PFP-Team
pfp@i2.cs.fau.de

Übungsblatt 12

Abgabe bis 24.07.2019, 18:00 Uhr

Hinweise:

- Verwenden Sie zur Lösung der Aufgaben nur die aus der Vorlesung bekannten, sowie die in den Übungen bekannt gegebenen Methoden und Funktionen der *Scala*-Standardbibliothek.
- Achten Sie bei der Bearbeitung der *Scala*-Aufgaben darauf, dass alle von Ihnen erstellten Methoden und Funktionen keine Seiteneffekte haben dürfen. Diese Regelung gilt sowohl für die Übungsaufgaben (Blätter 10–12) als auch für die Klausur!
- Nutzen Sie nur unveränderliche (immutable) Collection-Klassen. Die Verwendung von Klassen aus dem Paket `scala.collection.mutable` ist untersagt.
- Verwenden Sie nur das Scala-Schlüsselwort `val` zur Deklaration von Attributen und Konstanten. Die Verwendung von `var` ist untersagt.
- Geben Sie Ihre Lösungen für die Blätter 10–12 über die EST-Veranstaltung *PFP (F)* ab.

Aufgabe 12.1: Faltung

Implementieren Sie in der Datei „Fold.scala“ im `object Fold` folgende Funktionen ausschließlich unter Verwendung von Mustervergleich und Rekursion (die Verwendung von Listenmethoden der Standardbibliothek ist dabei untersagt, `::` ist erlaubt, wg. Mustervergleich):

a) `foldRight: String => ((Int, String) => String) => List[Int] => String`: Die Funktion `foldRight(ys)(f(x, zs))(xs)` iteriert über alle Elemente `x` der Liste `xs` und wendet die Funktion `f(x, zs)` auf jedes Element `x` (in der Reihenfolge von rechts nach links, d. h. beginnend beim letzten Element der Liste) an, wobei `zs` zu Beginn den Wert `ys` hat und für alle nachfolgenden Werte das Ergebnis des vorherigen Aufrufs von `f` ist.

Beispiel: `foldRight("<")((x, zs) => "(" + x + zs + ")")(List(1, 2, 3)) == "(1(2(3<)))"`.

b) `foldLeft: String => ((String, Int) => String) => List[Int] => String`: Die Funktion `foldLeft(ys)(f(zs, x))(xs)` iteriert über alle Elemente `x` der Liste `xs` und wendet die Funktion `f(zs, x)` auf jedes Element `x` (in der Reihenfolge von links nach rechts, d. h. beginnend beim Index 0) an, wobei `zs` zu Beginn den Wert `ys` hat und für alle nachfolgenden Werte das Ergebnis des vorherigen Aufrufs von `f` ist.

Beispiel: `foldLeft(">")((zs, x) => "(" + zs + x + ")")(List(1, 2, 3)) == "(((>1)2)3)"`.

Aufgabe 12.2: Lazy Evaluation

a) Was versteht man unter „Lazy Evaluation“?

b) Was lässt sich dank Lazy Evaluation erstellen, was in Programmiersprachen ohne dieses Konzept nicht möglich ist?

Implementieren Sie in der Datei „Lazy.scala“ im `object Lazy` folgende Funktionen mittels Lazy Evaluation unter Verwendung von Streams:

c) `natural: Stream[Long]`: Diese Funktion erstellt den unendlichen Stream der natürlichen Zahlen $\mathbb{N}$ (beginnend bei 1).

Beispiele:
```
natural(0) == 1
natural(1) == 2
natural(2) == 3
natural.take(3) == Stream(1, ?)
natural.take(3).toList == List(1, 2, 3)
```

d) `limit: Stream[Long] => Long => List[Long]`: Die Funktion `limit(s)(l)` ermittelt die Liste aller Zahlen n im aufsteigend sortierten Stream `s`, für die gilt $n \leq l$.

Beispiel: `limit(natural)(5) == List(1, 2, 3, 4, 5)`

e) `interval: Stream[Long] => (Long, Long) => List[Long]`: Die Funktion `interval(s)(k, l)` ermittelt die Liste aller Zahlen n im aufsteigend sortierten Stream `s`, für die gilt $k \leq n \leq l$.

Beispiel: `interval(natural)(20, 25) == List(20, 21, 22, 23, 24, 25)`

f) `factorials: Stream[Long]`: Erzeugt einen unendlichen Stream, so dass sich an Position `n` die `n`-te Fakultät befindet (beginnend bei 0!).

Beispiele: `factorials(0) == 1; factorials(2) == 2; factorials(5) == 120`

g) `euler: Int => Double`: Die Funktion `euler(n)` berechnet die Eulersche Zahl e in Näherung `n` gemäß folgender Formel: $\sum_{k=0}^{n} \frac{1}{k!}$. Verwenden Sie zur Berechnung der Fakultät die vorher von Ihnen implementierte Funktion `factorials: Stream[Long]` (eine Fakultätsfunktion, die `Int` zurückliefert, ist hier für die Berechnung nicht ausreichend).

