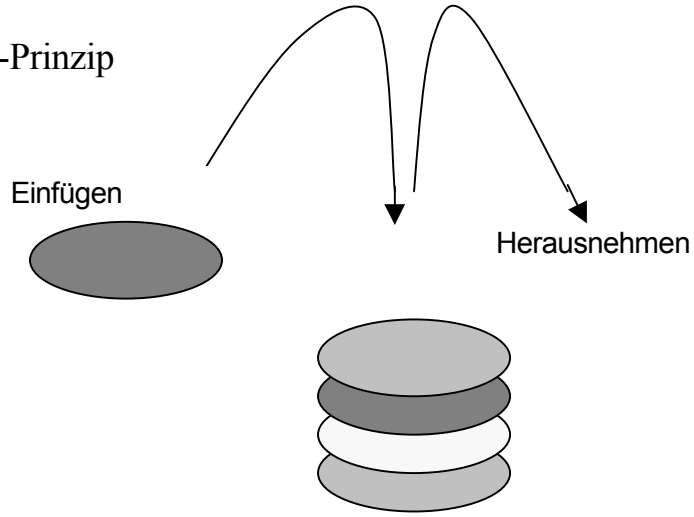
MINDMAP Informatik

Algorithmen und Datenstrukturen

Der Abstrakte Datentyp „Keller“

Ein Keller oder auch Stapel (engl. Stack) genannt ist eine abstrakte Datenstruktur, bei der Elemente eingefügt und wieder entfernt werden können. Dabei kann jedoch immer nur auf dasjenige Element zugegriffen werden, das zuletzt eingefügt wurde.

LIFO=Last In First Out-Prinzip



Datenstruktur:
Der Stack hat eine Struktur, die die Umsetzung des genannten LIFO-Prinzips ermöglicht.

Operationen:
Um die Umsetzung des LIFO-Prinzips zu ermöglichen besitzt die Datenstruktur „Keller“ spezifische Operationen. In der folgenden Tabelle sind die Operationen, mit ihrer Bedeutung, den betroffenen Daten und einem Prozedurkopf zur Implementation in Pascal (Schnittstelle des ADT‘s in Pascal) angegeben.

Operation / Bedeutung	Betroffene Daten	Schnittstelle zur Implementation in Pascal/
create Diese Operation erzeugt einen neuen (zumeist leere) Keller und initialisiert die Zeiger für Kelleranfang und Kellerende etc.	Keller	procedure create (VAR k : TKeller);
Empty Diese boolsche Operation prüft, ob der Keller leer ist und gibt entsprechend wahr oder falsch zurück	Keller	function empty (k : TKeller) : boolean;
Full Diese boolsche Operation prüft ober der Kelle (im Falle einer platzbegrenzten Implementierung) bereits voll ist und gibt wahr oder falsch zurück.	Keller	function full (k : TKeller) : boolean;
Pop Diese Operation entfernt das zuletzt auf den Keller gelegte Element.	Keller, Element	function pop (VAR k : TKeller):TElement;
Push Diese Operation legt ein Element auf den Keller.	Keller, Element	procedure push(VAR k : TKeller, e: TElement);
Top Diese Operation zeigt nur das zuletzt auf den Keller gelegte Element und nimmt es nicht vom Keller.	Keller	function top (k : Tkeller) : Telement;

Achtung! Einige Implementationen des ADT’s Keller verwenden andere Pop- und Top-Operationen: Häufig ist Pop so definiert, dass es nur einen Eintrag vom Keller löscht, ohne ihn zu liefern. Bei einigen IImplementationen fehlt die Top-Operation völlig!

Algorithmen und Datenstrukturen

Implementation des abstrakten Datentyps „Keller“

```
UNIT stack; {Implementation des ADTs Stack mit einem Feld}
{Diese Implementationsart wird als statisch bezeichnet, weil
 die Größe der Datenstruktur bei der Definition bereits bestimmt ist
 und sich nicht dynamisch entwickeln kann.}



INTERFACE    {Schnittstellendefinition: Hier werden Datenstruktur und Operationen angegeben}
USES import; {Unit für den Elementtyp}
CONST maximum = 50; {maximal mögliche Länge des Kellers}

TYPE TElement = Tinhalt;  {Elementtyp Tinhalt ist in der Unit import definiert}
     TKeller  = RECORD    {Datenstruktur des ADT's Keller}
                top      : 0..maximum;
                speicher : array[1..maximum] of TElement;
                END;

procedure create (VAR k : TKeller);             {erzeugt neuen, leeren Kelle}
function empty (k : Tkeller) : boolean;         {prüft, ob Kleer leer}
function full (k : Tkeller) : boolean;          {prüft, ob Keller voll}
function pop (VAR k : Tkeller):TElement;         {holt eine Elemnt vom Keller und entfernt es dabei}
procedure push(VAR k : Tkeller; e : TElement);  {legt ein Element auf den Keller}
function top (k: TKeller):TElement;             {zeigt das oberste Kellerelemnt an ohne es vom Kleer zu entfernen}
```

```
IMPLEMENTATION {Implementation der Operationen}

procedure create (VAR k : TKeller);             {erzeugt neuen, leeren Kelle}
begin
k.top := 0; {Keller ist leer}
end;

function empty (k : Tkeller) : boolean;         {prüft, ob Kleer leer}
begin
if k.top=0
   then empty:=true
   else empty:=false;
end;

function full (k : Tkeller) : boolean;          {prüft, ob Keller voll}
begin
if k.top=maximum
   then full:=true
   else full:=false;
end;

function pop (VAR k : Tkeller):TElement;         {holt eine Elemnt vom Keller und entfernt es dabei}
begin
if (NOT empty(k)) then
                 begin
                  pop := k.speicher[k.top];
                  k.top:=k.top-1;
                 end
                 else writeln('Keller ist leer!');

end;

procedure push(VAR k : Tkeller; e : TElement);  {legt ein Element auf den Keller}
begin
if (NOT full(k)) then begin
                      k.top:=k.top+1;
                      k.speicher[k.top]:=e;
                      end
               else writeln('Keller ist voll!');
end;
```

```
function top (k: TKeller):TElement;             {zeigt das oberste Kellerelemnt an ohne es vom Kleer zu entfernen}
begin
top:=k.speicher[k.top];
end;

BEGIN        {Keine Initialisierungen etc.}
END.


UNIT import;
{Diese Unit implementiert nur einen einfachen
Inhaltstyp Tinhalt zur Verwendung in ADTs wie z.B. Queue, Stack usw.}
INTERFACE
TYPE Tinhalt = integer; {Der Inhaltstyp besteht hier aus Integer-Zahlen}
IMPLEMENTATION
BEGIN
END.

program Stacktest;                    {Programm zum Testen des ADTs Keller}
uses wincrt,import,stack;

var k:TKeller;
  test:integer;

begin
create(k);
push(k,1);
push(k,2);
push(k,3);
writeln(top(k));
test:=pop(k);
writeln(test);
writeln(top(k));
end.


{Hinweise zu Pascal-Units:
 Ein Pascal-Unit wird als *.pas-Datei erzeugt und in das Unit-Verzeichnis
 der Entwicklungsumgebung compiliert. Dadurch wird aus der *.pas-Datei eine *.tpu-Datei.}
```