Übung 4 zu KMPS — Lösung

Aufgabe 21:
a) Welchen Platzaufwand hat die `sum`-Funktion unten auf Folie „HigherOrderProgrammierung" 3?

b) Implementieren Sie `sum` tail-rekursiv, um den Platzaufwand konstant zu halten, indem Sie das folgende Programmgerüst ergänzen:

```
def sum(f: Int => Int)(a: Int, b: Int): Int = {
  def iter(a: Int, result: Int): Int = {
    if (??) ??
    else iter(??, ??)
  }
  iter(??,??)
}
```

c) Geben Sie den Aufruf von `sum` an, um die Summe der Zahlen zwischen 3 und 5 zu erhalten?

Lösung:
a) Linear, da `sum` nicht tail-rekursiv aufgerufen wird.

b)
```
def sum(f: Int => Int)(a: Int, b: Int): Int = {
  def iter(a: Int, result: Int): Int = {
    if (a > b) result
    else iter(a+1, result + f(a))
  }
  iter(a, 0)
}
```

c) `sum(x=>x)(3,5)`

Aufgabe 22:
a) Definieren Sie eine Scala-Funktion `add`, die die Summe der Elemente einer Integer-Liste liefert.

b) Definieren Sie ausschließlich unter Verwendung der Funktionen `add` und `map` eine Higher-Order Funktion `addMap`, die eine Funktion auf alle Listenelemente anwendet und dann die Summe aller erhaltenen Listenelemente liefert.

Lösung:
a)
```
def add(xs:List[Int]) : Int =
  xs match {
    case Nil => 0
    case y::ys => y+add(ys)
  }
```

b)
```
def addMap(f:Int => Int, xs:List[Int]) : Int = add(map(xs ,f))
```

Übung 4 zu KMPS — Lösung

Aufgabe 23:
a) Definieren Sie eine Higher-Order-Funktion `foldl`, die für alle nichtleeren Integer-Listen mittels einer zweistelligen Funktion `f` alle Listenelemente von links nach rechts verknüpft.
Dabei wird bei der leeren Liste der anzugebenden Startwert zurückgegeben.
Bsp.: Falls es sich bei `f` um die Addition handelt und der Startwert `0` ist, erhält man die Summe aller Listenelemente.

b) Wie muss man `foldl` aufrufen, um das Produkt der Listenelemente zu erhalten?

Lösung:
a)
```
def foldl(f:(Int,Int) => Int, start: Int, xs: List[Int]) : Int =
  xs match {
    case Nil   => start       //bei leerer Liste Rückgabe von start
    case h::ts => foldl(f,f(start,h),ts)
  }
```
b)
```
foldl((x,y) => x*y, 1, 1::2::3::Nil)
```

Aufgabe 24:
Implementieren Sie eine Funktion `range`, die eine Liste der Integerzahlen zwischen a und b erzeugt.

Lösung:
```
def range(a:Int,b:Int) : List[Int] = if (a>b) Nil else a::range(a+1,b)
```

Aufgabe 25:
Definieren Sie eine Scala-Funktion zur Addition zweier Integer-Zahlen durch currying und erläutern Sie, wie man damit welche anderen Funktionen definieren kann und wie man diese aufruft.

Lösung:
```
def add(x: Int) : Int => Int  = {
  def addHelp(y: Int): Int = x + y
  addHelp
}
```

Man kann add verwenden, um zum Beispiel eine Funktion zu definieren, die eine Konstante zu einer Integerzahl addiert.
```
def add5 : Int => Int = add(5)  //geht auch ohne Typ

add5(6) -> 11
```

Übung 4 zu KMPS **Lösung**

Aufgabe 26:
a) Erweitern Sie die Funktion `map` auf Binärbäume mit Integerwerten als Einträge, so dass die Funktion `f` auf alle Knoten des Binärbaums angewendet wird.

b) Wie muss man das erweiterte `map` aufrufen, um die Einträge des Binärbaums zu verdoppeln?

Lösung:
a)
```
def mapTree(f: Int => Int, tree: BinTreeInt) : BinTreeInt = tree match  {
  case EmptyTree => EmptyTree
  case Node(elem,left,right) =>
Node(f(elem),mapTree(f,left),mapTree(f,right))
}
```

b)
```
mapTree(x => 2*x,
Node(1,Node(2,EmptyTree,EmptyTree),Node(3,EmptyTree,EmptyTree)))
```
➔ `Node(2,Node(4,EmptyTree,EmptyTree),Node(6,EmptyTree,EmptyTree))`

Aufgabe 27:
a) Erweitern Sie die Funktion `filter` auf Binärbäume mit Integerwerten als Einträge, so dass eine Liste die Knoten des Binärbaums in Präorderreihenfolge enthält, die die boolesche Funktion erfüllen.

b) Wie muss man das erweiterte `filter` aufrufen, um alle geraden Einträge des Binärbaumes in der Ergebnisliste zu speichern?

Lösung:
a)
```
def filter(p:Int => Boolean, xb: BinTreeInt) : List[Int] =
  xb match {
    case EmptyTree => Nil
    case Node(elem,left,right)  =>  if (p(elem))
                                      elem::filter(p,left):::filter(p,right)
                                    else filter(p,left):::filter(p,right)
}
```

b)
```
filter((x:Int)=>(x%2==0),Node(1,Node(2,EmptyTree,EmptyTree),Node(3,Node(4,Em
ptyTree,EmptyTree),EmptyTree)))  ->  2::4::Nil
```

Aufgabe 28:
Implementieren Sie eine Scala-Funktion `infix`, die überprüft, ob eine Liste `xs` in einer Liste `ys` enthalten ist. Verwenden Sie dabei die Scala-interne Listenstruktur. Sie dürfen dabei die Funktion `präfix` aus Aufgabe 12 verwenden (`präfix(xs,ys)` bedeutet, dass `xs` Präfix von `ys` ist).

Lösung:
```
def infix(xs:List[Any],ys:List[Any]) : Boolean =
  ys match {
    case Nil => (xs == Nil)
    case z::zs => präfix(xs,ys) || infix(xs,zs)
}
```