Beispiele:
```
euler(10) == 2.7182815255731922
euler(15) == 2.71828182845823
euler(20) == 2.7182818284590455
euler(30) == 2.7182818284590455
```

Aufgabe 12.3: Parallelisierung

a) Welche Parallelisierungskonzepte für *Scala* kennen Sie aus der Vorlesung?

b) Implementieren Sie in der Datei „Par.scala“ folgende Funktion:
`pairSumsPar: (Int, Int) => List[(Int, Int)]`: Die Funktion `pairSumsPar(n, s)` ermittelt parallel die Liste aller Zahlenpaare mit Elementen von 0 bis `n` (inklusive), deren Summe `s` entspricht.

Bonusaufgabe 12.4: Schriftliches Wurzelziehen **6 Punkte**

In dieser Aufgabe setzen Sie einen Algorithmus zum Ziehen der quadratischen Wurzel mit beliebiger Genauigkeit in Scala um. Die Berechnung von $\sqrt{654{,}321}$ beispielsweise funktioniert so:

S1 Unterteile die Zahl vom Dezimaltrennzeichen aus in beide Richtungen in Gruppen zu je 2 Ziffern (die erste Gruppe ist ggf. kleiner und die letzte wird mit 0 aufgefüllt): $\sqrt{6|54,32|10|00|\ldots}$

S2 Die erste Ziffer der Quadratwurzel ist die größte Zahl a, deren Quadrat gerade noch kleiner oder gleich der ersten Gruppe ist: Da $2^2 \leq 6 < 3^2$ ist, lautet die erste Stelle also $a = 2$.

S3 Das Quadrat dieser ersten Ziffer wird von der ersten Gruppe abgezogen und an das Ergebnis wird die nächste Gruppe $\underline{54}$ angehängt: $b := (6 - a^2) \cdot 100 + \underline{54} = 2\underline{54}$

S4 Nun wird die größte Ziffer c gesucht, für die $(a \cdot 20 + c) \cdot c \leq b$ gerade noch gilt: Wegen $(2 \cdot 20 + 5) \cdot 5 \leq 254 < (2 \cdot 20 + 6) \cdot 6$ lautet die zweite Stelle der Wurzel daher $c = 5$.

S5 Die Zwischenergebnisse b und a werden mit $d := (a \cdot 20 + c) \cdot c$ aus **S4** und der nächsten Gruppe $\underline{32}$ aktualisiert: $b \leftarrow (b - d) \cdot 100 + \underline{32} = 29\underline{32}$ und $a \leftarrow (a \cdot 10 + c) = 25$.

S6 Anschließend wiederholt man die Schritte **S4** und **S5** *ad infinitum*, wobei man nach der letzten Gruppe „beliebig“ lange mit $\underline{00}$ auffüllt. Sobald die nächste Gruppe unmittelbar „hinter“ dem Dezimalpunkt steht, wird auch in der Lösung der Dezimalpunkt gesetzt.

Der weitere Ablauf ab $a = 25$, $b = 29\underline{32}$ und $\sqrt{6|54,\underline{32}|10} = 25,\ldots$ lautet also:

▶ $(25 \cdot 20 + 5) \cdot 5 \leq 29\underline{32} < (25 \cdot 20 + 6) \cdot 6 \rightsquigarrow c = 5$
▷ $b \leftarrow 407\underline{10}$ sowie $a \leftarrow 255$ $\sqrt{6|54,32|\underline{10}} = 25,5\ldots$
▶ $(255 \cdot 20 + 7) \cdot 7 \leq 407\underline{10} < (255 \cdot 20 + 8) \cdot 8 \rightsquigarrow c = 7$
▷ $b \leftarrow 4961\underline{00}$ sowie $a \leftarrow 2557$ $\sqrt{6|54,32|10} = 25,57\ldots$
▶ $(2557 \cdot 20 + 9) \cdot 9 \leq 4961\underline{00} < (2557 \cdot 20 + 10) \cdot 10 \rightsquigarrow c = 9$
▷ $b \leftarrow 3575\underline{900}$ sowie $a \leftarrow 25579$ $\sqrt{6|54,32|10} = 25,579\ldots$
▶ $(25579 \cdot 20 + 6) \cdot 6 \leq 3575\underline{900} < (25579 \cdot 20 + 7) \cdot 7 \rightsquigarrow c = 6$
▷ $b \leftarrow 5063840\underline{0}$ sowie $a \leftarrow 255796$ $\sqrt{6|54,32|10} = 25,5796\ldots$
...

