Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
L. Noschinski, L. Hupel, Dr. J. Blanchette, M. Eberl

WS 2014/15
12.12.2014
Abgabe: 19.12.2014

Einführung in die Informatik 2
10. Übung

Aufgabe G10.1 Dem Zufall auf die Sprünge helfen

Gesucht ist eine Funktion `nRandomR :: (Int, Int) -> Int -> IO [Int]`, die eine Liste von Zufallszahlen generiert. Die Ausgabeliste von `nRandomR (`l`,`h`)` n soll n Elemente enthalten, wobei jedes dieser Elemente $\geq l$ und $\leq h$ sein soll. Außerdem soll die Ausgabeliste keine Elemente doppelt enthalten. Falls in dem vorgegebenen Bereich `(`l`, `h`)` nicht genügend unterschiedliche Zahlen sind, muss Ihre Implementierung nicht terminieren (z.B. für `nRandomR (1,2) 3`).

Verwenden Sie die Funktion `randomRIO :: (Int, Int) -> IO Int` aus `System.Random`, um eine einzige Zufallszahl aus dem vorgegebenen Bereich zu erzeugen. Duplikate vermeiden Sie mit einem Hilfsargument für bereits "gewürfelte" Zahlen. So erkennen Sie Wiederholungen und können einfach neu würfeln.

Aufgabe G10.2 Zahlenraten

Wir wollen folgendes Spiel in Haskell implementieren: Zunächst wählt das Programm eine zufällige Zahl zwischen 0 und 100 (einschließlich) und fragt dann den Benutzer nach einer Zahl. Hat der Benutzer die richtige Zahl erraten, gibt das Programm die Anzahl der Versuche zurück. Andernfalls zeigt es an, ob die gewählte Zahl größer oder kleiner ist und fragt erneut nach einer Eingabe, bis der Benutzer die richtige Zahl erraten hat.

1. Schreiben Sie zunächst eine IO-Aktion `getLineInt :: IO Int`, die eine Zeile von der Tastatur einliest. Repräsentiert diese Zeile eine (positive) Zahl, so soll die Zahl zurückgegeben werden, andernfalls soll eine Fehlermeldung ausgegeben werden und erneut eine Eingabe gelesen werden. Dies wiederholt sich solange, bis es eine gültige Eingabe gibt.

   Verwenden Sie die IO-Aktion `getLine :: IO String` um eine Zeile zu lesen. Mit der Funktion `readMaybe :: String -> Maybe Int` aus dem Modul `Text.Read` können Sie eine Eingabe in eine Zahl umwandeln.

2. Verwenden Sie die vorherige Teilaufgabe um das Spiel als IO-Aktion `guessNum :: IO Int` zu implementieren.

Hinweis zum Testsystem: Auf diesem Aufgabenblatt sollen Sie diverse Funktionen mittels `IO` implementieren. Um diese Interaktionen sinnvoll testen zu können, haben wir auf dem Testsystem eine eigene Version von `IO` definiert und die wichtigsten Standardfunktionen aus `Prelude` und `System.IO` nachgebildet. Einen Unterschied zu Ihrer lokalen Haskell-Installation sollten Sie im Regelfall nicht bemerken. Die Nutzbarkeit von `IO`-spezifischen Funktionen aus anderen Modulen können wir allerdings nicht garantieren. Wie immer gilt: **Ihre Abgabe muss auf dem Testsystem kompilieren.** Melden Sie etwaige Probleme bitte frühzeitig.

Aufgabe H10.1 Wiederholtes IO

Schreiben Sie eine Funktion

```
ioLoop :: (a -> Maybe b) -> IO a -> IO b
```

Der Ausdruck `ioLoop` f a soll die IO-Aktion so lange wiederholen, bis f den Wert akzeptiert (d.h. nicht `Nothing` zurückgibt) und dann das Ergebnis von f zurückgeben.

Benutzen Sie diese Funktion und `readMaybe` aus `Text.Read`, um eine IO-Aktion

```
getInteger :: IO Integer
```

zu implementieren, die solange eine Zeile vom Benutzer einliest, bis diese einen gültigen Integer darstellt und dann den Integer zurückgibt.

Aufgabe H10.2 CSV-Lookup

Der MC hat ein Problem: das Testsystem gibt ihm die Abgaben der Studenten als einen großen Tarball mit einem Unterordner für jeden Upload eines Studenten. Die Namen der Ordner sind die jeweiligen *Submission IDs*. Der MC interessiert sich aber nicht für die IDs, sondern will den *Namen* des jeweiligen Studenten wissen. Die Zuordnung von Submission IDs zu Namen ist in einer CSV-Datei gespeichert, die folgendermaßen aussieht:

```
5120,"eberl@example.com","Eberl","Manuel","Admin","Tutoren/Admins"
5494,"ranseier@example.com","Ranseier","Karl","Scholar","Alumni"
5582,"miersch@example.com","Mierscheid","Jakob Maria","Scholar","Gruppe 42"
```

Die erste Spalte enthält die Submission IDs, die dritte Spalte den Nachnamen und die vierte Spalte den Vornamen. Die restlichen Spalten sind für diese Aufgabe nicht relevant; sie können aber davon ausgehen, dass es immer exakt 6 durch Kommas getrennte Spalten gibt und keiner der Werte in einer Spalte ein Komma enthält.

