6. Übungsaufgabe zu
Fortgeschrittene funktionale Programmierung
Thema: Ein- und ausgeben, analysieren und interpretieren
ausgegeben: Mi, 27.04.2016, fällig: Mi, 04.05.2016

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der im folgenden angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `AufgabeFFP6.hs` in Ihrem Gruppenverzeichnis ablegen, wie gewohnt auf oberstem Niveau. Kommentieren Sie Ihre Programme aussagekräftig und benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

Wir betrachten den Datentyp `Token`:

```
data Token = Wrong | Mult | Add | Value Integer deriving (Eq,Show)
```

- Schreiben Sie eine Haskell-Rechenvorschrift `inOut :: (String -> String) -> IO Integer`, die wiederholt Zeilen von `stdin` liest, auf die Zeichenketten (ohne New-Lines "`\n`") die übergebene Funktion anwendet und die Ergebnisse als getrennte Zeilen in `stdout` schreibt. Bei der Eingabezeile `"end"` wird das wiederholte Einlesen und Verarbeiten abgebrochen. Das Ergebnis von `inOut` ist die Anzahl ausgegebener Zeilen.
- Schreiben Sie eine Haskell-Rechenvorschrift `tokenize :: String -> [Token]`, die eine Zeichenreihe in eine Liste von Tokens umwandelt. Jedes Zeichen `'*'` soll in `Mult` und jedes Zeichen `'+'` in `Add` umgewandelt werden. Jede ununterbrochene Folge von Ziffern (möglicherweise mit direkt (d.h. ohne trennende(s) Leerzeichen) vorangestelltem Zeichen `'-'`) soll in `Value n` umgewandelt werden, wobei `n` die Zahl ist, die durch die Zeichenfolge dargestellt wird. Leerzeichen werden in kein Token umgewandelt. Alle anderen Zeichen werden in `Wrong` umgewandelt.
- Schreiben Sie eine Haskell-Rechenvorschrift `toString :: [Integer] -> String`, die die Argumentliste umdreht und in eine Zeichenreihe mit durch (genau ein) Leerzeichen getrennten Zahl(darstellung)en umwandelt. Zum Beispiel soll der Aufruf von `toString` auf die Liste `[1,-2,35]` das Resultat `"35 -2 1"` liefern. Angewandt auf die leere Liste soll die Rechenvorschrift `toString` die Zeichenreihe `"Error"` als Resultat liefern.
- Schreiben Sie eine Haskell-Rechenvorschrift `interpret :: [Token] -> [Integer]`, die Tokenlisten mithilfe eines Stacks auswertet und den Stack (in Form einer Liste, "top of stack" vorne) zurückgibt. Anfangs ist der Stack leer. Für `Value`-Token wird die Zahl auf den Stack gelegt. Für `Mult`- oder `Add`-Token werden die beiden obersten Zahlen vom Stack genommen, multipliziert bzw. addiert und das Ergebnis auf den Stack gelegt. Wenn der Stack zu wenige Elemente enthält oder ein `Wrong`-Token als Argument auftritt, gibt `interpret` die leere Liste zurück.

- Schreiben Sie eine Haskell-Rechenvorschrift `main :: IO()`, die die Funktion `inOut` so aufruft, dass jede gelesene Zeile mithilfe der Funktionen `tokenize`, `interpret` und `toString` in eine Ausgabezeile umgewandelt wird. Zum Schluss soll "n`Zeilen`" ausgegeben werden, wobei n das Ergebnis der Funktion `inOut` ist.