Prof. Dr. Manfred Schmidt-Schauß
M.Sc. Nils Dallmeyer

Goethe-Universität Frankfurt am Main/ Institut für Informatik
Künstliche Intelligenz/Softwaretechnologie

Grundlagen der Programmierung 2

Sommersemester 2018

Aufgabenblatt Nr. 1

Abgabe: Mittwoch 18. April 2018 **vor!** der Vorlesung

Aufgabe 1 (0 Punkte)

Lesen Sie die „Allgemeinen Hinweise“ auf der Webseite zur Vorlesung:

`http://www.informatik.uni-frankfurt.de/~prg2/SS2018/#Aufgaben`

Aufgabe 2 (10 Punkte)

Implementieren und testen Sie eine Funktion `note` in Haskell, die 4 Eingaben erhält und die Note eines Studierenden für die PRG-2-Klausur berechnet, wobei gilt:

- Die ersten beiden Eingaben sind jeweils Wahrheitswerte (`True` oder `False`) und geben an, ob der Studierende im ersten Teil bzw. im zweiten Teil eine Aufgabe in der Übung vorgerechnet hat.
- Die dritte Eingabe ist eine Kommazahl an Bonuspunkten (zwischen 0 und 20), die der Studierende für die Klausur erhält, sofern er in beiden Teilen vorgerechnet hat.
- Die vierte Eingabe ist eine Kommazahl an Klausurpunkten (zwischen 0 und 100), die der Studierende in der Klausur erreicht hat (ohne Bonuspunkte).

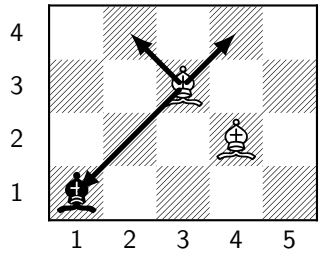
Wie sich die Note genau berechnet ist Ihnen bereits aus Aufgabe 1 bekannt.
Einige Beispielaufrufe:

```
*> note True True 16.66 74.5
1.0
*> note True True 15 66
1.7
*> note False True 20 66
2.7
*> note True False 20 49
5.0
*> note True True 15 35
5.0
*> note True True 15 40
3.7
```

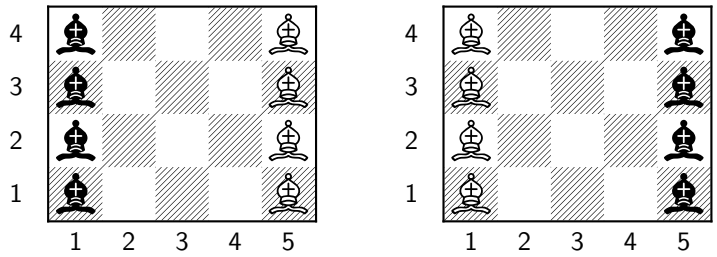
Aufgabe 3 (30 Punkte)

In dieser Aufgabe betrachten wir ein Rätsel auf einem 5×4 großen Schachbrett. Zur einfacheren Implementierung sind die Felder nicht wie sonst üblich durch eine Kombination aus Buchstabe und Ziffer benannt, sondern mit zwei Ziffern, wobei zuerst die horizontale Position genannt wird und dann die vertikale – statt `d3` verwenden wir also `43` als Feldangabe. Auf dem Spielfeld befinden sich nur Läufer, dabei darf ein Läufer ziehen wie im Schach:

Ein Läufer darf nur diagonal ziehen, kann nicht über andere Figuren drüberspringen, nicht auf Felder mit Läufern der selben Farbe ziehen und darf auf Felder mit Läufern der gegnerischen Farbe ziehen und diese zugleich schlagen, also vom Spielbrett entfernen. Falls ein Läufer mit einem einzigen Zug einen anderen schlagen kann, sagt man auch, dass er ihn *bedroht*. Die möglichen Züge sind in der folgenden Abbildung veranschaulicht, in welcher der weiße Läufer auf `33` den schwarzen Läufer auf `11` bedroht und weder nach `42` noch nach `51` ziehen darf, da auf `42` ein weißer Läufer steht.



Das Ziel des Rätsels ist das Finden einer Zugfolge, um ausgehend von der Stellung in der linken Abbildung die Stellung in der rechten Abbildung zu erreichen, wobei in jedem Zug ein beliebiger Läufer ziehen darf (die Farbe spielt keine Rolle) und zu keinem Zeitpunkt ein Läufer bedroht sein darf.



