Prof. Dr. Th. Letschert
Nebenläufige und verteilte Programme

TECHNISCHE HOCHSCHULE MITTELHESSEN
MNI Mathematik, Naturwissenschaften und Informatik

Scala Einstieg- und Aufwärmungübungen

Dieses Aufgabenblatt dient dem Einstieg in Scala, bzw. dem Aufwärmen der Kenntnisse. Es enthält einige Fingerübungen mit Lösungshinweisen. Es wird in der Veranstaltung nicht besprochen. Alle Teilnehmer, speziell solche ohne oder mit nur geringen Scala-Kenntnissen, sollten dieses Aufgabenblatt vor Beginn des Kurses unbedingt bearbeiten.

Aufgabe 1

Installieren Sie Java 8 und Scala 2.11 oder 2.12 auf Ihrem Rechner.

Aufgabe 2

Übersetzen Sie von Java nach Scala:

```
package rechnen;

public class Rechnen {

   public static int max(int x, int y) {
      return x>y?x:y;
   }

   public static int min(int x, int y) {
      return x<y?x:y;
   }

   public static int ggt(int x, int y) {
      if (x <= 0 || y <= 0) throw new IllegalArgumentException();
      while (x != y) {
         if (y > x) {
            int t = x;
            x = y; y = t;
         }
         x = x-y;
      }
      return x;
   }

   public static int kgv(int x, int y) {
      if (x <= 0 || y <= 0) throw new IllegalArgumentException();
      return x*y/ggt(x,y);
   }

   public static void p(int a, int b) {
      for (int i= a; i<b; i++) {
        System.out.println("i = " + i);
      }
   }

}
```

Lösungshinweis

```
package aufgabe_2.rechnen

object Rechnen {

  def max(x: Int, y: Int): Int = if (x>y) x else y

  def ggt(x: Int, y: Int) : Int = {
    if (x <= 0 || y <= 0) throw new IllegalArgumentException()
    var x_ = x
    var y_ = y
    while (x_ != y_) {
      if (y > x) {
        val t = x;
        x_ = y_; y_ = t
      }
      x_ = x_ -y_
    }
    x
  }

  // besser
  def ggT(x: Int, y: Int) : Int = {
    val max = if (x>y) x else y
    val min = if (x>y) y else x
    if (x==y) x else ggt(max-min, min)
  }

  // oder auch:
  def gGT(x: Int, y: Int) : Int = {
    val (max, min) = (if (x>y) x else y, if (x>y) y else x)
    if (x==y) x else ggt(max-min, min)
  }

  // oder auch:
  def g_GT(x: Int, y: Int) : Int = {
    if (x==y) return x
    val (max, min) = (if (x>y) x else y, if (x>y) y else x)
    ggt(max-min, min)
  }

  def kgv(x: Int, y: Int): Int = {
       if (x <= 0 || y <= 0) throw new IllegalArgumentException();
       x*y/ggt(x,y);
    }

  def p(a: Int, b: Int) = {
    for (i <- a until b) {
      println("i = " + i);
    }
  }

}
```

Aufgabe 3

Eine Folge von natürlichen Zahlen sei definiert als:

$a_0 = 0$
$a_i = 2 * a_{i-1} + 1$

Die Funktion f sei definiert als

$f(x) = a_x$

In Java würde man f als statische Methode einer statischen Klasse `Folge` im Paket `folge` implementieren. Implementieren Sie f in äquivalenter Art in Scala.

Implementieren Sie in Scala als Erweiterung von `Folge` eine iterative Variante `fIter` von `f` an. In `fIter` soll f iterativ, d.h. durch eine Schleife berechnet werden.

Schreiben Sie in Scala als Erweiterung von `Folge` eine Funktion `sumF`, die für eine natürliche Zahl x den Wert

$$\sum_{i=0}^{x} f(i)$$

berechnet.

Lösungshinweis

```
object Folge {

  def f(x: Int) : Int = if (x==0) 0 else f(x-1)*2 + 1

  def fIter(x: Int) : Int = {
    var a = 0
    var i = 0
    while (i < x) {
      a = a*2+1
      i = i+1
    }
    a
  }

  def sum(x: Int) : Int =
    (0 to x).
       map(f(_)).
          foldLeft(0)(_+_)

}
```

Aufgabe 4

Ein Palindrom ist eine Zeichenkette die von rechts und links gelesen gleich ist. Beispielsweise ist *Anna* ein Palindrom. (Groß-und Kleinschreibung ignorieren wir.)

Der Begriff Palindrom kann leicht rekursiv definiert werden: Ein Wort ist ein Palindom, wenn keinen oder nur einen Buchstaben enthält, oder wenn der erste und der letzte Buchstabe gleich sind und die Zeichenkette dazwischen ein Palindrom ist.

