Team: 11, Finn Dohrn und Fabian Pfaff

Aufgabenaufteilung:

1. Erstellung der Skizze (gemeinsam)
   Liste, Stack, Queue, Array (gemeinsam)
2. Erstellung der Skizze (gemeinsam)
   Liste, Stack, Queue, Array (gemeinsam)

Quellenangaben: -

Begründung für Codeübernahme: -

Bearbeitungszeitraum:
14.10.14 – 6h Skizzenanfertigung und erste Implementierung (gemeinsam)
16.10.15 – 1h Korrektur der Skizze

Aktueller Stand: Die Software ist vollständig skizziert und teilweise implementiert.

Skizze:

Allgemeine Anmerkungen für alle ADTs:

Jedes ADT wird als Tupel dargestellt. An erster Stelle befindet sich der Typ. Nur mit dieser Typisierung können wir in Erlang zwischen den ADT's unterscheiden.

Mutator-Funktionen verändern die Datenstrukturen.

Alle anderen Funktionen verändern die Datenstruktur nicht.

List: {list, Tupel}
Stack: {stack, List}
Queue: {queue, Stack, Stack}
Array: {array, List}

ADT Liste (list.erl):

Funktionen:
create: ∅ → list
isEmpty: list → bool
laenge: list → int
insert: list × pos × elem → list (Mutator)
delete: list × pos → list (Mutator)
find: list × elem → pos
retrieve: list × pos → elem
concat: list × list → list

Bemerkungen:
- Eine leere Liste hat 0 Elemente.
- Die Indexierung der Liste beginnt bei 1.
- Wenn nicht vorhandene Elemente gelöscht werden, oder Elemente an Positionen hinzugefügt werden sollen, die nicht möglich sind, soll die Fehlerbehandlung durch „Ignorieren“, also eine unveränderte Rückgabe der Datenstruktur erfolgen.
- Die Liste ist nicht destruktiv: Falls ein Element an eine Position eingefügt werden soll, wo bereits ein Element existiert wird dieses nicht überschrieben sondern eingeschoben.
- Findet „find“ kein Element wird -1 zurückgegeben, damit der aufrufende Programmcode darauf angemessen reagieren kann.

ADT Stack (stack.erl):

createS: ∅ → stack
push: stack × elem → stack (Mutator)
pop: stack → stack (Mutator)
top: stack → elem
isEmptyS: stack → bool

Bemerkungen:
- Das ADT Stack wird mittels der ADT Liste realisiert.
- Deswegen soll der Stack genau dann leer sein, wenn die interne ADT Liste leer (isEmpty) ist.
- Wie üblich funktioniert der Stack mit dem LIFO-Prinzip (Last-In-First-Out). Das letzte Element was reingeht, geht auch als erstes wieder raus.
- Wenn aus einem leeren Stack etwas entfernt werden soll, soll der unveränderte (leere) Stack zurückgegeben werden.

ADT queue (schlange.erl):

createQ: ∅ → queue
front : queue → elem (Selektor)
enqueue : queue × elem → queue (Mutator)
dequeue : queue → queue (Mutator) (Mutator)
isEmptyQ : queue → bool

Bemerkungen:
- Es werden zwei Stacks verwendet.
- Die Queue ist genau dann leer, wenn beide Stacks leer sind.
- Die ADT Queue arbeitet im Gegensatz zum Stack mit dem FIFO (First-in-First-Out) Prinzip.
- Trotzdem soll sie aus dem ADT Stack erstellt werden.
- In dem ersten Stack werden Elemente eingefügt. Aus dem zweiten Stack werden Elemente zurückgegeben.
- Soll entfernt oder zurückgegeben werden und der zweite Stack ist leer müssen vorher alle Elemente des ersten Stacks in den zweiten Stack verschoben werden.

ADT array (array.erl):

initA: ∅ → array
setA: array × pos × elem → array (Mutator)
getA: array × pos → elem
lengthA: array → pos

Bemerkungen:
- Die Indexierung des Arrays beginnt bei 0.
- Falls ein Element bereits an einer Position exisitiert, wird es überschrieben.
- Das Array hat eine dynamische und wird ohne festgelegte Größe initialisiert.
- Unbeschriebene Positionen im Array werden mit 0 initialisiert.
- Der größte beschriebene Index ist gleichzeitig die Größe des Arrays.
- Wenn ein “Out-of-Bounds”-Zugriff erfolgt wird eine -1 zurückgegeben, damit der aufrufende Programmcode darauf angemessen reagieren kann.