Team: < 04 >, < Schäfer & Ahmad >

Aufgabenaufteilung:

1. <Aufgaben, für die Teammitglied 1 verantwortlich ist>, <Dateien, die komplett/zum Teil von Teammitglied 1 implementiert/bearbeitet wurden>
2. <Aufgaben, für die Teammitglied 2 verantwortlich ist>, <Dateien, die komplett/zum Teil von Teammitglied 2 implementiert/bearbeitet wurden>

Quellenangaben: <

http://users.informatik.haw-hamburg.de/~klauck/AlguDat/aufg1.html

http://users.informatik.haw-hamburg.de/~klauck/AlguDat/AD.pdf

>

Begründung für Codeübernahme: < Es wurde kein Code übernommen. >

Bearbeitungszeitraum: <

Für den Entwurf: 2 Stunden 12.10

30 Minuten 21.10 (Erweiterung)

>

Aktueller Stand: < Entwurf fertig >

Änderungen im Entwurf: < KEINE >

Es sollen folgende ADTs in Erlang/OTP implementiert werden und die Dateien müssen wie vorgegeben heißen:

- Liste („liste.erl“)
- Stack („stack.erl“)
- Queue („schlange.erl“)
- Array („arrayS.erl“)

Es folgen nun erst einmal die Signaturen der ADTs. Bei denen ist es sehr wichtig, dass diese exakt eingehalten werden, um ggf. ADTs austauchbar mit anderen Gruppen zu machen, die sich ebenfalls an diese Signaturen halten.

Signatur

Methodenname: Parameter1 x Parameter2 x ParameterN -> Rückgabewert

„∅“ = steht für keinen Parameter

ADT Liste

Zu realisieren als einfach verkettete Liste.

create: $\varnothing \rightarrow$ list
Initialisiert eine Liste (Erzeugung) und liefert diese zurück

isEmpty: list $\rightarrow$ bool
Prüft ob die übergebene Liste leer ist und gibt true zurück wenn ja, sonst false

laenge: list $\rightarrow$ int
Gibt die Länge (Anzahl Elemente innerhalb) der übergebenen Liste zurück

insert: list $\times$ pos $\times$ elem $\rightarrow$ list
Fügt der übergebenen Liste an der übergebenen Position das übergebene Element hinzu und gibt die modifizierte Liste zurück

delete: list $\times$ pos $\rightarrow$ list
Entfernt das Element an der übergebenen Position (falls vorhanden) in der übergebenen Liste und gibt die modifizierte Liste zurück
find: list $\times$ elem $\rightarrow$ pos
Sucht nach einem übergebenen Element in der übergeben Liste und gibt die Position dessen zurück (falls gefunden)

retrieve: list $\times$ pos $\rightarrow$ elem
Gibt das Element an der übergebenen Position in der übergeben Liste zurück (falls vorhanden)

concat: list $\times$ list $\rightarrow$ list
Konkatiniert die übergebenen Listen und gibt das Ergebnis zurück

Weitere Vorgaben:

- Eine ADT Liste ist zu implementieren: als einfach verkette Liste
- Die Liste beginnt bei Position 1
- Die Liste arbeitet nicht destruktiv, d.h. wird ein Element an einer vorhandenen Position eingefügt, wird das dort stehende Element um eine Position verschoben

ADT Stack

Zu implementieren auf der ADT Liste, soll heißen nur die Funktionen der ADT Liste dürfen intern verwendet werden um mit dem Stack zu arbeiten.

createS: $\varnothing \rightarrow$ stack
Initialisiert einen Stack (Erzeugung) und liefert diesen zurück

push: stack $\times$ elem $\rightarrow$ stack
Fügt dem übergebenen Stack das übergebene Element hinzu und gibt den modifizierten Stack zurück

