Die Datenstrukturen Stapel und Schlange

Stapel (auch Stack oder Keller)

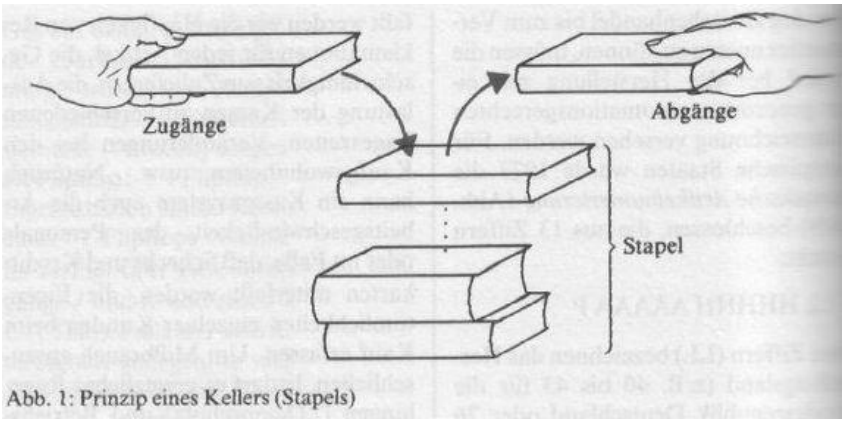


Abb. 1: Prinzip eines Kellers (Stapels)

(Quelle: Duden Informatik, Dudenverlag 1993)

Prinzip: Last in First out, LiFo (oder „wer zuletzt kommt, darf als erster gehen“)

Definition: Eine Datenstruktur heißt *Stapel*, wenn Elemente nur an einem Ende, der Stapelspitze (`top`), hinzugefügt oder entfernt werden können. Für Stapel sind die folgenden Operationen definiert:

1. Erzeugen eines leeren Stapels (`init`)
2. Hinzufügen eines Elements (`push`)
3. Prüfung, ob ein gegebener Stapel leer ist (`isempty`)
4. Inspizieren der Stapelspitze (`top`)
5. Entfernen der Stapelspitze (`pop`)

Anwendungen:
Auswerten von Rechenausdrücken, syntaktische Analyse von Programmen, Bau von Übersetzern; Undo, Aufrufen von Unterprogrammen, Tiefensuche in Graphen

Schlange (Queue)

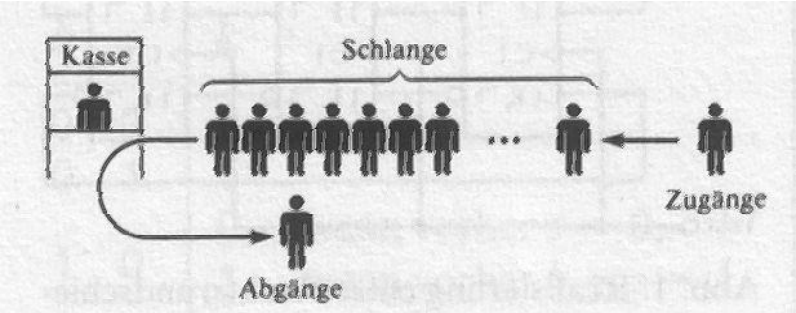


Abb. 1: Beispiel für eine Schlange

(Quelle: Duden Informatik, Dudenverlag 1993)

Prinzip: First in First out, FiFo (oder „ wer zuerst kommt, mahlt zuerst“)

Definition: Eine Datenstruktur heißt *Schlange*, wenn Elemente nur an einem Ende (dem Ende der Schlange) angefügt und nur am anderen Ende (dem Kopf der Schlange) entfernt werden können. Für Schlangen sind die folgenden fünf Operationen definiert:

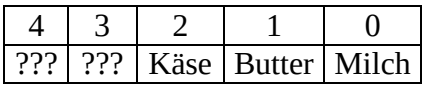
1. Erzeugen einer neuen leeren Schlange (`init`)
2. Hintenanfügen eines Elements an eine gegebene Schlange (`enqueue` oder `enter` bzw. `put`)
3. Prüfung, ob die Schlange leer ist (`isempty`)
4. Inspizieren des Schlangenkopfes (`front`)
5. Entfernen des Schlangenkopfes (`dequeue` oder `remove` bzw. `get`)

