Universität der Bundeswehr
Fakultät für Informatik – Institut 2
Prof. Dr. Wolfgang Hommel
Michael Grabatin, Michael Steinke

WT 2020
Übungsblatt 4

Einführung in die Informatik II

07. und 10.02.2020

1 Schach – Turmbedrohung

In dieser Aufgabe soll ein Scala-Programm entwickelt werden, welches die Koordinaten aller von weißen Türmen bedrohten Felder auf einem Schachbrett ausgibt und zu jedem dieser Felder angibt, ob es leer ist oder welche gegnerische Figur es enthält.

Folgender Pseudocode skizziert einen möglichen Lösungsansatz:

```
1 //Bestimme die Positionen aller weissen Tuerme auf einem Schachbrett
2
3 for (Position <- Positionen aller weissen Tuerme) {
4   //Gib alle bedrohten Felder oberhalb von Position aus
5   //Gib alle bedrohten Felder rechts von Position aus
6   //Gib alle bedrohten Felder unterhalb von Position aus
7   //Gib alle bedrohten Felder links von Position aus
8 }
```

Für die Implementierung sind folgende Definitionen vorgegeben:

```
1  object ChessmanType extends Enumeration {
2    type ChessmanType = Value
3    val King, Queen, Rook, Bishop, Knight, Pawn = Value
4  }
5
6  object Color extends Enumeration {
7    type Color = Value
8    val White, Black = Value
9  }
10
11 import ChessmanType._
12 import Color._
13
14 abstract class FieldContent
15 case class Chessman(cmt : ChessmanType, color : Color) extends
       FieldContent
16 case class Empty() extends FieldContent
17
18 def printField(row : Int, col : Int, fc : FieldContent) : Unit = {
19   println("(" + row + ", " + col + "): " + fc)
20 }
21
22 abstract class Position
23 case class Coord(row : Int, col : Int) extends Position
24 case class Undef() extends Position
25
```

```
26 type Chessboard = Array[Array[FieldContent]]
27
28 def emptyChessboard : Chessboard = Array.fill(8, 8)(Empty())
```

a) Implementieren Sie die Bestimmung der Positionen aller weißen Türme (Typ: `Rook`) als Methode. Zur weiteren Verwendung sind die Positionen in geeigneter Weise in einer Variablen zu speichern.

b) Implementieren Sie die Methode `nextField`, die zu einer gegebenen Position und einer angegebenen Richtung die nächstliegende Position bestimmt. Ist die Position dabei außerhalb des zulässigen Bereichs, ist `Undef()` zurückzugeben. Nutzen Sie folgende Definitionen:

```
1 type Direction = (Int, Int)
2 val LEFT = (0, -1)
3 val RIGHT = (0, 1)
4 val UP = (-1, 0)
5 val DOWN = (1, 0)
6
7 def nextField(p : Position, dir : Direction) : Position = ...
```

c) Implementieren Sie eine Hilfsprozedur, die zu einem übergebenem Schachbrett, einer angegebenen Position und einer gegebenen Richtung alle bedrohten Positionen und den jeweiligen Inhalt ausgibt.

d) Implementieren Sie die Prozedur `printThreatenedFields`, die alle von weißen Türmen bedrohten Felder ausgibt. Nutzen Sie die Teilergebnisse aus den vorherigen Teilaufgaben.

```
1 def printThreatenedFields(cb : Chessboard) : Unit = ...
```

e) Überlegen Sie sich, wie sich der durch den Pseudocode beschriebene Ansatz erweitern lässt, sodass er für alle „Linienfiguren“ (Turm, Läufer, Dame) anwendbar ist. Welche Änderungen sind in der Implementierung notwendig?

2 Flaggenproblem

Eine Reihung, bestehend aus Elementen des Aufzählungstyps `Farbe` (mit den Werten `Schwarz`, `Rot` und `Gold`) soll nach den Farben in der Reihenfolge `Schwarz-Rot-Gold` aufsteigend sortiert werden. Es sind zwei Lösungsansätze denkbar, bei denen sich die Aufteilung der vier Bereiche (*schwarz (s)*, *rot (r)*, *gold (g)* und *unsortiert (u)*) unterscheiden. Zum einen können die Bereiche in der Reihenfolge „*s-r-g-u*“aufgeteilt werden, alternativ sieht der zweite Ansatz die Aufteilung „*s-r-u-g*“vor.

a) Begründen Sie, inwiefern sich die Aufteilung der Bereiche positiv auf die Anzahl notwendiger Vertauschungen auswirkt.

b) Vervollständigen Sie die Prozedur `sortiereFlagge`, die eine übergebene Flagge gemäß dem zweiten Lösungsansatz sortiert:

Hinweis: Es ist von Vorteil sich hier eine Skizze mit den verschiedenen Indizes und möglichen Zuständen des Flaggen-Arrays zu machen.

```
1  object Farbe extends Enumeration {
2    type Farbe = Value
3    val Schwarz, Rot, Gold = Value
4  }
5  import Farbe._
6
7  type Flagge = Array[Farbe]
8
9  def sortiereFlagge(flag : Flagge) : Unit = {
10   val n = flag.length
11
12   // Initialisierung von Indizes für die (sortierten) Bereiche
13   // _s_chwarz, _r_ot und _g_old
14   var s = -1 // Ende des Bereichs schwarz
15   var r = s  // Ende des Bereichs rot
16   var g = n  // Anfang des Bereichs gold
17
18   // "unsortierter" Bereich ist zwischen dem Ende von rot und Anfang
19   // von gold -> solange Vertauschen bis der Bereich leer ist
20   while (r + 1 != g){
21     //Verkleinere Indexbereich von r + 1 bis g - 1 durch
22     //Vertauschen und Aktualisieren der Indizes
23     ...
24   }
25 }
```