Fakultät IV – Elektrotechnik und Informatik
MPGI 1 WS 2013/14
Pepper / Höger / Reicherdt / Zuber / Tutoren

Aufgabenblatt 8

letzte Aktualisierung: 04. 12, 17:42

Ausgabe: 04.12.2013
Abgabe: 12./13.12.2013

Thema: *Pattern Matching, Listenfunktionale*

Achtung: In diesem Aufgabenblatt erwarten wir von euch ein **Testmodul**, das Testfunktionen ähnlich der Hausaufgabe 3.2 des 5. Aufgabenblattes enthält. Es soll **alle** eure Module der Hausaufgabe importieren und deren Funktionen mit **geeigneten** Testwerten füttern und **nachvollziehbare** Ausgaben liefern.

1. Aufgabe: Datentypen und Sequenzen: CD-Liste

1.1. (Tut) Datenstruktur

Gegeben ist der folgende Datentyp:

```
TYPE cd == cd(name:denotation, artist:denotation, year:nat)
```

Handelt es sich um einen Aufzählungs-, Produkt- oder Summentyp?

1.2. (Tut) Suchen in einer Liste I

Gegeben ist eine Liste von CDs `seq[cd]`.

Schreibt zu der gegebenen Deklaration eine Funktionsdefinition, die alle CDs aus einem bestimmten Jahr heraussucht und deren Namen aufgelistet zurückgibt.

```
FUN findCDs: seq[cd] ** nat -> seq[denotation]
```

1.3. (Tut) Suchen in einer Liste II

Wie sind Funktionsdeklaration und -definition zu ändern, wenn nicht nur die Namen, sondern alle Daten der gefundenen CDs im Ergebnis enthalten sein sollen?

Hinweis: Falls ihr die Funktion testen wollt, ist die Backtick-Funktion für den Datentyp `cd` vorgegeben.

2. Aufgabe: Pattern Matching

2.1. (Tut) Einführung

Welcher Gedanke steckt hinter dem Pattern Matching?

2.2. (Tut) Beispiel Fibonacci

Leonardo Fibonacci beschrieb das Wachstum einer Population mit der nachstehenden Folge:

$$f(0) = 0$$
$$f(1) = 1$$
$$f(n) = f(n-1) + f(n-2)$$

Gegeben sind folgende Funktionsdeklaration und -definition für die Umsetzung der Fibonaccifolge in Opal mit Pattern Matching:

```
FUN fib: nat -> nat

DEF fib(0) == 0
DEF fib(1) == 1
DEF fib(n) == fib(n-1) + fib(n-2)
```

Welches Ergebnis liefert der Funktionsaufruf: `>e fib(10)`? Wie muss die Funktionsdefinition geändert werden, damit das korrekte Ergebnis der Fibonaccifolge ermittelt wird?

2.3. (Tut) Pattern Matching für Listen

Schreibt eine Funktion `findCD_PM`, die wie `findCD` aus Aufgabe 1 funktioniert und verwendet dabei Pattern Matching.

2.4. (Pflicht-Hausaufgabe) PriorityQueue Funktionen

In Aufgabenblatt 7 wurden PriorityQueues mit den Funktionen `pop`, `enqueue` und `dequeue` vorgestellt. Implementiert diese mit Pattern Matching!

Hinweis: Schaut in die Tutoriumslösungen zu Blatt 7, informiert euch über den Datentyp und bearbeitet die vorgegebene Struktur in den Vorgaben!

3. Aufgabe: Listenfunktionale

Für das Arbeiten mit Listen gibt es in Opal spezielle Funktionen, sogenannte **Listenfunktionale**. Diese sind in den Strukturen `SeqFilter`, `SeqMap`, `SeqZip` und `SeqReduce` zu finden und wie folgt definiert:

```
FUN filter : (data -> bool) ** seq[data] -> seq[data]
FUN zip    : (from1**from2->to)->seq[from1]**seq[from2]->seq[to]
FUN map    : (from -> to) -> seq[from] -> seq[to]
FUN reduce : (from ** to -> to) ** to -> seq[from] -> to
```

