5. Aufgabenblatt zu Funktionale Programmierung vom Mi, 16.11.2016.
Fällig: Mi, 23.11.2016 (jeweils 15:00 Uhr)

Themen: *Typklassen und Funktionen auf algebraischen Datentypen*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe5.hs` ablegen. Sie sollen für die Lösung dieses Aufgabenblatts also wieder ein "gewöhnliches" Haskell-Skript schreiben. Versehen Sie wieder wie auf den bisherigen Aufgabenblättern alle Funktionen, die Sie zur Lösung brauchen, mit ihren Typdeklarationen und kommentieren Sie Ihre Programme aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

Zur Frist der Zweitabgabe: Siehe allgemeine Hinweise auf Aufgabenblatt 1.

1. Wir betrachten noch einmal die auf Aufgabenblatt 3 eingeführten Typen zur Darstellung ganzer Zahlen zur Basis 3, allerdings ohne `deriving`-Klauseln.

```
data Digit      = Zero | One | Two
type Digits     = [Digit]
data Sign       = Pos | Neg -- Pos fuer Positive, Neg fuer Negative
newtype Numeral = Num (Sign,Digits)
```

   Alle weiteren Festlegungen über Werte dieser Datentypen zu Kanonzität, etc., von Aufgabenblatt 3 werden für diese Aufgabe übernommen.

   Im Haskell-Prelude sind u.a. die Typklassen `Eq`, `Ord`, `Show` und `Num` zusammen mit den Signaturen der Rechenvorschriften definiert, die für jede Instanz dieser Typklassen implementiert sein müssen. Für einige dieser Rechenvorschriften sind dabei bereits Proto-Implementierungen in den Klassendefinitionen angegeben. Bei Instanzbildungen reicht es deshalb oft aus, für eine Teilmenge der in einer Typklasse vorgesehenen Rechenvorschriften typspezifische Implementierungen bei der Instanzbildung anzugeben, wenn die sich aus den Proto-Implementierungen automatisch ergebenden Vollimplementierungen das gewünschte Verhalten aufweisen. Machen Sie davon bei den folgenden Teilaufgaben Gebrauch und implementieren Sie lediglich die Minimalmenge von Rechenvorschriften und übernehmen überall sonst, wo möglich, die Proto-Implementierung unverändert, d.h. überschreiben Sie diese nicht.

   Machen Sie die Datentypen `Digit`, `Sign` und `Numeral` explizit, d.h. ohne Rückgriff auf eine *deriving*-Klausel, zu je einer Instanz der Typklassen

   a) `Eq` (Minimale Vervollständigung: Implementierung von (==) oder (/=): implementieren Sie für diese Aufgabe den Relator (==).)

   b) `Show` (Minimale Vervollständigung: Implementierung von `show` oder `showsPrec`: implementieren Sie für diese Aufgabe die Rechenvorschrift `show`.)

   Durch `show` sollen die Werte des Typs `Digit` durch die (Ziffern-) Zeichen '0', '1' und '2' dargestellt werden, Werte des Typs `Sign` als Zeichen '+' und '-' und Werte des Typs `Numeral` als Zeichenreihen "$vzd_1...d_k$" mit $vz \in \{'+','-'\}$ und $d_i \in \{'0','1','2'\}$, $i \in \{1,...,k\}$. Dabei soll weiters gelten, dass Numeralwerte in kanonischer Darstellung ausgegeben werden, d.h., $d_1 = '0' \Rightarrow k=1$ (keine führenden Nullen).

   Machen Sie die Datentypen `Digit` und `Numeral` explizit, also wieder ohne Rückgriff auf eine *deriving*-Klausel, zu je einer Instanz der Typklassen

   c) `Ord` (Minimale Vervollständigung: Implementierung von (<=) oder `compare`: implementieren Sie für diese Aufgabe den Relator `compare`.)

   Machen Sie schließlich den Datentyp `Numeral`, ebenfalls wieder ohne Rückgriff auf eine *deriving*-Klausel, zu einer Instanz der Typklasse

   d) `Num` (Minimale Vervollständigung: Implementierung von (`negate` oder $(-)$) und aller anderen in der Klasse vorgesehenen Rechenvorschriften: implementieren Sie für diese Aufgabe alle in der Klasse vorgesehenen Rechenvorschriften, also auch beide Rechenvorschriften `negate` und $(-)$.)

Die Bedeutung aller vorgesehenen Rechenvorschriften soll denen des Typs `Integer` entsprechen.

Ohne Abgabe: Vergleichen Sie die Ausgabe Ihrer `show`-Funktion mit derjenigen der automatisch mithilfe der *deriving*-Klausel für `Numeral` abgeleiteten `show`-Funktion. Welcher "Darstellungsidee" von Werten des Typs `Nat` folgt die automatisch abgeleitete `show`-Funktion, welcher die von Ihnen selbst implementierte?

Hinweis: Keine früheren Lösungen als Module importieren!

Wenn Sie zur Lösung einzelne Funktionen früherer Lösungen wiederverwenden möchten, so kopieren Sie diese unbedingt explizit in Ihre neue Programmdatei ein. Importieren schlägt im Rahmen der automatischen Programmauswertung fehl. Es wird nicht nachgebildet. Deshalb: Wiederverwendung ja, aber durch kopieren, nicht durch importieren!

Haskell Live

An einem der kommenden *Haskell Live*-Termine, der nächste ist am Freitag, den 18.11.2016, werden wir uns u.a. mit der Aufgabe *World of Perfect Towers* beschäftigen.

World of Perfect Towers

In diesem Spiel konstruieren wir Welten perfekter Türme. Dazu haben wir n Stäbe, die senkrecht auf einer Bodenplatte befestigt sind und auf die mit einer entsprechenden Bohrung versehene Kugeln gesteckt werden können. Diese Kugeln sind ebenso wie die Stäbe beginnend mit 1 fortlaufend nummeriert.

Die auf einen Stab gesteckten Kugeln bilden einen Turm. Dabei liegt die zuerst aufgesteckte Kugel ganz unten im Turm, die zu zweit aufgesteckte Kugel auf der zuerst aufgesteckten, usw., und die zuletzt aufgesteckte Kugel ganz oben im Turm. Ein solcher Turm heißt *perfekt*, wenn die Summe der Nummern zweier unmittelbar übereinanderliegender Kugeln eine Zweierpotenz ist. Eine Menge von n perfekten Türmen heißt *n-perfekte Welt*.

In diesem Spiel geht es nun darum, n-perfekte Welten mit maximaler Kugelzahl zu konstruieren. Dazu werden die Kugeln in aufsteigender Nummerierung, wobei mit der mit 1 nummerierten Kugel begonnen wird, so auf die n Stäbe gesteckt, dass die Kugeln auf jedem Stab einen perfekten Turm bilden und die Summe der Kugeln aller Türme maximal ist.

Schreiben Sie in Haskell oder einer anderen Programmiersprache ihrer Wahl eine Funktion, die zu einer vorgegebenen Zahl n von Stäben die Maximalzahl von Kugeln einer n-perfekten Welt bestimmt und die Türme dieser n-perfekten Welt in Form einer Liste von Listen ausgibt, wobei jede Liste von links nach rechts die Kugeln des zugehörigen Turms in aufsteigender Reihenfolge angibt.