KELLER (STACKS)

Informell:

(Zwischen-) Speicher nach dem LIFO-Prinzip: Last-In-First-Out-Listen

Anwendung:

- Übersetzung rekursiv → iterativ + Stapel
- Compiler
  - •• Laufzeitkeller für Prozeduraufrufsverwaltung
  - •• Auswertung allg. arithm. Ausdrücke
  - •• Überprüfen wohlgeformter Klammerstrukturen

als ADT:

Sei *A* eine gegebene Menge von Datenelementen. Es muß sichergestellt sein, daß der Elementtyp *A* die Operationen implementiert, die im ADT Elem definiert sind (s. Spezifikation des ADT List(*A*)).

STACK(*A*) besteht aus:

- Menge der Listen $L = \langle a_1, a_2, \ldots, a_n \rangle$ (endl. Folgen) über *A;*
- den folgenden Operationen: (Wir verwenden die Operationen des ADT List(*A*) für die Spezifikation der Stack-Operationen. Dabei stellt der Listenanfang das oberste Kellerelement dar.)

ADT Stack(*A*)

Sei *s* von Typ Stack(*A*):

Stack(): Konstruktor-Methode; wie bei allgemeinen Listen;

clear(), boolean isEmpty(): wie bei allgemeinen Listen;

push(Elem e); Kellern: Lege Element e auf den Stack;

(V) -

(N) Falls *s*.isEmpty() = **true**, ist
s.push(*e*) ≡ *s*.append(*e*);
anderenfalls: ist *it* = *s*.getIterator(), dann gilt
s.push(*e*) ≡ *it*.first(); *s*.insert(*it,e*)

Elem top(): liefere "oberstes" Element des Stacks;

(V) ¬ *s*.isEmpty()

(N) Ist *it* = *s*.getIterator(), dann gilt
s.top() = *e* mit *it*.first();
e = *s*.retrieve(*it*)

pop(); Entkellern: lösche oberstes Element des Stacks;

(V) ¬ *s*.isEmpty()

(N) Ist *it* = *s*.getIterator(), dann gilt
s.pop(*e*) ≡ *it*.first();
s.delete(*it*)

Anmerkung: top() und pop() sind hier orthogonal zueinander eingeführt.

Die folgenden Operationen stellen sicher, daß Stack(*A*) selbst Implementierung des ADT Elem sein kann. Die Spezifikationen entsprechen denen der Operationen auf allgemeinen Listen.

Stack copy():	Kopieroperation
boolean equals(Stack s):	Test auf Gleichheit
boolean less(Stacks s):	Test auf "Kleiner"
StringtoString():	Konvertierungsoperation