Team: 01, Sebastian Cordes, Fabian Beck

Aufgabenaufteilung:

1. Skizze, Sebastian Cordes
2. Skizze, Fabian Beck

Quellenangaben: -

Begründung für Codeübernahme: -

Bearbeitungszeitraum: Mi 15.10.2014

Aktueller Stand: Skizze ist fertig

Änderungen in der Skizze: <Vor dem Praktikum auszufüllen: Welche Änderungen sind bzgl. der Vorabskizze vorgenommen worden.>

Achtung: die Typisierung, hier zB {list,Daten} ist nur innerhalb der jeweiligen ADT zu sehen, d.h. der Stack weiss nicht, dass die Liste die Gestalt {list,Daten} hat!! Daher wäre die Beschreibung etwa sgünstiger als {list,List} zu lesen bzw {stack,Stack} etc.

Skizze:

ADTs:

Die ADTs sollen als 2-Tupel dargestellt werden, auf diese Art kann man durch das Patternmatching von Erlang die ADTs zu ihren Funktionen zuordnen. Ein ADT schaut somit folgendermaßen aus:{list,Daten}.

ADT Liste:

{list,Daten}
Die Liste beginnt bei der Position 1 und es wird nicht destruktiv gearbeitet.

Create:
∅ → list
Erstellt eine leere Liste.

isEmpty:
list → bool
Vergleicht die übergebene Liste mit der leeren Liste. Wenn die übergebene Liste eine leere Liste ist wird wahr zurückgegeben, ist sie keine leere Liste wird falsch zurückgegeben.

laenge:
list → int
Gibt die Länge der übergebenen Liste zurück. Wenn die Liste leer ist wird 0 zurückgegeben. Falls sie nicht leer ist wird zurückgegeben wie viele Elemente sie enthält.

insert:
list × pos × elem → list
Die übergebene Liste muss bereits bis pos-1 befüllt sein um das Element an der übergebenen Position einzufügen. Die übergebene Position darf nicht negativ sein. Ist der übergebene fehlerhaft wird die Liste unverändert zuückgegegen sonst wird eine neue Liste mit dem eingefügten Element zurückgegeben. Um nicht destruktiv zu sein müssen alle Elemente nach der Position um eine Position höher gespeichert werden.

delete:

list × pos → list

Es wird eine neue Liste ohne das Element an der gegebenen Position zurückgegeben. Ist die Position nicht vorhanden wird die Liste unverändert zurückgegeben.

find:

list × elem → pos

Es wird eine Liste und ein Element übergeben. Find gibt die Position des übergebenen Elements zurück. Falls das Element nicht vorhanden ist wird null zurückgegeben.

retrieve:

list × pos → elem

Es wird eine Liste und eine Position übergeben und nach dem Element an der Position gesucht. Ist die Position vorhanden wird das Element zurückgegeben, ist sie nicht vorhanden wird null zurückgegeben.

concat:

list × list → list

Die Eingabewerte sind zwei Listen und der Rückgabewert ist eine Liste. Die zurückgegebene Liste soll aus den beiden übergebenen bestehen. Die Listen sollten hintereinander gehängt sein(Bsp.: [4,5,6],[1,2,3] → [4,5,6,1,2,3]).

ADT Stack:

{stack,Daten}
Der Stack besteht aus einer Liste und arbeitet nach dem LIFO Prinzip (Last-in First-out).

createS:

∅ → stack

Gibt einen leeren Stack zurück.

push:

stack × elem → stack

Übergeben wird ein Stack und ein Element. Der Rückgabewert ist ein neuer Stack an dessen Liste das Element an der ersten Position eingefügt wurde.

pop:

stack → stack

Übergeben wird ein Stack. Der Rückgabewert ist ein neuer Stack an dessen Liste das erste Element gelöscht wurde.

top:

stack → elem

Übergeben wird ein Stack. Es wird das erste Element das in der Liste des Stacks steht zurückgegeben.

isEmptyS:

stack → bool

Übergeben wird ein Stack. Ist die Liste des Stacks leer wird wahr zurückgegeben sonst falsch.

ADT queue:

{queue,Daten}
Die Queue basiert auf zwei Stacks (In-Stack und out-Stack) und arbeitet nach dem FIFO Prinzip (First-in First-out).

createQ:
∅ → queue
Gibt eine leere Schlange zurück.

front:
queue → elem (Selektor)
Übergeben wird eine Queue. Das erste Element aus dem Out-Stack wird zurückgegeben ohne es vom Out-Stack zu löschen. Ist der Out-Stack leer wird der In-Stack Element für Element in den Out-Stack geschrieben und das neue erste Element ausgegeben. Falls beide Stacks leer sind wird null zurückgegeben.

enqueue:
queue × elem → queue
Es wird eine Queue und ein Element übergegen. Es wird eine neue Queue zurückgegeben deren In-Stack das Element an oberster Stelle enthält.

Dequeue:
queue → queue (Mutator)
Übergeben wird eine Queue. Das erste Element des Out-Stacks wird gelöscht. Ist der Out-Stack leer wird der In-Stack Element für Element in den Out-Stack geschrieben und dann das erste Element des Out-Stacks gelöscht. Sind beide leer wird die Queue unverändert zurückgegeben.

isEmptyQ:
queue → bool
Es wird eine Queue übergeben. Sind beide Stacks leer wird wahr zurückgegeben sonst falsch.

ADT Array:

{array,Daten}
Das Array basiert auf einer Liste, fängt bei 0 an, arbeitet destruktiv, wird mit 0 initialisiert und hat keine feste Größe. Die Länge wird vom Element an der größten Position das keine 0 ist bestimmt.

initA:
∅ → array
Gibt ein Array zurück das auf 0 initialisiert ist.

setA:
array × pos × elem → array
Übergeben wird ein Array, eine Position und ein Element. Zurück gegeben wird ein Array das an der übergebenen Position das Element stehen hat. Es wird destruktiv gearbeitet also werden die an der Position stehenden Daten überschrieben.

getA:
array × pos → elem
Übergeben wird ein Array und eine Position. Gibt das Element das an der gegebenen Position im Array steht zurück.

lengthA:

array → pos

Übergeben wird ein Array. Zurückgegeben wird die Länge des Arrays. Also die Anzahl der Element die bis zum letzten Element, das nicht 0 ist, vorhanden sind.