Besprecht ihre Arbeitsweisen und mögliche Einsatzszenarien.
Wir werden Listenfunktionale und ihre Funktionsweise anhand von folgendem Szenario einführen:
Die Werkstatt Müller und Schulte hat seit kurzem den Betrieb aufgenommen und sich auf die Reparatur von Autos, Motorrädern und LKWs spezialisiert. Da die Aufträge auch EDV gestützt aufgenommen werden, habt ihr die Aufgabe einige Funktionen für die Auswertung der Daten zu implementieren. Gegeben sind dabei folgende Datenstrukturen:

```
TYPE vehicle == car bike truck
TYPE status == open inProgress finished
TYPE job == job(object: vehicle, status: status, cost: real)
```

Hinweis: Die folgenden Unteraufgaben sollen **ohne Rekursion** allein mit Hilfe von Listenfunktionalen und Pattern Matching implementiert werden.

Die Lösung jeder der folgenden Unteraufgaben benötigt gerade deswegen meist nur wenige Zeilen. Damit eure Implementierung übersichtlich wird, überlegt vorher, welche bereits definierten Funktionen für die Bearbeitung hilfreich sein können. Strukturiert euren Code sinnvoll mit dem LET bzw. WHERE Konstrukt.

3.1. (Tut) Statusabhängige Aufträge mit `filter` auswählen

Definiert die Funktion `searchJobs`, welche aus einer Auftragsliste mittels eines übergebenen Status all diejenigen Aufträge herausfiltert, die diesem Status entsprechen.

Hinweis: Die Funktion zum Vergleichen des Status ist bereits vorgegeben.

```
FUN = : vehicle ** vehicle -> bool
```

3.2. (Tut) Gesamtkosten mit `reduce` und `map` berechnen

Definiert die Funktion `overallCost`, die alle Kosten einer Auftragsliste summiert.

Hinweis: Verwendet `map`, um das Feld `cost` der Aufträge zu selektieren.

3.3. (Tut) Rabattaktion mit `map` erstellen

Definiert eine Funktion `discount`, die eine Auftragliste erhält und eine Liste mit um 25% gesenkten Reparaturkosten liefert.

3.4. (Pflicht-Hausaufgabe) Unfertige Aufträge

Definiert die Funktion `notFinished`, welche aus einer übergebenen Liste alle Aufträge herausfiltert, die nicht den Status `finished` besitzen.

3.5. (Pflicht-Hausaufgabe) Kosten je Fahrzeugtyp

Definiert die Funktion `costInProgress`, welche aus einer gegebenen Auftragsliste alle Aufträge heraussucht, die den Status `inProgress` besitzen. Aus diesen Aufträgen sollen nun die Kosten für jeden Fahrzeugtyp errechnet werden. Zurückliefern soll die Funktion ein Tripel der Kosten, wobei folgende Rückgabereihenfolge beachtet werden soll: (Kosten der Autos, Kosten der Fahrräder, Kosten der LKWs).

3.6. (Pflicht-Hausaufgabe) Statusänderung

Schriebt eine Funktion `advance`, die eine Auftragsliste übergeben bekommt und den Status aller enthaltenen Aufträge wie folgt ändert:

- aus `open` wird `inProgress`
- aus `inProgress` wird `finished`

Außerdem soll die Funktion ein `bool`-Flag übergeben bekommen und Aufträge der Eingabeliste, die bereits den Status `finished` haben, herausfiltern, wenn das Flag `true` ist.

3.7. (Pflicht-Hausaufgabe) Noch offene Aufträge

Definiert die Funktion `openJobs`, welche aus einer Auftragsliste alle noch offenen Aufträge ermittelt und eine Zeichenkette zurückgibt. Dabei sollen lediglich der Fahrzeugtyp und die Kosten aufgeführt werden. Diese Aufgabe soll ohne Rekursion gelöst werden.

Hinweis: Eine Hilfsfunktion für die Ausgabe eines Auftrags muss geschrieben werden. Die Backtick-Funktion für den Datentyp `vehicle` ist vorgegeben. Die Zeichenkette soll wie folgt aussehen:
“Vehicle:car, Cost:30.5
Vehicle:Bike, Cost:20.3”