

# Einführung in die Informatik II

31.01 und 03.02.2020

## 1 IBAN Verifikation

In der Vorlesung (Kap. 2, Folie 5) wurde gezeigt, wie die Gültigkeit einer Kontonummer mit Hilfe einer Prüfsumme überprüft werden kann. Diese Aufgabe soll nun eine Validierung für IBANs geschrieben werden.

Eine IBAN besteht aus:

- 2-stelliger Ländercode (Großbuchstaben)
- 2-stellige Prüfziffer (Ziffern)
- max. 30-stellige Kontonummer (Großbuchstaben oder Ziffern)

Um eine IBAN zu modellieren, soll in dieser Aufgabe folgender Typ verwendet werden.

```
1 | type IBAN = Array[Char]
```

Um eine IBAN auf ihre Gültigkeit zu prüfen, wird diese in eine Zahl umgewandelt und diese  $\text{mod}97$  genommen. Wenn die IBAN gültig ist, muss das Ergebnis 1 sein. Die Umwandlung in eine Zahl erfolgt folgendermaßen:

- Die ersten vier Zeichen (Ländercode und Prüfziffer) werden an das Ende der Kontonummer verschoben.
- Die in der IBAN enthaltenen Buchstaben werden entsprechend ihrer Position im Alphabet + 10 in Ziffern umgewandelt. Also  $A = 10$ ,  $B = 11$ , ...,  $Z = 35$ .
- Die so erstellte, nur aus Ziffern bestehende, Zeichenkette wird in einen Integer umgewandelt, der durch  $\text{mod}97$  geprüft werden kann.

Im Folgenden soll die Validierungsfunktion Schritt für Schritt entwickelt werden.

a) Vervollständigen Sie folgende Funktion, die die ersten 4 Zeichen einer übergebenen IBAN an das Ende verschiebt.

```
1 | def rearrange(iban: IBAN): Array[Char] = {  
2 |     if(iban.length < 5) {  
3 |         // Damit die IBAN den Ländercode, Prüfziffer und eine  
4 |           Kontonummer enthält muss sie mindestens 5 Stellen haben.  
5 |         // Ergänzen Sie hier die Fehlerbehandlung und lösen Sie eine  
6 |           Exception vom Typ Exception aus.  
7 |     }  
8 |     var rearranged = new Array[Char](iban.length) //Kopie des IBAN-  
9 |       Arrays zum modifizieren  
10 |    // Verschieben der ersten 4 Zeichen  
11 |    // ...  
12 |    return rearranged  
13 | }
```

*Hinweis:* Zum Verschieben können Sie die in der Vorlesung vorgestellte Funktion `Array.copy` verwenden.

- b) Implementieren Sie nun eine Funktion, die alle in einem `Array[Char]` vorkommenden Buchstaben, entsprechend des in der Aufgabenstellung gegebenen Schemas (Position im Alphabet + 10, A = 10, ..., Z = 35) umwandelt.

Bei einem `Char` kann über die Funktionen `.isDigit` oder `.isLetter` überprüft werden, ob das Zeichen eine Zahl oder ein Buchstabe ist. Außerdem kann ein `Char`, der ein Buchstabe ist, mit der Funktion `.getNumericValue` wie für diese Aufgabe benötigt, in eine Zahl umgewandelt werden:

```
1 | 'A'.getNumericValue = 10 : Int
```

Um aus dem `Array[Char]` einen `String` zu erzeugen können Sie die Funktion `.mkString` verwenden. Den `String` können Sie dann wie folgt in einen `BigInt` umwandeln. Ein `BigInt` ist hier notwendig, da die aus der IBAN resultierende Zahl zu groß für einen normalen Integer sein kann.

```
1 | BigInt(Array[Char]('1','2').mkString)
```

Beim Erstellen des `BigInt` kann eine `NumberFormatException` ausgelöst werden, falls der gegebene `String` nicht in einen Integer umgewandelt werden kann. Verwenden Sie einen `try, catch` Block um diese Exception abzufangen und lösen Sie in dem `catch`-Teil eine eigene Exception zur Fehlerbehandlung aus.

Füllen Sie im gegebenen Funktionsgerüst die mit dem Platzhalter `???` markierten Stellen.

```
1 | def convert2int(sa: Array[Char]): BigInt = {
2 |   var lc = 0 // letter count
3 |
4 |   ???
5 |
6 |   // Neues Array mit Platz für die
7 |   // expandierten Buchstaben
8 |   var converted = new Array[Char](sa.length+lc)
9 |
10 |  ???
11 |  for(i <- 0 until sa.length) {
12 |    ???
13 |  }
14 |
15 |  ???
16 |  return BigInt(converted.mkString)
17 |  ???
18 | }
```

- c) Vervollständigen Sie nun die Funktion, die eine IBAN überprüft und dabei im Fehlerfall die Exceptions abfängt und deren Nachricht als Text ausgibt.

```
1 | var iban1 : IBAN = Array[Char]('G', 'B', '8', '2', 'W', 'E', 'S', 'T',
2 |   , '1', '2', '3', '4', '5', '6', '9', '8', '7', '6', '5', '4', '3',
3 |   '2')
4 | var iban2 : IBAN = Array[Char]('D', 'E', '1', '9', '1', '2', '3', '4',
5 |   , '1', '2', '3', '4', '1', '2', '3', '4', '1', '2', '3', '4', '1',
6 |   '2')
7 |
8 | def verify(iban: IBAN): Boolean = {
9 |   ???
10 |   try {
11 |     ???
12 |   } catch {
```

9		???
10		}
11		???
12		}

## 2 Sudoku

In dieser Aufgabe sollen einzelne Funktionen für Sudoku-Arrays (Vorlesung Kap. 1, Folie 49) implementiert werden. Ein Sudoku-Array ist wie folgt definiert:

```
1 | type Sudoku = Array[Array[Int]]
```

*Hinweis:* Anders als herkömmliche Sudokus werden die Elemente der Sudoku-Arrays mit den Zahlen von 0 bis 8 (und **nicht** von 1 bis 9) belegt.

