3. tétel

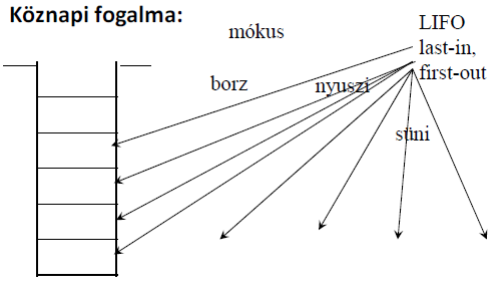
Adatszerkezetek és algoritmusok vizsga

Frissült: 2013. január 28.

Verem

ADT axiomatikus és funkcionális specifikáció

Elképzelés:



Amit betettem utoljára, azt tudom először kivenni.

ADT Axiomatikus leírás

E adattípus feletti V verem típus jellemzése
Műveletek

- empty: $\rightarrow V$ - üres verem - létrehozása
- isempty: $V \rightarrow L$ - üres-e a verem?
- push: $V \times E \rightarrow V$ - elem betétele
- pop: $V \rightarrow V \times E$ - elem kivétele
- top: $V \rightarrow E$ - legfelső elem lekérdezése
- full

Megszorítások: *pop* és *top* értelmezési tartománya: $V \setminus \{\text{empty}\}$
Axiómák

- isempty(empty) / v=empty $\rightarrow$ isempty(v)
- isempty(v) $\rightarrow$ v=empty

- ¬ isempty(push(v,e))
- pop(push(v,e))=(v,e)
- push(pop(v))=v
- top(push(v,e))=e

ADT funkcionális leírás

Matematikai reprezentáció: a verem rendezett párok halmaza:

- 1. komponens: a veremben elhelyezett érték
- 2. komponens: a verembe helyezés időpontja.

Megszorítás (invariáns): az idő értékek különbözők.
Nem így implementáljuk!

Példa - a "pop" specifikációja

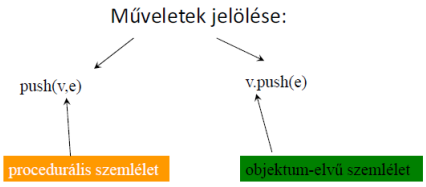
$V \times E$ - állapottér típusai ($V = \{(e_i, t_i)...\}$)
v e - változói
V - kezdeti értékkel rendelkezik
v'
Előfeltétel:

$$Ef = (v = v' \wedge v' \neq \emptyset)$$

Utófeltétel:

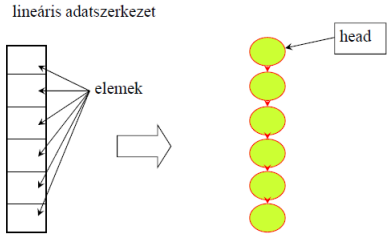
$$Uf = ((v = v' \backslash \{(e_j, t_j)\}) \wedge (e = e_j) \wedge ((e_j, t_j \in v') \wedge (\forall i((e_i, t_i) \in v' \wedge i \neq j) : t_j > t_i))$$

A műveletek jelölése



ADS statikus és dinanimus reprezentáció

A verem működése ADS-ben:



Ábrázolási módok:

Aritmetikai / statikus ábrázolás

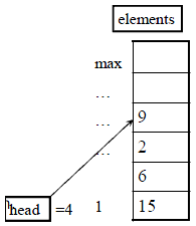
Egy max hosszú vektor (ez az elemek tömbje)

$$elements[1...max]$$

és a verem tetejének mutatója

$$head \in [0, max]$$

ha $head = 0 \Leftrightarrow$ üres a verem.



Műveletek megvalósítása

- **v.Empty** - üresre állítja a vermet

  ```
  v.head <- 0
  ```

  vagy:

  ```
  v.head := 0
  ```

  más jelöléssel.
  '←' jelentése: értékadás.

- **v.IsEmpty** - üres-e a verem? - logikai értéket ad vissza

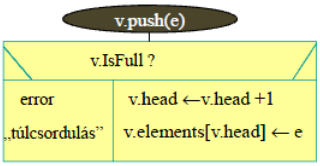
  ```
  return (v.head = 0)
  ```

- **v.IsFull** - tele van a verem? - logikai értéket ad vissza

  ```
  return (v.head = max)
  ```

- **v.push(e)** - e-t beteszi a v verem tetejére

  ```
  if v.IsFull
    then error „túlcsordulás";
    else v.head <- v.head +1;
      v.elements[v.head] <- e;
  end if;
  ```



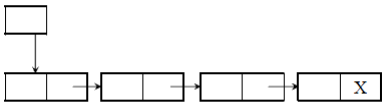
- **v.pop** - kiveszi a legfelső elemet és visszaadja

```
if v.IsEmpty
  then error „alulcsordulás";
  else v.head <- v.head -1;
    return v.elements[v.head+1];
end if;
```

