Christoph Lüth
Martin Ring

1. Übungsblatt

Ausgabe: 29.04.14
Abgabe: 13.05.14

Dieses Übungsblatt basiert auf den Inhalten der Vorlesungen vom 29. April sowie vom 6. Mai. Wenn Sie also zunächst bei der Lösung gar nicht weiter kommen, verzweifeln Sie nicht sondern warten sie die zweite Vorlesung ab.

Wir empfehlen die Scala-IDE (`http://scala-ide.org/download/sdk.html`) zur Bearbeitung.

1.1 *Fingerübung für NSA Mitarbeiter* *8 Punkte*

In der Veranstaltungsbeschreibung haben wir versprochen, dass die Veranstaltung Sie für die NSA qualifiziert. Jeder NSA Agent zerlegt vor dem Frühstück 1000 Zahlen in ihre Primfaktoren, das sollten Sie auch beherrschen:

Schreiben Sie die beiden Funktionen `isPrime` und `primeFactors`:

```
def isPrime(int: Int): Boolean = ???
def primeFactors(int: Int): List[Int] = ???
```

Nutzen sie nun ihre Implementation von `isPrime` um die Funktion `primeFactors` zu überprüfen. Folgendes sollte für alle $x \geq 1$ gelten:

```
(primeFactors(x) == Nil) == (x == 1)
primeFactors(x).forall(isPrime)
primeFactors(x).product = x
```

Anmerkung: Die Zeilen mit dem `???` kompilieren tatsächlich so. `???` ist in der Standardbücherei definiert und hat den Typ `Nothing` (Untertyp von allen anderen Typen). bei der Ausführung fliegt uns das natürlich um die Ohren.

1.2 *Pimp my Int* *4 Punkte*

Ein beliebtes Muster in Scala ist das sogenannte *Pimp-My-Library*-Pattern. Durch eine implizite Konversion (Ein zunächst gewöhnungsbedürftiges Scala Feature) kann ein bestehender Typ *verwandelt* werden. Das funktioniert so: Bei einem Typfehler gibt der Scala Compiler nicht gleich auf, sondern sucht nach (im aktuellen Kontext sichtbaren) Methoden die mit dem Modifier *implicit* markiert sind. Wenn genau eine dieser Methoden dazu geeignet ist, den Typfehler zu lösen, wird sie auf den falsch getypten Ausdruck angewandt. Auf diese Weise ist es möglich, zusätzliche eigene Methoden auf bestehenden Typen zu definieren oder bestehende Typen von inkompatiblen Bibliotheken miteinander kompatibel zu machen.

Ein Beispiel:

```
object Example extends App {
  import scala.language.implicitConversions

  class PimpedString(underlying: String) {
    def makeImportant =
      underlying.toUpperCase + "!!!111oneone"
  }

  implicit def stringToPimpedString(value: String) = new PimpedString(value)

  println("hallo".makeImportant) // > HALLO!!!111oneone
}
```

Benutzen sie das Pattern um die Funktionen `isPrime` und `primeFactors` (beide diesmal ohne Parameter) auf `Int` zu definieren.

1.3 *Text Mining Light* *8 Punkte*

Neben der Primfaktorzerlegung verbringt ein NSA Agent die meiste Zeit damit in fremden Dokumenten herumzuschnüffeln. Damit das effizienter funktioniert, definieren sie einen Algebraischen Datentyp `TextElement` zur Repräsentation von Dokumenten. Der Typ sollte Überschriften, Formatierung (Fett, Kursiv, Unterstrichen), Absätze, Listen und Hyperlinks in beliebiger Verschachtelung unterstützen.

Definieren sie nun folgende rekursive Funktionen:

```
/**
 * Gibt das Dokument formatiert auf die Konsole aus. Formatierungen können Sie
 * dabei textuell markieren. (z.B. "**Fetter Text**") Verschachtelte
 * Absätze sollen Eingerückt werden.
 */
def prettyPrint(doc: TextElement) = ???

/**
 * Sucht alle Links aus dem Dokument heraus und gibt sie als Liste zurück.
 */
def getHyperlinks(doc: TextElement): List[String] = ???

/**
 * Bringt alle Elemente auf eine einzige Verschachtelungsebene. (Eine Liste von
 * Elementen die keine weiteren Listen von Elementen enthält)
 */
def flatten(doc: TextElement): TextElement = ???

/**
 * Sortiert alle Listen aufsteigend alphabetisch und gibt das Dokument ansonsten
 * unverändert zurück.
 */
def sortLists(doc: TextElement): TextElement = ???
```