- a) Vervollständigen Sie folgende Funktion, die feststellt, ob zwei Sudoku-Arrays den gleichen Inhalt haben.

```
1 | def gleich(s1: Sudoku, s2: Sudoku): Boolean = ...
```

*Hinweis:* Sie dürfen voraussetzen, dass beide Parameter 9 Zeilen mit je 9 Werten enthalten.

- b) Sei  $P$  eine Permutation von  $0, 1, 2, \dots, 8$  und sei  $S$  ein Sudoku-Array. Schreiben Sie eine Funktion `wendeAuf`, so dass der Aufruf `wendeAuf(P, S)` einen Sudoku-Array zurückgibt, der sich durch Anwendung von  $P$  auf jedes Element von  $S$  ergibt. Für welches  $Q$  ergibt `wendeAuf(Q, S)` einen Sudoku-Array, dessen erste Zeile `Array(0, 1, 2, 3, 4, 5, 6, 7, 8)` ist?
- c) Aus einer einzigen Sudoku-Lösung entstehen durch Anwenden von Permutationen  $9!$  Sudoku-Lösungen. Wir wollen nun feststellen, ob sich eine Sudoku-Lösung  $S2$  aus einer Sudoku-Lösung  $S1$  durch Anwenden einer Permutation ergibt, d.h. ob es eine Permutation  $P$  gibt mit  $S2 = P \circ S1$ .  
`gleich(S2, wendeAuf(P, S1))` ergibt „true“.

Um die Frage zu beantworten, ermitteln wir zunächst die (einzige!) Permutation  $Q$ , welche die erste Zeile von  $S1$  in die erste Zeile von  $S2$  überführt. Gilt für  $Q$ , dass

```
1 | gleich(S2, wendeAuf(Q, S1))
```

dann ist die Frage mit „ja“ zu beantworten, sonst mit „nein“. Implementieren Sie dieses Entscheidungsverfahren als Scala-Funktion.

*Tipp:* Die Anwendung von Permutationen, wie durch die Funktion `wendeAn` definiert (Vorlesung Kap. 1, Folie 47), verhält sich assoziativ: Sei  $\circ$  die Abbildung, die eine Permutation gemäß der Funktion `wendeAn` auf eine andere anwendet und seien  $P, Q$  und  $R$  Permutationen, dann gilt  $(P \circ Q) \circ R = P \circ (Q \circ R)$ .

- d) Beim Sudoku gibt es neben dem Sudoku-Array einen Array `kandidaten` (Vorlesung Kap. 1, Folie 52).

- Welches Element von `kandidaten` muss „true“ sein, damit die Zuweisung

```
1 | sudoku(i)(j) = k // (*)
```

erlaubt ist?

- Welche Elemente von `kandidaten` sind im Anschluss an diese Zuweisung auf „false“ zu setzen?

Vervollständigen Sie die folgende Funktion, die genau dann „false“ zurückgibt, wenn die Zuweisung `(*)` nicht zulässig ist; anderenfalls wird an den globalen Arrays `sudoku` die in den Spiegelstrichen beschriebenen Änderungen vorgenommen (und „true“ zurückgegeben).

```
1 | def setze(i: Int, j: Int, k: Int): Boolean = ...
```

### 3 Zwiebelringmatrix

In dieser Aufgabe soll eine quadratische Matrix mit der Kantenlänge  $n$  spiralförmig von 1 bis  $n^2$  durchnummeriert werden. Im folgenden soll eine Version „immer an der Wand entlang“ durch schrittweise Verfeinerung umgesetzt werden. Die durchnummerierende Matrix  $M$  ist durch eine zweistufige Reihung der Dimension  $(n+2) \times (n+2)$  (inkl. Überlaufrand  $-1$ ) gegeben.

Tabelle 1: Beispiel für eine Matrix mit Kantenlänge  $n = 3$

-1	-1	-1	-1	-1
-1	1	2	3	-1
-1	8	9	4	-1
-1	7	6	5	-1
-1	-1	-1	-1	-1

Gehen Sie von folgendem Pseudocode aus:

```
1 //Initialisiere Matrix
2 //Setze Position auf (1, 1)
3 //Initialisiere Richtung "nach rechts"
4 for (i <- 1 to n * n) {
5     //Besetze Position mit i
6     //Bestimme mithilfe der Richtung vorläufige neue Position
7
8     if (vorläufige Position auf Rand oder bereits besetzt) {
9         //Ändere Richtung (rechts um)
10        //Bestimme mithilfe der Richtung neue Position
11    } else {
12        //Übernimm vorläufige Position als neue Position
13    }
14 }
```

- Implementieren Sie zunächst eine Funktion `init`, die zu einer gegebenen Dimension  $n$  eine zweistufige Reihung der Dimension  $(n+2) \times (n+2)$  (inkl. Überlaufrand) erstellt und deren Elemente mit 0 bzw. die Elemente des Überlaufrandes mit  $-1$  initialisiert sind.
- Überlegen Sie sich zuerst, wie sie die Anweisungen zur Initialisierung von Position und Richtung sowie deren Änderungen umsetzen können. Welche Hilfsdatenstrukturen und/oder -funktionen sind nützlich?
- Überführen Sie nun den Pseudocode in ein lauffähiges Scala-Programm.