zip") {

        val seq1 = List(1,2)

        val seq2 = List(3,4)

        val res = List((1,3),(2,4))

        val zip = Sequence.testZip(seq1,seq2)

        assert(zip == res)

    }

    test("Sequence - ForAll") {

        val seq = List(2,4,6,8)

        val func = (value: Int) => value % 2 == 0

        val res = true

        val forAll = Sequence.testForAll(seq)(func)

        assert(forAll == res)

    }

    test("Sequence - Palindrom") {

        val seq = List(1,2,2,1)

        val res = true

        val palindrom = Sequence.testPalindrom(seq)

        assert(palindrom == res)

    }

    test("Sequence - FlatMap") {

        val seq = List(1,2,3,4)

        val func = (value: Any) => List(List(value))

        val res = List(List(1),List(2),List(3),List(4))

        val flatMap = Sequence.testFlatMap(seq)(func)

        assert(flatMap == res)

    }

    test("Strings - Uppercase") {

        val string = "aaa"

        val res = "AAA"

        val uppercase = Strings.testUppercase(string)

        assert(uppercase == res)

    }

    test("Strings - Interpolations") {

        val name = "Bashar"

        val res = "Hi my name is Bashar and I am 20 years old."

        val interpolations = Strings.testInterpolations(name, 20)

        assert(interpolations == res)

    }

    test("Strings - Computation") {

        val res = "Hi, \n now follows a quite hard calculation. We try ta add: \n\ta := 5 \n\tb := 10\n\n\tresult is 15"

        val computation = Strings.testComputation(5, 10)

        assert(computation == res)

    }

    test("Strings - TakeTwo") {

        val string = "test"

        val res = "te"

        val takeTwo = Strings.testTakeTwo(string)

        assert(takeTwo == res)

    }

}

## Для четвертой лабораторной работы:

package laba4

import org.scalatest.FunSuite

class Laba4Tests extends FunSuite{

    test("Typeclasses - ReversableString") {

        val string = "abc"

        val res = "cba"

        val reversableString = Typeclasses.testReversableString(string)

        assert(reversableString == res)

    }

    test("Typeclasses - SmashInt") {

        val a = 3

        val b = 4

        val res = 7

        val smashInt = Typeclasses.testSmashInt(a, b)

        assert(smashInt == res)

    }

    test("Typeclasses - SmashDouble") {

        val a = 3

        val b = 4

        val res = 12

        val smashDouble = Typeclasses.testSmashDouble(a, b)

        assert(smashDouble == res)

    }

    test("Typeclasses - SmashString") {

        val a = "ab"

        val b = "cd"

        val res = "abcd"

        val smashString = Typeclasses.testSmashString(a, b)

        assert(smashString == res)

    }

}

# Вывод:

Изучена библиотека ScalaTest. В процессе изучения были освоены навыки тестирования языка scala и работа coverage тестов.