Entwickeln Sie aus dieser Idee eine rekursive Funktion die testet ob eine Zeichenkette ein Palindrom ist.

Lösungshinweis

```
package aufgabe_4

object Palindrom {
```

```
  def pal1(s: String) : Boolean =
    s.length() == 0 || s.length() == 1 ||
       s.head == s.last &&
          pal1(s.subSequence(1, s.length-1).asInstanceOf[String])

  // oder:
  def pal2(s: Seq[Char]) : Boolean = s match {
    case Seq() => true
    case Seq(x) => true
    case _ => pal2(s.slice(1, s.length-1)) && s.head == s.last
  }

  // oder:
  def pal3(s: Seq[Char]) : Boolean = s match {
    case Seq() => true
    case Seq(x) => true
    case head +: tail =>
      val r = tail.reverse;
      head == r.head && pal3(tail.reverse.tail)
  }

  // oder:
  def pal4(l: Seq[Char]) : Boolean = l.toList match {
    case Nil => true
    case head :: Nil => true
    case head :: tail => head == tail.last && pal4(tail.drop(l.length-1))
  }

}
```

Aufgabe 5

Schreiben Sie eine Funktion, die ein Array mit den ersten n Fibonacci–Zahlen liefert.

Lösungshinweis

```
object FibTab extends App {

  def fibs(n:Int) : Array[Int] =
    Array.tabulate(n)(
       i => new Function1[Int,Int]{ // anonyme Funktion
             def apply(x:Int):Int = if(x<2) 1 else apply(x-1) + apply(x-1)
          }.apply(i))


  println(fibs(5).toList)

}
```

Aufgabe 6

Betrachten Sie die folgenden Codesequenzen und erläutern Sie, welche Ausgabe wird eventuell erzeugt, bzw. geben Sie Anwendungsbeispiele der Funktionen an:

```
val a : Pair[Int, Int] = (1, 2)
val b = ("Tim", "White")
val c : Pair[Double, Int] = (1.9, 23)
println(a._1)
println((1, 2)._2)
println(b._2)
```

```
val startElement = 0
def operation(a:Int, b:Int) : Int = a + b
List(1, 2, 3, 4).foldLeft(startElement)(operation)
```

```
def evensUntil(x: Int) =
  for (i <- (1 until x);
      if i % 2 == 0) yield i
```

```
def allPairs(ts: List[Int], us: List[Int]): List[(Int, Int)] =
  for {
    t <- ts
    u <- us
  } yield (t, u)
```

```
def allPairs[T, U](ts: Seq[T], us: Seq[U]): Seq[(T, U)] =
  for {
    t <- ts
    u <- us
  } yield (t, u)
```

```
def printItems[T](items: Seq[T]) {
  items.foreach(println)
}
```

```
def plusX(x: Int): Int => Int =
  (y: Int) => x + y
```

```
def foo(x: Int): Option[Int] =
  if (x < 0) {
    None
  } else {
    Some(x)
  }
```

```
def matchOption(opt: Option[Object]) =
  opt match {
    case Some(x) => println("Got object: " + x)
    case None => println("Null value")
  }
```

```
List(1, 2, 3, 4).foldLeft(0){(a:Int, b:Int) => a + b}
```

```
List(1, 2, 3, 4).foldLeft(0){(a:Int, b:Int) => a * b}
```

```
def myFoldLeft(items: List[Int], accumulator: Int, func: (Int, Int) => Int): Int =
  items match {
    case Nil => accumulator
    case item :: rest =>
      myFoldLeft(rest, func(accumulator, item), func)
  }
```

```
def myFoldLeft(items: List[Int], accumulator: String, func: (Int, String) => String:
    String =
  items match {
    case Nil => accumulator
    case item :: rest =>
      myFoldLeft(rest, func(accumulator, item), func)
  }
```

```
def myFoldLeft[A,B](items: List[A], accumulator: B, func: (B, A) => B): B =
  items match {
    case Nil => accumulator
    case item :: rest =>
      myFoldLeft(rest, func(accumulator, item), func)
  }
```

