Dr. David Sabel
Institut für Informatik

Fachbereich Informatik und Mathematik
Johann Wolfgang Goethe-Universität Frankfurt am Main

Praktikum BKSPP

Sommersemester 2012

Aufgabenblatt Nr. 1

Abgabe: Mittwoch, 25. April 2012

1 Einrichten der Arbeitsumgebung

Zur Bearbeitung der Praktikumsaufgaben, ist es notwendig, dass Sie Haskell-Programme (samt einiger Programmbibliotheken) interpretieren und kompilieren können. Außerdem ist der Umgang mit dem CVS-System notwendig.

Aufgabe 1 Diese Aufgabe *sollte jeder für sich* **alleine** *bearbeiten.*

Wenn Sie Ihren eigenen Rechner verwenden, so installieren Sie die Haskell Platform (`http://hackage.haskell.org/platform/`*). Prüfen Sie, ob die Programme* `ghc`, `ghci`, `haddock` *und* `happy` *anschließend verfügbar sind (wenn Sie auf einem Rechner der RBI arbeiten, sollte das Prüfen ausreichen). Dokumentieren Sie kurz, welches Betriebssystem Sie verwenden.*

Suchen Sie sich einen Editor aus und erstellen Sie das Haskell-Programm:

```
module Main where
 main = putStrLn "Hallo Welt"
```

in einer Datei names `Main.hs`*. Kompilieren Sie anschließend das Programm mit dem GHC und führen Sie es aus.*

Dokumentieren Sie, welchen Editor Sie zur Erstellung von Haskell-Quelltext verwenden und mit welchem Kommando das obige Programm kompiliert wurde.

Ändern Sie obiges Programm derart ab, dass zuerst der Name *des Benutzers von der Standardeingabe eingelesen wird, und anschließend „*`Hallo` Name*“ auf der Standardausgabe ausgegeben wird.*

Legen Sie in Ihrem CVS-Verzeichnis im Verzeichnis `bkspp/blatt1/aufgabe1/` *ein Unterverzeichnis mit Ihrem Nachnamen an. Legen Sie in dieses Verzeichnis die zu dieser Aufgabe zugehörigen Quelltexte und Dokumentation und fügen Sie es der Versionsverwaltung hinzu. Beachten Sie, dass das Verzeichnis und die Dateien mit Ihrem Usernamen hinzugefügt und eingecheckt werden (und nicht mit dem Usernamen eines Ihrer Gruppenmitglieder!). Dokumenieren Sie, welches Werkzeug Sie zum Zugriff auf das CVS-Repository verwendet haben (z.B. Eclipse, Shell-Kommandos, etc.).*

2 Umrechnung zwischen Stellenwertsystemen

Ein Stellenwertsystem dient zur Darstellung von Zahlen. Ein *Alphabet* $\mathcal{A}$ ist eine Folge $(a_1, \ldots, a_n)$ von paarweise verschiedenen Symbolen. Eine *Zahl* im Stellenwertsystem zum Alphabet $\mathcal{A}$ ist ein Wort über dem Alphabet $\mathcal{A}$. Jedem Symbol $a_i \in \mathcal{A}$ ist eine Dezimalzahl $p(a_i)$ aus dem Bereich $[0, \ldots, |\mathcal{A}|-1]$ entsprechend seiner Position (abzüglich 1) in der Folge $\mathcal{A}$ zugeordnet, wobei $|\mathcal{A}|$ die Länge des Alphabets bezeichnet. Eine Zahl $c_n \ldots c_1 c_0$ im Stellenwertsystem zum Alphabet $\mathcal{A}$ hat den dezimalen Wert $\sum_{i=0}^{n} p(c_i) \cdot b^i$ wobei $b = |\mathcal{A}|$. Um eine eindeutige Darstellung zu erhalten, fordern wir zusätzlich, dass Zahlen mit mehr als einer Stelle im Stellenwertsystem zum Alphabet $\mathcal{A}$ nie mit dem Symbol a_1 beginnen, wobei a_1 das erste Symbol des Alphabets $\mathcal{A}$ sei (für welches $p(a_1) = 0$ gilt).

Prominente Beispiele für Stellenwertsysteme sind:

- Das Dezimalsystem ist das Stellenwertsystem über dem Alphabet $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)$.
- Das Binärsystem ist das Stellenwertsystem über dem Alphabet $(0, 1)$. Hierbei ist $p(0) = 0$ und $p(1) = 1$. Z.B. lässt sich der dezimale Wert der binären Zahl 11001001 berechnen als $1{\cdot}2^0{+}0{\cdot}2^1{+}0{\cdot}2^2{+}1{\cdot}2^3{+}0{\cdot}2^4{+}0{\cdot}2^5{+}1{\cdot}2^6{+}1{\cdot}2^7 = 1{+}8{+}64{+}128 = 201$. Durch obige Konvention sind Binärzahlen mit führenden 0en verboten.
- Das Hexadezimalsystem ist das Stellenwertsystem über dem Alphabet $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)$. Hierbei ist $p(A) = 10, p(B) = 11, p(C) = 12, p(D) = 13, p(E) = 14, p(F) = 15$ und $p(x) = x$ sonst. Z.B. lässt sich der dezimale Wert der hexadezimalen Zahl $CDEF$ berechnen als $15 \cdot 16^0 + 14 \cdot 16^1 + 13 \cdot 16^2 + 12 \cdot 16^3 = 15 + 224 + 3328 + 49152 = 52719$.