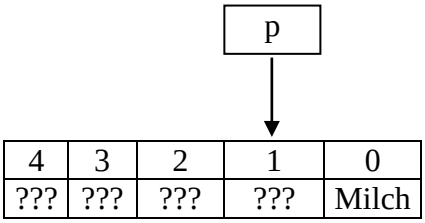
Anwendungen:
Druckaufträge (Druckerwarteschlange), Verwaltung von Prozessaufrufen im Betriebssystem, Hardwareschnittstellen zur seriellen Ein-/Ausgabe (Pufferspeicher z.B. Tastatur)

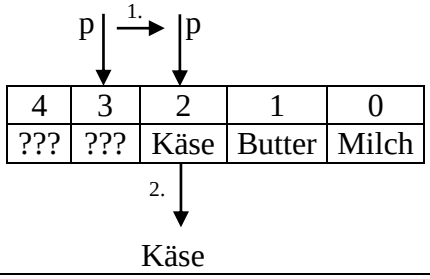
Umsetzung von Stapeln und Schlangen mit Feldern

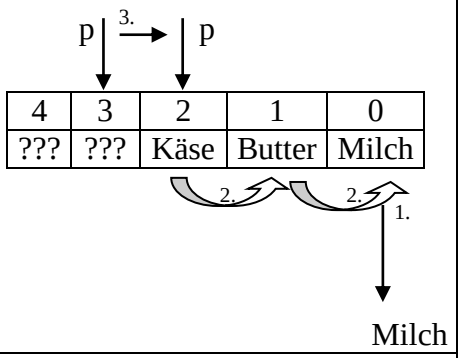
Sollen die Datentypen Stapel und Schlange in Programmen verwendet werden, müssen sie in Programmiersprachen erzeugt werden. In Pascal / Delphi stehen dafür die Datentypen Pointer (Zeiger) und Array (Feld) zur Verfügung. Die Erzeugung mit Hilfe von Zeigern bietet den Vorteil der dynamischen Speicherverwaltung, d.h. es wird nur soviel Speicher verwendet wie für die Daten wirklich benötigt wird. Leider ist der Umgang mit Zeigern etwas komplizierter als der mit Feldern. Die Arbeit mit Stapeln und Schlangen lässt sich in drei Teile zerlegen. Erstens die *Initialisierung*, zweitens das *Anhängen von Datensätzen* und drittens das *Abhängen von Datensätzen*.
Ein Feld besteht aus einer festen Anzahl von Elementen. Der Inhalt dieser ist immer vom gleichen Datentyp. Die Verkettung innerhalb des Feldes erfolgt intern. Man kann über Feldnummern auf die Inhalte zugreifen. Für Stapel und Schlangen sind eindimensionale Felder zu nutzen. Als erstes wird das Feld initialisiert. Ab dieser Stelle wird ein fester Speicherbereich im Arbeitsspeicher reserviert, der während der Programmausführung nicht wieder freigegeben wird. Der Inhalt des Feldes ist an dieser Stelle nicht definiert. Es wird eine Variable benötigt, die die Nummer des letzten Datensatzes speichert.

<table>
<tr>
<td>program einkaufsliste;

type
dsatz = array[0..30] of string;
var artikel: dsatz;
…..</td>
<td></td>
<td>Array

</td>
</tr>
<tr>
<td>procedure finit;
begin
p:=0;
end;</td>
<td>Initialisierung:
p (Feldende) wird auf null gesetzt, d.h. die Länge der Liste (Stapel o. Schlange) ist null. Also die Liste ist leer.</td>
<td></td>
</tr>
<tr>
<td>procedure push;
begin
artikel[p]:=’Milch’;
p:=p+1
end;</td>
<td>Anfügen eines Datensatzes (Stapel und Schlange)

1. Inhalt in Feldelement legen
2. p (Feldende) um eins erhö-hen</td>
<td>
</td>
</tr>
<tr>
<td>function pop: dsatz;
begin
p:=p-1;
pop:=artikel[p];
end;</td>
<td>Entfernen eines Datensatzes von einem Stapel

1. p um eins verringern
2. Inhalt am Ende auslesen</td>
<td>
</td>
</tr>
<tr>
<td>function get: dsatz;
get:=artikel[0];
for h1:=1 to p do
artikel[h1-1]:=artikel[h1];
p:=p-1
end;</td>
<td>Entfernen eines Datensatzes von einer Schlange

1. Inhalt vom Kopf auslesen
2. Feldinhalte nach dem Kopf um 1 nach vorn rutschen lassen
3. p um eins verringern</td>
<td>
</td>
</tr>
<tr>
<td>Prüfung ob leer

function isempty: boolean;
begin
if p=0 then isempty
end;</td>
<td>Prüfung ob leer

Wenn die Länge null ist, dann ist Stapel oder Schlange leer</td>
<td></td>
</tr>
</table>