Aufgabe 7

Übersetzen Sie nach Scala:

```
public class FoldLeftExample {
   public static int summation(List<Integer> list) {
   int sum = 0;
   for (Integer i : list) {
      sum += i.intValue();
   }
   return sum;
   }
}
```

```
import java.util.ArrayList;
import java.util.List;

public class MapExample {
   public static List<Integer> addAmount(List<Integer> list, int amount) {
      List<Integer> retval = new ArrayList<Integer>(list.size());
      for (Integer i: list) {
         retval.add(i+amount);
      }
      return retval;
   }

   public static void main(String[] args) {
      List<Integer> lst = new ArrayList<>();
      for (int i=0; i< 10; i++) {
         lst.add(i);
      }
      lst = addAmount(lst, 1);
      for (int i : lst) {
```

```
         System.out.print(i + " ");
      }
      System.out.println();
   }

}
```

```
import java.util.*;

public class FilterExample {
   public static List<Integer> getGreaterThanAmount(List<Integer> list,
                       int amount) {
   List<Integer> retval = new ArrayList<Integer>(list.size());
   for (Integer i: list) {
      if (i.intValue() > amount) {
      retval.add(i);
      }
   }
   return retval;
   }
}
```

Lösungshinweis

```
object FoldLeftExample {

  def summation(list : List[Int]): Int = {
    var sum = 0;
    for (i <-list) {
      sum += i
    }
    sum
  }

  // oder
  def summation_(list : List[Int]): Int = list.foldLeft(0)(_+_)

}
```

```
import scala.collection.mutable.{ArrayBuffer, ListBuffer}

object MapExample {

  // direkt uebersetzt
  def addAmount(list: List[Int], amount: Int) : List[Int] = {
    val retval = new ArrayBuffer[Int](list.length);
    for (i <- list) {
      retval += i+amount;
    }
    retval.toList;
  }

  // besser
  def addAmount_1(list: List[Int], amount: Int) : List[Int] = {
    val retval = ListBuffer[Int]()
    for (i <- list) {
      retval += i+amount;
```

```
    }
    retval.toList;
  }

  // oder
  def addAmount_2(list: List[Int], amount: Int) : List[Int] = {
    var retval = List[Int]()
    list.foreach(x => { retval = x+amount :: retval } )
    retval
  }


  // kurz:
  def addAmount_3(list: List[Int], amount: Int) : List[Int] = list.map(_+amount)



  def main(args: Array[String]): Unit = {
    println(addAmount(List(1,2,3,4,5,6,7,8,9,0),5))
    println(addAmount_1(List(1,2,3,4,5,6,7,8,9,0),5))
    println(addAmount_2(List(1,2,3,4,5,6,7,8,9,0),5))
    println(addAmount_3(List(1,2,3,4,5,6,7,8,9,0),5))
  }

}
```

```
  def getGreaterThanAmount(list: List[Integer], amount: Int) : List[Integer] =
    list.filter(_>amount)
```

Aufgabe 8

Ein Polynom

$a_0 + a_1x + a_2x^2 + ...a_nx^n$

kann nach dem Hornerschema berechnet werden als:

$a_0 + x(a_1 + x(a_2 + x(...xa_n)))$

Schreiben Sie eine Funktion `horner(x, koef)` die den Wert eines Polynoms aus x und den als Liste vorliegenden Koeffizienten $a_0, a_1, \cdots a_n$ berechnet. Verwenden Sie dazu Faltungs- oder Reduktionsoperatoren.

Lösungshinweis

```
object Horner extends App {

  def horner(x: Double, koef: List[Double]): Double =
     koef.reduce((acc, v) => acc * x + v)

  println(horner(3, List(1, 4, 1, -6))) // => 60

}
```

Aufgabe 9

Schreiben Sie eine Funktion f, die zwei Listen von Ints als Argument hat und – falls beide Listen die gleiche Länge haben – die Liste mit Tripel liefert, Das i-te Tripel soll aus dem i-ten Element der ersten, dem i-ten Element der zweiten und deren Summe bestehen.

Beispiel: f(List(1,2), List(10, 11)) = List((1,10,11),(2 11,13))

Lösungshinweis

```
object Triples extends App {

  def f(l1: List[Int], l2: List[Int]) : List[(Int, Int, Int)] =
    l1.zip(l2).map( x => (x._1, x._2, x._1+x._2))

  println(f(List(1,2), List(10, 11)))
}
```

Aufgabe 10

Schreiben Sie eine Funktion *mkF*, die eine ganze Zahl n als Argument hat und eine Funktion liefert, die zwei ganze Zahlen a und b als Argument hat und $n * a + b$ liefert. Ergänzen Sie dazu:

```
object Aufgabe_9 extends App {

  def mkF(n: Int) : (Int, Int) => Int = ???

  val f = mkF(5)
  println(f(2, 3)) // 13
  println(mkF(5)(2, 3)) // 13

}
```

Lösungshinweis

```
object Aufgabe_10 extends App {

  def mkF(n: Int) : (Int, Int) => Int = (a: Int, b: Int) => n*a + b

  val f = mkF(5)
  println(f(2, 3)) // 13
  println(mkF(5)(2, 3)) // 13

}
```