5. Übungsaufgabe zu
Fortgeschrittene funktionale Programmierung
Thema: Erschöpfende Suche, Teile und Herrsche
ausgegeben: Mo, 30.04.2012, fällig: Mi, 09.05.2012

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der im folgenden angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `AufgabeFFP5.hs` in Ihrem Gruppenverzeichnis ablegen, wie gewohnt auf oberstem Niveau. Kommentieren Sie Ihre Programme aussagekräftig und benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

Seien *low* und *high* ganze Zahlen mit $low \leq high$ und sei a ein eindimensionales Feld mit kleinstem Index *low* und größtem Index *high*. Seien weiter i und j ganze Zahlen mit $low \leq i \leq j \leq high$.

Dann heißt der Ausschnitt von a mit kleinstem Index i und größtem Index j, in Zeichen $a \downarrow [i, j]$, ein *Abschnitt* von a.

Für das Weitere nehmen wir an, dass die Elemente von a ebenfalls ganze Zahlen sind und dass b ein Abschnitt von a ist. Die *Abschnittsumme* von b ist dann die Summe der Elementwerte von b.

Beispiel:

```
a = array (1,9) [(1,3),(2,(-5)),(3,0),(4,9),(5,2),(6,(-1)),(7,2),(8,(-5)),(9,1)]
```

Dann gilt: Die Abschnittsumme von

$a \downarrow [2, 5]$ ist: $(-5) + 0 + 9 + 2 = 6$

$a \downarrow [7, 9]$ ist: $2 + (-5) + 1 = -2$

$a \downarrow [4, 7]$ ist: $9 + 2 + (-1) + 2 = 12$

$a \downarrow [3, 8]$ ist: $0 + 9 + 2 + (-1) + 2 + (-5) = 7$

- Schreiben Sie eine Haskell-Rechenvorschrift `mas :: Array Int Int -> Int`, die angewendet auf ein Feld den maximalen Wert der Abschnittsummen dieses Feldes berechnet.

  Beispiel: Für das Feld a liefern die Abschnitte $a \downarrow [3, 7]$ und $a \downarrow [4, 7]$ die größte Abschnittsumme mit Wert 12. Es gilt:

  ```
  mas a ->> 12
  ```

- Schreiben Sie eine Haskell-Rechenvorschrift `amas :: Array Int Int -> [(Int,Int)]`, die angewendet auf ein Feld die Liste derjenigen Abschnitte berechnet, dargestellt jeweils durch ihren kleinsten und größten Index, deren Abschnittsumme maximal für das Argumentfeld ist.

Die Ergebnisliste soll dabei so geordnet sein, dass (i, j) genau dann weiter links in der Ergebnisliste stehen soll als (k, l), wenn gilt:

$$i < k \quad \text{oder} \quad i = k \wedge j < k$$

Beispiel: Für die Felder a und b liefert die Funktion `amas` die Resultate:

```
amas a ->> [(3,7),(4,7)]

b = array (1,9) [(1,3),(2,(-1)),(3,(-2)),(4,9),(5,2),(6,(-1)),
                 (7,2),(8,0),(9,(-1))]
amas b ->> [(1,7),(1,8),(4,7),(4,8)]
```

- Schreiben Sie eine Haskell-Rechenvorschrift `lmas :: Array Int Int -> (Int,Int)`, die den längsten Abschnitt mit maximaler Abschnittsumme berechnet. Gibt es mehrere, so liefert die Funktion `lmas` den Abschnitt mit kleinstem Anfangsindex.

  Beispiel:

  ```
  lmas a ->> (3,7)
  lmas b ->> (1,8)

  c = array (1,5) [(1,2),(2,3),(3,(-10)),(4,1),(5,4)]
  lmas c ->> [(1,2)]
  ```

  Hinweis: Die Funktionen `mas`, `amas` und `lmas` werden ausschließlich mit vollständig definierten Feldern aufgerufen; es gibt in Argumenten keine Indizes mit "undefiniertem" Elementwert.

- Sei a ein eindimensionales Feld und wf eine Wahrheitswertfunktion. Schreiben Sie eine Haskell-Rechenvorschrift `minIndex :: (Ix a, Show a) => Array a b -> (b -> Bool) -> a`, die angewendet auf ein Feld und eine Wahrheitswertfunktion den kleinsten Index bestimmt, für dessen Elementwert die Wahrheitswertfunktion erfüllt ist, also den Wert `True` liefert. Erfüllt kein Element die Wahrheitswertfunktion, bricht die Berechnung mittels eines Aufrufs der Funktion `error` ab.

  Implementieren Sie die Funktion `minIndex` mithilfe des "Teile und Herrsche"-Prinzips. Stützen Sie dazu die Implementierung von `minIndex` auf das Funktional `divideAndConquer` aus Kapitel 3.1 der Vorlesung ab. Geben Sie dazu Implementierungen der Funktionen `mi-indiv`, `mi-solve`, `mi-divide` und `mi-combine` an und rufen Sie damit das Funktional `divideAndConquer` entsprechend auf.

  Wandeln Sie dabei das als Feld gegebene Argument in eine Liste um, damit Sie `divideAndConquer` unverändert anwenden können.

Beispiele:

```
minIndex a (>5)   ->> 4
minIndex a (<0)   ->> 2
minIndex a (even) ->> 3
minIndex b (odd)  ->> 1
minIndex b (>100) ->> error "No matching index"

data Week = Mon | Tue | Wed | Thu | Fri | Sat | Sun deriving (Eq,Ord,Ix,Show)

d :: Array Week String
d = array (Tue,Sat) [(Wed,"work"),(Thu,"study"),(Tue,"study"),
                     (Fri,"chill"),(Sat,"relax")]

minIndex d (=="relax") ->> Sat
minIndex d (=="work")  ->> Wed
minIndex d (=="chill") ->> Fri
minIndex d (/="chill") ->> Tue
minIndex d (=="swim")  ->> error "No matching index"
```

Hinweis: Die Funktion `minIndex` wird ausschließlich mit vollständig definierten Feldern aufgerufen; es gibt in Argumenten keine Indizes mit "undefiniertem" Elementwert.