Ihre Aufgabe ist nun:

1. Schreiben Sie eine IO-Aktion

   ```
   readNameMap :: FilePath -> IO [(Integer, String)]
   ```

die die gegebene CSV-Datei aus dem angegebenen Dateipfad liest und eine Liste von Zuordnungen jeweils einer Submission ID zu jeweils einem Namen im Format `"Vorname Nachname"` zurückgibt.

2. Schreiben Sie eine IO-Aktion

   ```
   submissionQuery :: [(Integer,String)] -> IO ()
   ```

   die eine Liste von Zuordnungen wie die, die Teilaufgabe 1 zurückgibt, als Argument nimmt und

   - die Zeile `Please enter ID` auf der Konsole ausgibt
   - dann eine Zeile mit einer Submission ID von der Konsole liest
   - schließlich den zu dieser ID gehörenden Namen auf der Konsole ausgibt (oder `None`, wenn es keine Submission mit dieser ID gibt)

   Sie können davon ausgehen, dass der Benutzer nur gültige IDs (also ganze Zahlen ≥ 0) eingibt.

Beispiel: Sei `meta.csv` die obige Beispiel-CSV-Datei. Dann sollen sich Ihre IO-Aktionen in GHCi folgendermaßen verhalten:

```
> m <- readNameMap "meta.csv"
> m
[(5120,"Manuel Eberl"),(5494,"Karl Ranseier"),
 (5582,"Jakob Maria Mierscheid")]
> submissionQuery m
Please enter ID
5582
Jakob Maria Mierscheid
> submissionQuery m
Please enter ID
1234
None
```

Aufgabe H10.3 CAPTCHA

Die Übungsleitung möchte das Testsystem auch für externe Teilnehmer öffnen. Dafür muss eine Möglichkeit geschaffen werden, sich als Nutzer zu registrieren. Damit aber nur Menschen und keine Bots an Accounts gelangen dürfen, und sich die Übungsleitung auch nicht an Google binden möchte, ist eine CAPTCHA-Implementation notwendig. Ihre Aufgabe ist es, ein Programm hierfür zu schreiben.

Der Reihe nach müssen folgende Schritte durchgeführt werden:

1. den Nutzer nach einem Nutzernamen fragen

2. den Nutzernamen dem Server melden und CAPTCHA abholen (ASCII-Grafik)
3. das CAPTCHA anzeigen
4. den Nutzer auffordern, das CAPTCHA zu lösen
5. Lösung abfragen und dem Server melden
6. Rückmeldung vom Server abholen und als `Bool` zurückgeben

Alle Nutzereingaben dürfen beliebige Zeichen enthalten (außer Leerzeichen und Zeilenumbruch), dürfen aber nicht leer sein.

Diese Schrittfolge soll von der IO-Aktion `captcha :: IO Bool` durchgeführt werden.

Beispielinteraktionen mit dem Server (Antworten des Servers eingerückt):

```
REGISTER ranseier
  CHALLENGE 8
  ##############
  ##     aa    ##
  ##    a a    ##
  ##   a  a    ##
  ##      a    ##
  ##      a    ##
  ##      a    ##
  ##############
SOLUTION 1
  ACK
```

```
REGISTER ranseier
  CHALLENGE 8
  ##############
  ##     aa    ##
  ##    a a    ##
  ##   a  a    ##
  ##      a    ##
  ##      a    ##
  ##      a    ##
  ##############
SOLUTION 2
  NAK
```

Die genaue Protokollbeschreibung erhalten Sie, wenn Sie dem Server (`vmnipkow3.in.tum.de`, Port 8080) die Nachricht `PROTOCOL` schicken.

Aufgabe H10.4 Man kann es nicht allen recht machen (Wettbewerbsaufgabe)

Wenn Sie nicht am Wettbewerb teilnehmen wollen, bearbeiten Sie bitte diese Aufgabe wie gewohnt und geben Sie sie bis zum 19.12.2014 ab. Falls Sie teilnehmen wollen, lesen Sie bitte weiter, andernfalls können Sie den nächsten Absatz ignorieren.

Diese Wettbewerbsaufgabe ist zweiwöchig. Sie haben bis nächstes Jahr Zeit, einen cleveren Algorithmus zu implementieren. Der MC wird nach einer Woche Bearbeitungszeit bereits eine Zwischenauswertung vornehmen und Ihnen die Ergebnisse mitteilen. Hierfür wird er alle Abgaben berücksichtigen, die bis zur regulären Abgabefrist (19.12.2014) eingegangen sind. Um Hausaufgabenpunkte auf diese Aufgabe zu erhalten, müssen Sie ebenfalls bereits bis zum 19.12.2014 eine funktionstüchtige Implementation abgeben.

