4. Aufgabenblatt zu Funktionale Programmierung vom 03.11.2004, fällig: 10.11.2004

Themen: *Funktionen auf ganzen Zahlen, Zeichenreihen und Listen*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe4.hs` ablegen. Sie sollen also wieder ein "gewöhnliches" Haskell-Script schreiben. Versehen Sie 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. Im einzelnen sollen Sie folgende Problemstellungen bearbeiten:

1. Natürliche Zahlen in römischer Darstellung oder kürzer römische Zahlen sind über den Symbolen `M`, `D`, `C`, `L`, `X`, `V` und `I` aufgebaut, die für die Zahlwerte 1000, 500, 100, 50, 10, 5 und 1 stehen. Um weitere Zahlen darzustellen, können diese Symbole aneinandergereiht werden. Grundsätzlich gilt dabei, dass nebeneinander stehende Symbole aufaddiert werden (Beispiel: `LXXVII` für 77, `MDLXIII` für 1563). Insbesondere um Eindeutigkeit der Zahldarstellung zu erreichen, treffen wir zusätzlich folgende Vereinbarungen: Es dürfen nie mehr als 3 gleiche Symbole nebeneinanderstehen; steht ein Symbol mit kleinerem Wert vor einem Symbol mit größerem Wert, so wird der Wert des kleineren Symbols von dem des größeren Symbols subtrahiert (Beispiel: `IX` für 9); eine Subtraktionsdarstellung wird nur verwendet, wenn sich der Zahlwert (unter den übrigen Vereinbarungen) nicht additiv darstellen lässt (Beispiel: `IXX` für 19, nicht jedoch `XVIIII`. Andererseits nicht `IIXX` für 18, denn 18 ist additiv durch `XVIII` darstellbar). Kleinere Symbole stehen nicht vor einem größeren Symbol `S`, wenn es schon ein noch weiter links stehendes Vorkommen von `S` gegeben hat (Beispiel: `IMM` für 1999, nicht aber `MIM`); von zwei additiven Darstellungen ist nur die kürzere gültig (Beispiel: `MM` für 2000, nicht aber `MDD`). Aus diesen Nebenbedingungen ergibt sich insbesondere ein Wert für eine größte in römischer Darstellung darstellbare Zahl. Schreiben Sie eine Haskell-Rechenvorschrift `convert2Roman` mit Signatur `convert2Roman :: Int -> String`, die ganze Zahlen, die eine im obigen Sinn gültige Darstellung in römischer Notation besitzen, in ihre römische Darstellung überführt; ansonsten liefert die Funktion `convert2Roman` die Zeichenreihe `"Ungueltige Eingabe"` zurück.

2. Schreiben Sie eine Haskell-Rechenvorschrift `decodeRoman` mit Signa-

tur `decodeRoman :: String -> Int`, die eine als Argument übergebene Zeichenreihe daraufhin überprüft, ob sie im Sinne der vorigen Aufgabe eine gültige Codierung einer natürlichen Zahl in römischer Darstellung ist. In diesem Fall liefert die Funktion die entsprechende Zahl zurück, ansonsten -1.

3. Vertraulichkeit von Nachrichten versucht man schon seit der Antike durch geeignete Verschlüsselung der Nachrichten sicherzustellen. Julius Caesar verwendete dafür eine vergleichsweise einfache Methode, die heute seinen Namen trägt. Die Idee der Caesar-Chiffrierung liegt in einer zyklischen Alphabetverschiebung, konkret etwa jeden Buchstaben in einer Nachricht durch den im Alphabet 3 Positionen später stehenden Buchstaben zu ersetzen. Nach diesem Schema wird z.B. `Caesar` durch `Fdhvdu` und `Zar` durch `Cdu` codiert. Schreiben Sie zwei Haskell-Rechenvorschriften `codeCaesar` und `decodeCaesar` mit Funktionalität `String -> Int -> String` zur Codierung bzw. Decodierung von Zeichenreihen nach dem Schema der Caesar-Chiffrierung. Dabei bezeichnet das erste Argument die zu codierende bzw. zu decodierende Zeichenreihe, das zweite Argument die zu beachtende zyklische Verschiebung über dem Alphabet {a,b,c,d,...,z, A,B,C,D,...,Z,1,2, ...,9,0}. Eventuell im Argument vorkommende Zeichen, die nicht in diesem Alphabet enthalten sind, z.B. Interpunktionszeichen oder Umlaute, bleiben unter der Caesar-Chiffrierung invariant. Die Verschiebung bei Ziffern erfolgt in gleicher Weise wie bei Buchstaben. Bei einer Verschiebung von 3 wird also 1 durch 4, 5 durch 8 und 9 durch 2 codiert. Beachten Sie, dass Verschiebungen sowohl positiv wie negativ sein können. Für negative Verschiebungen gelten die Ausführungen entsprechend. So wird für eine Verschiebung von -2 D durch B, a durch y und 2 durch 0 codiert.

4. Schreiben Sie eine Haskell-Rechenvorschrift `concatSpecial` mit Funktionalität `[a] -> [a] -> Int -> [a]`. Dabei soll `concatSpecial` die als zweites Argument übergebene Liste an der durch das dritte Argument bezeichneten Stelle in der ersten Liste einfügen. Die geänderte Liste soll dann als Resultat abgeliefert werden. Dabei heißt Einfügen an der i-ten Stelle Einfügen unmittelbar vor dem i-ten Element der ersten Liste. Ist i kleiner als 1 oder größer als die Anzahl der Elemente in der ersten Liste, soll die zweite Liste vor dem ersten bzw. nach dem letzten Element in der ersten Liste eingefügt werden. Beispiele: `concatSpecial [1,3,2] [4,5,6] 3` $\leadsto$ `[1,3,4,5,6,2]`, `concatSpecial ['a','b'] ['t','c'] 42` $\leadsto$ `['a','b','t','c']`.