Implementieren Sie die folgenden Funktionen in der Datei „Sqrt.scala“ im `object Sqrt`.

a) `split1: BigInt => Stream[BigInt]`: Die Funktion `split1(i)` zerlegt die Zahl `i` in Zweiergruppen und liefert deren Werte als Strom zurück, beginnend mit der höchstwertigen Zweiergruppe. Falls `0` übergeben wird, muss ein leerer Strom zurückgegeben werden. Sie können davon ausgehen, dass $i \geq 0$ gilt.

Beispiele:

```
split1(0).toList == List()
split1(65005).toList == List(6, 50, 5)
```

b) `split2: BigDecimal => Stream[BigInt]`: Die Funktion `split2(d)` zerlegt die Nachkommastellen von `d` in Zweiergruppen und liefert deren Werte als *unendlichen* Strom zurück, beginnend mit der höchstwertigen Zweiergruppe. Sie können davon ausgehen, dass $0 \leq d < 1$ gilt.

Tipp: Sie erhalten den ganzzahligen Anteil einer `BigDecimal`-Instanz mit der Methode `toBigInt`.

Beispiele:

```
split2(0).take(3).toList == List(0, 0, 0)
split2(0.65005).take(5).toList == List(65, 0, 50, 0, 0)
```

c) `split: BigDecimal => Stream[BigInt]`: Die Funktion `split(d)` zerlegt sowohl den ganzzahligen Anteil der Zahl `d` als auch deren Nachkommastellen unter Verwendung von `split1` und `split2` in einen unendlichen Strom von Zweiergruppen. Die Kommastelle soll darin an der richtigen Position durch die Zahl `-1` repräsentiert werden. Sie können davon ausgehen, dass $d \geq 0$ gilt.

Beispiele:

```
split(0).take(3).toList == List(-1, 0, 0)
split(500.005).take(6).toList == List(5, 0, -1, 0, 50, 0)
```

d) `findC: (BigInt, BigInt) => BigInt`: Die Funktion `findC(a, b)` ermittelt die größte Ziffer c, für die $(a*20+c)*c \leq b$ gerade noch gilt. Verwenden Sie zur Ermittlung keine Funktion zum Wurzelziehen (`scala.math.sqrt`, o.ä.), da es sich bei dieser Bonusaufgabe ja um ein *schriftliches* Verfahren handeln soll. Sie können davon ausgehen, dass $a \geq 0$, $b \geq 0$ und $0 \leq c \leq 9$ gilt.

e) `sqrtHelper: Stream[BigInt] => (BigInt, BigInt, BigInt, BigInt) => Stream[BigInt]`: Die Funktion `sqrtHelper(zs)`$(a_{n-1}, b_{n-1}, c_{n-1}, d_{n-1})$ berechnet die Quadratwurzel r der Zahl z, deren Stellen z_n ($n \leq 0$) als Zweiergruppen im unendlichen Stream `zs` gegeben sind (`zs = split(z)`). Das Ergebnis ist ein unendlicher Strom der einzelnen Ziffern r_n von r, wobei das Komma durch die Zahl `-1` repräsentiert wird. Die Berechnung einer Ergebnisziffer r_n erfolgt dabei aus den gegebenen Parametern $(a_{n-1}, b_{n-1}, c_{n-1}, d_{n-1})$ wie folgt:

- $a_{-1} = 0, b_{-1} = 0, c_{-1} = 0, d_{-1} = 0$
- Falls $z_n = -1$: $a_n = a_{n-1}, b_n = b_{n-1}, c_n = c_{n-1}, d_n = d_{n-1}, r_n = -1$
- Falls $z_n \neq -1$:
  * $a_n = a_{n-1} * 10 + c_{n-1}$
  * $b_n = (b_{n-1} - d_{n-1}) * 100 + z_n$
  * $c_n =$ `findC`(a_n, b_n)
  * $d_n = (a_n * 20 + c_n) * c_n$
  * $r_n = c_n$

Die einzelnen Berechnungsschritte am Beispiel $\sqrt{654,3210}$ können Sie der Beschreibung im Eingangstext entnehmen. Achten Sie darauf, dass Ihre Implementierung von `sqrtHelper` sich rekursiv wiederholt aufruft.

Beispiel:

```
sqrtHelper(split(654.321))(0, 0, 0, 0).take(7).toList ==
                List(2, 5, -1, 5, 7, 9, 6)
```

f) `sqrt: BigDecimal => Stream[Char]`: Die Funktion `sqrt(d)` berechnet die Quadratwurzel von `d` mit beliebiger Genauigkeit. Der unendliche Ergebnisstrom enthält jeweils eine Ziffer des Ergebnisses, das Komma wird an der richtigen Stelle als `','` zurückgegeben.

Beispiele:

```
sqrt(654.321).take(10).mkString == 25,5796989
sqrt(2).take(100).mkString == 1,41421356237309504880168872420969807856967187537694807317667973799073247846210703885038753432764157
```

6 Punkte