Das Spielfeld entspricht einer Matrix, die durch eine zweistellige Funktion f modelliert wird, wobei $f\ x\ y$ das Matrix-Element in Spalte x und Zeile y darstellt. Dabei steht ein Matrix-Element mit Wert `'w'` für einen weißen Läufer, `'s'` für einen schwarzen Läufer und `' '` für ein leeres Feld. Beachten Sie, dass man im GHCi keine Funktion anzeigen kann. Zur Anzeige eines Spielfelds kann die Funktion `zeigeFeld` verwendet werden, die in der Datei `blatt1.hs` definiert ist und ein Spielfeld auf die Kommandozeile ausgibt. Diese Datei findet sich beim Aufgabenblatt 1 auf der Webseite zur Vorlesung. Die in dieser Datei definierte Funktion `feldA` entspricht dem linken Spielfeld.

Die Verwendung von Listenfunktionen und Listensyntax ist bei allen Teilaufgaben **verboten**.

a) Implementieren und testen Sie in Haskell eine Funktion `istZugDiagonal`, die als erstes Argument die x-Koordinate der zu ziehenden Figur, als zweites die dazugehörige y-Koordinate, als drittes die x-Koordinate des Zielfelds und als viertes die dazugehörige y-Koordinate enthält und genau dann `True` zurückliefert, falls der Zug diagonal ist. (5 Punkte)

Beispielaufruf:

```
*> istZugDiagonal 1 4 2 2
False
```

b) Die Zugrichtungen sind ab dieser Teilaufgabe wie folgt kodiert:

0=oben links, 1=oben rechts, 2=unten rechts, 3=unten links

Implementieren und testen Sie in Haskell eine Funktion `bedrohtRichtung`, die als erstes Argument eine x-Koordinate, als zweites eine y-Koordinate und als drittes die Farbe der Figur (`'w'`

oder `'s'`), als viertes die Richtung gemäß obiger Kodierung und als fünftes das Spielfeld erhält und `True` genau dann zurückgibt, falls das Setzen eines Läufers der übergebenen Farbe auf das übergebene Feld einen Läufer in der übergebenen Richtung bedroht. (5 Punkte)

Beispielaufruf:

```
*> bedrohtRichtung 3 1 's' 1 feldA
True
```

c) Implementieren und testen Sie in Haskell eine Funktion `bedroht`, die als erstes Argument eine x-Koordinate, als zweites eine y-Koordinate, als drittes die Farbe der Figur (`'w'` oder `'s'`) und als viertes das Spielfeld erhält und `True` genau dann zurückgibt, falls das Setzen eines Läufers der übergebenen Farbe auf das übergebene Feld einen Läufer bedroht. (5 Punkte)

Beispielaufrufe:

```
*> bedroht 4 2 's' feldA
True
*> bedroht 2 2 's' feldA
False
```

d) Implementieren und testen Sie in Haskell eine Funktion `istZugGueltig`, die als erstes Argument die x-Koordinate der zu ziehenden Figur, als zweites die dazugehörige y-Koordinate, als drittes die x-Koordinate des Zielfelds, als viertes die dazugehörige y-Koordinate und als fünftes das Spielfeld erhält und genau dann `True` zurückliefert, falls der Zug gemäß aller beschriebenen Regeln gültig ist – insbesondere darf niemals ein Läufer bedroht werden. (5 Punkte)

Beispielaufruf:

```
*> istZugGueltig 1 3 2 2 feldA
True
*> istZugGueltig 1 2 2 1 feldA
False
```

e) Implementieren und testen Sie in Haskell eine Funktion `zieheWennGueltig`, die als erstes Argument die x-Koordinate der zu ziehenden Figur, als zweites die dazugehörige y-Koordinate, als drittes die x-Koordinate des Zielfelds, als viertes die dazugehörige y-Koordinate und als fünftes das Spielfeld erhält und das Spielfeld nach Ausführung des Zuges zurückliefert. Falls der Zug nicht gültig ist, soll das Spielfeld unverändert zurückgegeben werden.

Tipp: Das Ergebnis von `zieheWennGueltig` ist eine Funktion mit zwei Argumenten. Hier bietet sich eine anonyme Funktion, also eine Funktion ohne Namen an: `(\a b -> ...)` Unter welchen Bedingungen sind die Werte des gegebenen Spielfelds zu übernehmen? (5 Punkte)

Beispielaufruf:

```
*> zeigeFeld (zieheWennGueltig 1 3 2 2 feldA)
s   w
    w
ss  w
s   w
*> (zieheWennGueltig 1 3 2 2 feldA) 2 2
's'
```

f) Implementieren und testen Sie in Haskell eine Funktion `geloest`, die für ein Spielfeld `True` genau dann zurückliefert, wenn das Rätsel gelöst ist. (5 Punkte)

Beispielaufruf:

```
*> geloest (zieheWennGueltig 4 3 5 2 feldB)
True
```

g) *(Optional)* Lösen Sie das Rätsel und prüfen Sie die Korrektheit Ihrer Lösung unter Verwendung der Funktionen `geloest` und `zieheWennGueltig`. (**0** Punkte)