Übung 5 zu KMPS — Lösung

Aufgabe 29:
Implementieren Sie eine Scala-Funktion `palindrom`, die überprüft, ob es sich bei einer Liste um ein Palindrom handelt auf der Scala-internen Listenstruktur.
Überlegen Sie, welche Funktion Sie dabei sinnvollerweise verwenden und implementieren Sie diese als nested Funktion von `palindrom`.

Lösung:
```
def palindrom(xs:List[Any]) : Boolean = {
  def reverse(xs:List[Any]) : List[Any] = xs match {
    case Nil => Nil
    case x::x1s => reverse(x1s):::x::Nil
  }
  reverse(xs) == xs
}

// Alternativ: Überprüfe, ob 1. und letztes Element gleich sind:
def palindrom(xs:List[Any]) : Boolean = xs match {
  case Nil => true
  case x::Nil => true
  case x+:x1s:+y => ((x==y) && palindrom(x1s))
}
```

Aufgabe 30:
Implementieren Sie eine Scala-Funktion `zip`, die die Elemente zweier Integerlisten an gleicher Position mit der Funktion `f` verknüpft, und eine Liste der verknüpften Elemente zurückgibt.

Lösung:
```
def zip(f:(Int,Int) => Int, xs:List[Int], ys:List[Int]) : List[Int] = xs
match {
  case Nil => Nil
  case x::x1s => ys match {
    case Nil => Nil
    case y::y1s => f(x,y) :: zip(f,x1s,y1s)
  }
}
```

Aufgabe 31:
Erweitern Sie die Definition von Binärbäumen aus Aufgabe 9, so dass diese polymorph sind.

Lösung:
```
abstract class BinTree[+A]
case object EmptyTree extends BinTree[Nothing]
case class Node[A](elem:A,left:BinTree[A],right:BinTree[A]) extends
BinTree[A]
```

Übung 5 zu KMPS **Lösung**

Aufgabe 32:
Implementieren Sie eine Funktion `schalte`, die zwischen den Farben einer Ampel schaltet.

Lösung:
```
def schalte(farbe: Ampel) : Ampel = farbe  match {
  case Rot => RotGelb
  case RotGelb => Grün
  case Grün => Gelb
  case Gelb => Rot
}
```

Aufgabe 33:
a) Implementieren Sie analog zu Aufgabe 23 eine Funktion `foldr`, die für alle nichtleeren Integer-Listen mittels einer zweistelligen Funktion `f` alle Listenelemente von rechts nach links verknüpft. Dabei wird bei der leeren Liste der anzugebenden Startwert zurückgegeben.
Bsp.: Falls es sich bei `f` um die Subtraktion handelt, werden alle Listeneinträge von rechts nach links vom Startwert abgezogen.

b) Wie muss man `foldr` aufrufen, um die Subtraktion der Listenelemente vom Startwert zu erhalten?

Lösung:
```
a)
def foldr(f:(Int,Int) => Int, start: Int, xs: List[Int]) : Int =
  xs match {
    case Nil => start
    case y::ys => f(foldr(f,start,ys),y)
  }
b)
foldr((x,y) => x-y,0,1::2::3::4::Nil)   -> -10
```

Aufgabe 35:
a) Machen Sie aus der Funktion `mapBT` aus Aufgabe 26 eine polymorphe Funktion, so dass sie möglichst allgemein verwendbar ist.

b) Wie muss man die Funktion aus a) aufrufen, um alle Integer-Knoteneinträge, die gleich 6 sind, durch „Nikolaus" zu ersetzen und ansonsten, die Werte beizubehalten?

Lösung:
```
a)
def mapBTGen[A,B](f: A => B, tree: BinTree[A]) : BinTree[B] = tree match  {
  case EmptyTree => EmptyTree
  case Node(elem,left,right) =>
Node(f(elem),mapBTGen(f,left),mapBTGen(f,right))
}

b) def idNik(n:Int) : Any = if (n==6) "Nikolaus" else n
mapBTGen[Int,Any](idNik,Node(1,Node(6,EmptyTree,EmptyTree),Node(2,EmptyTree,
EmptyTree)))
```

Übung 5 zu KMPS

Lösung

Aufgabe 41:
a) Implementieren Sie eine Higher-Order-Funktion `prod`, die das Produkt von Funktionswerten im Bereich von a bis b liefert.

b) Implementieren Sie eine Higher-Order-Funktion `prodCurry`, indem Sie auf `prod` aus a) Currying auf das 1. Argument anwenden.

c) Definieren Sie mit der Funktion aus b) eine Funktion, die das Produkt der Integerzahlen zwischen a und b liefert.

d) Definieren Sie mit der Funktion aus c) eine Funktion, die die Fakultät einer positiven Integerzahl berechnet.

Lösung:

```
a)
def prod(f: Int => Int, a: Int, b: Int): Int =
  if (a > b) 1 else f(a) * prod(f,a + 1, b)
prod(x=>2*x, 2,4)

b)
def prodCurry(f: Int => Int): (Int, Int) => Int = {
  def prodF(a: Int, b: Int): Int =
    if (a > b) 1 else f(a) * prodF(a + 1, b)
  prodF
}
prodCurry(x=>2*x)(2,4)

c)
def prodInts = prodCurry(x=>x)
prodInts(2,4)

d)
def fak(n:Int) = prodInts(1,n)
fak(4)
```