Ein bekanntes logisches Problem ist das *SAT*-Problem, oder das „Erfüllbarkeitsproblem der Aussagenlogik“: gegeben eine aussagenlogische Formel (d.h. die Boole'schen Ausdrücke, die Sie

bereits kennen) wird eine Belegung der Variablen gesucht, sodass diese Aussage wahr ist – falls es eine gibt.

Eine weniger bekannte Variante davon ist das *Max-SAT*-Problem: gegeben ist eine Formel in Klauselnormalform (KNF) und gesucht wird eine Belegung, die so viele Klauseln wie möglich erfüllt.

Eine KNF hat immer die Form $C_1 \wedge \ldots \wedge C_n$, wobei jedes C_i eine *Klausel* ist. Eine Klausel hat immer die Form $l_1 \vee \ldots \vee l_k$, wobei jedes l_i ein *Literal* ist. Ein Literal ist dabei entweder eine Variable x (positives Literal) oder die Negation einer Variable, $\neg x$ (negatives Literal). Variablen kodieren wir in der für *SAT*-Instanzen üblichen Form als ganze Zahlen aus $\{1, 2, \ldots\}$; Literale kodieren wir mit ganzen Zahlen aus $\mathbb{Z} \setminus \{0\}$, wobei eine positive Zahl x für das positive Literal x steht und eine negative Zahl $-x$ für das negative Literal $\neg x$. Eine Variablenbelegung kodieren wir als eine Liste der Variablen, die `True` sind.

Beispiel: Wir betrachten die Formel $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2 \vee x_4)$.

- Die zwei Klauseln dieser Formel sind $x_1 \vee x_2$ und $\neg x_1 \vee x_2 \vee x_4$
- Die Kodierung der Klauseln ist `[1, 2]` bzw. `[-1, 2, 4]`
- Die Kodierung der Formel ist also `[[1, 2], [-1, 2, 4]]`
- Eine *Max-SAT*-Lösung wäre die Belegung ($x_1 \mapsto$ True, $x_2 \mapsto$ True), die zwei Klauseln erfüllt.
- Die Kodierung dieser Belegung wäre `[1, 2]`.

Folgende Typsynonyme sind vorgegeben:

```
type Var = Int
type Literal = Int
type Clause = [Literal]
type Assignment = [Var]
```

Ihre Aufgaben sind nun:

1. Schreiben Sie eine Funktion

   ```
   vars :: [Clause] -> [Var]
   ```

   die eine Liste aller in einer Klausel vorkommenden Variablen ohne Duplikate in beliebiger Reihenfolge ausgibt. Bedenken Sie, dass Variablen immer ≥ 1 sind.

2. Schreiben Sie eine Funktion

   ```
   isSatisfied :: Clause -> Assignment -> Bool
   ```

   die ausgibt, ob die gegebene Variablenbelegung die gegebene Klausel erfüllt.

3. Schreiben Sie eine Funktion

   ```
   maxSat :: [Clause] -> (Assignment, Int)
   ```

   die zu der gegebenen Formel in KNF eine Variablenbelegung zurückgibt, die so viele Klauseln wie möglich erfüllt sowie die Anzahl der erfüllten Klauseln.

Beispiele:

```
vars [[1,-4,3],[2,1,3]] == [1,4,3,2]

isSatisfied []       _      == False -- leere Klausel, also False
isSatisfied [1,2,3] [2,4] == True
isSatisfied [1,2,3] [4]   == False

maxSat [[ 1, 2, 3], [-1,-2,-3], [ 1,-2,-3]] == ([1, 3], 3)
    -- oder ([1,2], 3) oder ([1], 3) ...

maxSat [[ 1, 2, 3], [-1,-2,-3], [ 1,-2,-3], [-1, 2,-3],
        [ 1, 2,-3], [-1,-2, 3], [ 1,-2, 3], [-1, 2, 3]]
    == ([1,2,3], 7)
```

Hinweise: Die Funktionen `subsequences` und `maximumBy` aus `Data.List` sowie `comparing` aus `Data.Ord` könnten hilfreich sein.

Für die Hausaufgabe ist ein *Brute-Force*-Ansatz, der alle Belegungen durchprobiert, völlig ausreichend (solange die sehr großzügigen Timeout-Grenzen auf dem Abgabeserver nicht überschritten werden); für den Wettbewerb zählt als Hauptkriterium die Effizienz auf den vom MC willkürlich ausgewählten Beispielen. Bei ähnlicher Effizienz wird der MC auf ähnlich willkürliche Art und Weise die Eleganz und Formatierung des Codes beurteilen.

Wichtig: Wenn Sie diese Aufgabe als Wettbewerbsaufgabe abgeben, stimmen Sie zu, dass Ihr Name ggf. auf der Ergebnisliste auf unserer Internetseite veröffentlicht wird. Sie können diese Einwilligung jederzeit widerrufen, indem Sie eine Email an `fp@fp.in.tum.de` schicken. Wenn Sie nicht am Wettbewerb teilnehmen, sondern die Aufgabe allein im Rahmen der Hausaufgabe abgeben möchten, lassen Sie bitte die {`-WETT-`}...{`-TTEW-`} Kommentare weg. Bei der Bewertung Ihrer Hausaufgabe entsteht Ihnen hierdurch kein Nachteil.