Ein Alphabet kann in Haskell als Liste dargestellt werden. Um zu verdeutlichen, dass es sich um Alphabete handelt, führen wir hierfür ein Typsynonym ein:

```
type Alphabet a = [a]
```

Nicht-negative Zahlen aus einem Stellenwertsystem können ebenfalls als Listen dargestellt werden. Auch hierfür führen wir ein Typsynonym ein:

```
type Number a = [a]
```

Aufgabe 2 *Implementieren Sie in Haskell ein Modul* `Stellenwertsystem`*, dass neben den Typen* `Number` *und* `Alphabet` *ausschließlich die folgenden Funktionen exportiert:*

- `integer2alpha :: Alphabet a -> Integer -> Number a`*, die ein Alphabet in Form einer Liste (z.B.* `[0,1]`*) und eine nicht-negative Ganzzahl (im Dezimalsystem) vom Typ* `Integer` *erwartet und diese Zahl im Stellenwertsystem passend zum Alphabet darstellt.*

- `alpha2Integer :: Alphabet a -> Number a -> Integer`, *die ein Alphabet $\mathcal{A}$ in Form einer Liste und eine Zahl im Stellenwertsystem zum Alphabet $\mathcal{A}$ erwartet und diese in eine nicht-negative Ganzzahl vom Typ* `Integer` *(im Dezimalsystem) umrechnet.*
- `alpha2alpha' :: Alphabet a -> Alphabet b -> Number a -> Number b`, *die zwei Alphabete $\mathcal{A}$ und $\mathcal{A}'$ sowie eine Zahl im Stellenwertsystem zum Alphabet $\mathcal{A}$ erhält, und die Darstellung der Zahl im Stellenwertsystem zum Alphabet $\mathcal{A}'$ berechnet.*

Hinweise:

- *Möglicherweise können Sie die Funktionen nicht mit den angegebenen Typen implementieren, da noch Bedingungen an Typklasseninstanzen benötigt werden. Diese dürfen Sie hinzufügen. Als Vorgehen sei dabei empfohlen, den Typ der Funktionen zunächst nicht selbst anzugeben, sondern vom Interpreter berechnen zu lassen und anschließend den berechneten Typen (oder eine Instanz davon) hinzuzufügen.*
- *Sie benötigen eventuell Funktionen, die Zahlen vom Typ* `Int` *in Zahlen vom Typ* `Integer` *und umgekehrt konvertieren. Beachten Sie, dass dies die folgenden Klassen-Funktionen der Typklassen* `Num a` *und* `Integral a` *bereits leisten:*
  - `toInteger :: a -> Integer`
  - `fromInteger :: Integer -> a`
- *Erzeugen Sie eine aussagekräftige Fehlermeldung für die Fälle:*
  - *Der übergebene* `Integer`*-Wert ist negativ.*
  - *Die übergebene Zahl enthält Symbole, die nicht zum Alphabet passen.*
  - *Die übergebene Zahl ist eine leere Liste.*

  Hierfür können Sie die `error`*-Funktion verwenden.*

3 Einfache Textkompression

Im Folgenden wird eine einfache Zeichenbasierte Kompressionsmethode beschrieben, die die Umwandlung zwischen verschiedenen Stellenwertsystemen benutzt. Der Eingabestring wird als Zahl bezüglich des Stellenwertsystems aller im String vorkommenden Zeichen interpretiert. Dieser String wird nun konvertiert in die Darstellung bezüglich des Stellenwertsystems aller ASCII-Zeichen. Da das Alphabet in diesem Schritt hoffentlich vergrößert wird, ist zu hoffen, dass der Ausgabestring kürzer als der Eingabestring ist. Damit der komprimierte String auch wieder dekodiert werden kann, muss zusätzlich das Alphabet mit ausgegeben werden. Wir kürzen dies durch ein neues Typsynonym ab:

```
type Code = (Alphabet Char,String)
```

Aufgabe 3 *Implementieren Sie ein Modul* `SimpleCompress`*, welches Funktionen*

- `simpleCompress :: String -> Code` *und*
- `simpleDecompress :: Code -> String`

bereitstellt, die nach oben beschriebener Methode einen String komprimieren bzw. dekomprimieren.

Implementieren Sie anschließend ein Modul `Main`*, welches nach Kompilation ein Konsolen-Programm ergibt, dass Dateien mit der oben beschriebenen Methode komprimieren und dekomprimieren kann. Sei* `sc` *das kompilierte Programm. Es sollte zumindest die folgenden Aufrufe von der Konsole ermöglichen:*

- `sc -c` *"Eingabedatei" "Ausgabedatei": Der Inhalt der Datei namens "Eingabedatei" wird gelesen, der komprimierte Inhalt wird in die Datei namens "Ausgabedatei" geschrieben.*
- `sc -d` *"Eingabedatei": Der Inhalt der Datei namens "Eingabedatei" wird dekomprimiert und auf der Standardausgabe ausgedruckt.*

Hinweise:

- *Beachten Sie, dass das Alphabet zum Eingabestring nie so gewählt werden sollte, dass der Eingabestring mit einem Symbol beginnt, welches der 0 entspricht.*
- *Beachten Sie die Sonderfälle, dass der zu komprimierende String leer ist, oder nur aus gleichen Zeichen besteht.*
- *Für die Umrechnung von ASCII-Zeichen in Zahlen im Bereich* $[0, ..., 255]$ *und umgekehrt können Sie die Funktionen* `ord` *und* `chr` *aus der Bibliothek* `Data.Char` *verwenden.*
- *Zur Implementierung des Konsolenprogramms empfiehlt es sich die Funktion* `getArgs` *aus der Bibliothek* `System.Environment` *zu verwenden.*
- *Zum Abspeichern des Codes müssen Sie dessen beide Teile (Alphabet und kodiertes Wort) durch ein Trennsymbol o.ä. trennen. Dies muss so gewählt werden, dass das eindeutige Dekodieren möglich ist!*