ADT Liste

Die Liste ist ein Tupel, welches aus einem Typidentifier und einer Subliste besteht. Eine Subliste ist wiederum ein Tupel, welches entweder leer ist, um das Ender der Liste zu symbolisieren, oder aus einem beliebigen Element und einer weiteren Subliste besteht. Die Subliste ist also immer wieder in sich geschachtelt, bis das Ende der Liste erreicht ist.

## create

Erstellt eine Instanz von dem Listentupel mit einer leeren Subliste.

## isEmpty

Es wird geschaut, ob die oberste, also erste, Subliste im Tupel die leere Subliste ist.

## laenge

Die Subliste wird rekursive durchiteriert bis die leere Subliste erreicht ist. Dabei zähl ein Akkumulator die Tiefe der Liste durch.

## insert

Es wird über die Liste rekursiv iteriert mit zwei Akkumulatoren. Einer für die Position und einer für die neu zusammengebaute Liste. Bei jedem Iterationsschritt wird das aktuelle Element in der Ausgangsliste an den Akkumulator angefügt. Falls man sich an der gewünschten Position befindet, wird vor dem aktuellen Element noch das einzufügende Element angefügt. Ab dem Punkt werden nur noch die restlichen Elemente angefügt. Falls sich der Index außerhalb der Länge der Liste befindet, wird das Element einfach an das Ende der Liste angefügt.

## delete

Delete arbeitet ähnlich wie insert, der Unterschied liegt darin, dass anstatt des Einfügen eines neuem Elements, wird das Element am gewünschten Index ausgelassen und einfach übergangen. Danach läuft das anfügen der Elemente normal weiter.

## find

Es wird über die Liste rekursiv iteriert mit einem Akkumulator, der die aktuelle Position mitnimmt. Es wird so lange durch die Liste gegangen, bis das gesuchte Element dem aktuellem Element entspricht und somit der Akkumulatorstand zurückgegeben wird oder bis man ans Ende der Liste gelangt ist, wo ein negativer Wert zurückgegeben wird.

## retreive

Es wird über die Liste rekursiv iteriert mit einem Akkumulator, der die aktuelle Position mitnimmt. Wenn der Iterator die gewünschte Position erreicht hat wird das dort befindliche Element zurückgegeben. Falls der gewünschte Index ausserhalb der Liste befindet, wird ein definiertes, leeres zurückgegeben.

## concat

Es wird die erste Liste rekursiv auf ein Iterator geschrieben. Danach wird die zweite Liste mit dem gleichen Verfahren darüber gesetzt.

ADT Stack

Der Stack besteht aus einem Typ und unserer Liste. Dabei repräsentiert das zuletzt eingefügte Element das oberste Element des Stacks.

## createS

Erzeugt eine neue Instanz des Tupels mit einer leeren Liste.

## push

Fügt ein Element am Ende der Liste ein.

## pop

Entfernt das zuletzt hinzugefügte Element vom Stack und gibt diesen zurück. Beim leeren Stack passiert nichts.

## top

Gibt das zuletzt hinzugefügte Element des Stacks zurück (ohne den Stack zu verändern). Ist der Stack leer, passiert nichts.

isEmptyS

Der Stack ist leer, wenn die interne Liste leer ist.

ADT Queue

Eine Queue besteht aus zwei Stacks - einem In-Stack und einem Out-Stack. Beim Hinzufügen eines Elements wird dieses im In-Stack abgelegt. Durch die Realisierung über zwei Stacks wird gewährleistet, dass das zuerst hinzugefügte Element auch als erstes wieder aus der Queue entnommen wird (FIFO-Prinzip). Es wird solange auf den In-Stack geschrieben, bis eine Leseoperation auf den Out-Stack ausgeführt wird. Dann wird der Inhalt des In-Stacks auf den Out-Stack umgeschichtet, und das erste Element des Out-Stacks gelesen. Ist der Out-Stack nicht leer, wird nur das erste Element des Out-Stacks gelesen, ohne den Inhalt aus dem In-Stack umzuschichten.

## createQ

Erzeugt eine neue Instanz des Tupels mit zwei leeren Stacks (IN & OUT).

## front

Gibt das erste Element der Queue (--> oberstes Element des Out-Stacks) zurück, ohne es zu entfernen.

## enqueue

Reiht das Element am Ende der Queue ein. D.h., es wird auf dem In-Stack abgelegt.

## dequeue

Entfernt das erste Element der Queue, also das oberste Element des Out-Stacks und gibt sie zurück.

## isEmptyQ

Prüft, ob sowohl In- und Out-Stack leer sind.

ADT Array

Ein Array besteht aus einem Typ und einer Liste, die standardmäßig leer ist. Das Array hat keine festgelegt Länge und kann somit immer wachsen, wenn es benötigt wird. Das Array beginnt bei der Position 0.

## initA

Erzeugt eine neue Instanz des Tupels mit einer leeren Liste.

## setA

Ersetzt das Element an angegebener Position mit dem einzufügenden Element. Wenn die angegebene Position über die Länge der aktuellen Liste hinaus geht, wird die Liste bis zur angegebenen Position mit Nullen aufgefüllt und anschließend das Element am Ende hinzugefügt.

## getA

Gibt das Element an der angebenene Position zurück. Falls die angebenene Position über die aktuelle Länge hinaus geht, wird 0 zurückgegeben. lengthA: Gibt die Länge der internen Liste zurück.