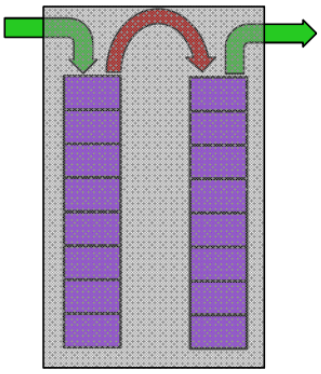
pop: stack $\rightarrow$ stack
Entfernt vom übergebenen Stack das oberste Element (falls vorhanden) und gibt den modifizierten Stack zurück

top: stack $\rightarrow$ elem
Gibt das oberste Element (falls vorhanden) des übergebenen Stacks zurück

isEmptyS: stack $\rightarrow$ bool
Prüft ob der übergebene Stack leer ist, falls ja wird true zurück gegegeben sonst false

ADT Queue

Zu implementieren auf der ADT Stack, mit zwei expliziten Stacks.
Ein „In-Stack“ und ein „Out-Stack“, die gemeinsam eine Schlange realisieren sollen.



in-stack (feste Position)
out-stack (feste Position)
„umstapeln: wenn out-stack leer ist,
werden (bei Bedarf) **alle** aus dem
in-stack in den out-stack „umgestapelt“

createQ: ∅ → queue
Initialisiert eine Queue (Erzeugung) und liefert diese zurück

front: queue → elem (Selektor)
Gibt das vorderste Element zu der übergebenen Queue zurück (falls vorhanden)

enqueue: queue × elem → queue
Fügt der übergebenen Queue das übergebene Element ganz hinten hinzu und gibt die modifizierte Queue

dequeue: queue → queue (Mutator)
Entfernt das vorderste Element der übergebenen Queue (falls vorhanden) und gibt die modifizierte Queue zurück

isEmptyQ: queue → bool
Gibt true zurück, wenn die übergebene Queue leer ist, sonst false

ADT Array

Zu implementieren auf der ADT Liste.

initA: $\varnothing \rightarrow$ array
Initialisiert ein Array (Erzeugung) und liefert dieses zurück

setA: array $\times$ pos $\times$ elem $\rightarrow$ array
Fügt dem übergebenen Array an der übergebenen Position das übergebene Element hinzu und gibt das modifizierte Array zurück

getA: array $\times$ pos $\rightarrow$ elem
Gibt das Element vom übergebenen Array an der übergebenen Position zurück (falls vorhanden, sonst 0...siehe weitere Vorgaben)

lengthA: array $\rightarrow$ pos
Gibt die länge des übergebenen Arrays zurück

Weitere Vorgaben:

- Das Array beginnt bei Position 0
- Das Array arbeitet destruktiv, d.h. wird ein Element an einer vorhandenen Position eingefügt, wird das dort stehende Element überschrieben
- Die Länge des Arrays wird bestimmt durch die bis zur aktuellen Abfrage größten vorhandenen und explizit beschriebenen Position im array
- Das Array ist mit 0 initialisiert, d.h. greift man auf eine bisher noch nicht beschriebene Position im Array zu erhält man 0 als Wert
- Das Array hat keine Größenbeschränkung, d.h. bei der Initialisierung wird keine Größe vorgegeben

Fehlerbehandlung

Sollten nicht vorhandene Elemente gelöscht werden, in der Liste an unmöglicher Stelle eingefügt werden, von einem leeren Stack das oberste Element gelöscht werden etc. ist die Fehlerbehandlung durch "Ignorieren" durchzuführen, d.h. es wird so gehandelt, als wäre die Operation in Ordnung gewesen und z.B. eine nicht modifizierte Datenstruktur zurück gegeben.

Wichtig

- Es muss eine Struktur für die einfach verkettete Liste gebaut werden, die die Anforderungen erfüllt.
- Es muss entschieden werden, wie man damit umgeht, dass die ADT Liste mit Position 1 beginnt und die ADT Array, die auf dieser aufbauen soll, mit Position 0.
- ADT Liste arbeitet nicht destruktiv, ADT Array arbeitet destruktiv.
- In der ADT Array darf an beliebiger Position ein Element eingefügt werden, bei der ADT Liste nur an bereits bestehenden Positionen oder hinten dran.