- **v.top** - lekérdez a legfelső elemet

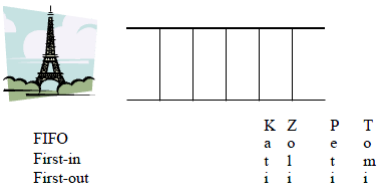
```
if v.IsEmpty
  then error „alulcsordulás";
  else return v.elements[v.head];
end if;
```

Láncolt ábrázolás - gyakorlaton volt



Sor

Köznapi fogalma:



Amit beteszek először, azt tudom legelőször kivenni.

ADT axiomatikus és funkcionális specifikáció

Axiomatikus leírás. E alaptípus feletti S sor típus jellemzése. Műveletek:

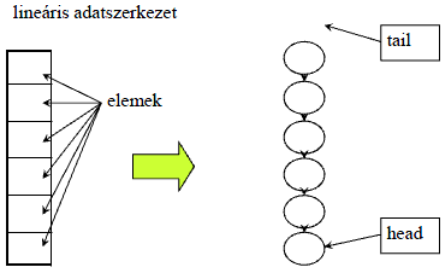
- Empty: $\rightarrow S$ - üres sor, létrehozásnál.
- IsEmpty: $S \rightarrow L$ - üres-e a sor?

- In: $S \times E \to S$ - elem betétele a sorba
- Out: $S \to S \times E$ - elem kivétele a sorból
- First: $S \to E$ - első elem lekérdezése
- IsFull

Megszorítások: Out és First értelmezési tartománya $S \setminus \{\text{Empty}\}$
Axiómák, funkcionális specifikáció - hasonló. (lehet hogy kérdezik!!!)
Axiómák:

- s = Empty $\to$ IsEmpty(s)
- IsEmpty(s) $\to$ s = Empty
- $\neg$ IsEmpty(In(s, e))
- $\neg$ IsEmpty(s) $\to$ Out(In(s, e))$_2$ = Out(s)$_2$
- $\neg$ IsEmpty(s) $\to$ In(Out(s)$_1$, e) = Out(In(s, e))$_1$
- $\neg$ IsEmpty(s) $\to$ Out(In(s,e))$_2$ = e
- First(s) = Out(s)$_2$

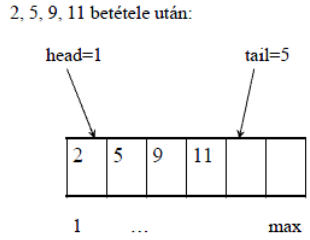
ADS statikus és dinanimus reprezentáció



Aritmetikai / statikus ábrázolás

Kell

- Egy *max* hosszú vektor (ez az elemek tömbje): elements[1...max].
- a sor első elemének mutatója: head $\in$ [1,..max]
- a sor első üres (vagy utolsó) helyének mutatója: tail $\in$ [1,..max]



De kell még egy jelző - empt - mely mutatja, hogy a sor üres-e. Ez kezdetben igaz. (esetleg egy számlálót, hogy hány elem van a sorban)

Műveletek megvalósítása

- **s.Empty** - üresre állítja a sort

```
s.head <- 1;
s.tail <- 1;
s.empt <- true
```

- **s.IsEmpty** - üres a sor? logikai értéket ad vissza.

```
return s.empt
```

- **s.IsFull** - tele van a sor? - logikai értéket ad vissza

```
return ((not s.empt) and (s.head = s.tail))
```

- **s.In(e)** - e-t beteszi a sor végére, s-tail-t ciklikusan növeli.

```
if s.IsFull
  then error „túlcsordulás";
  else s.empt <- false;
    s. elements[s.tail] <- e;
    if s.tail = max
      then s.tail <- 1
      else s.tail <- s.tail+1;
    end if
end if
```

- **s.Out** - kiveszi és visszaadja az *s* sor első elemét, s.head-et ciklikusan növeli, figyeli hogy nem üres-e a sor.

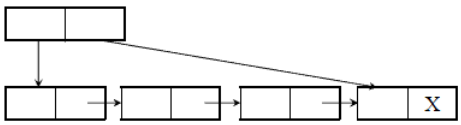
```
if s.empt
  then error „alulcsordulás";
  else e <- s. elements[s.head] ;
    if s.head = max
      then s.head <- 1
      else s.head <- s.head+1;
    end if;
    if s.head = s.tail then s.empt <- true;
```

- **s.First** - visszaadja az *s* sor első elemét. Figyeli hogy nem üres-e a sor.

```
if s.empt
  then error „alulcsordulás";
  else return s. elements[s.head] ;
end if
```

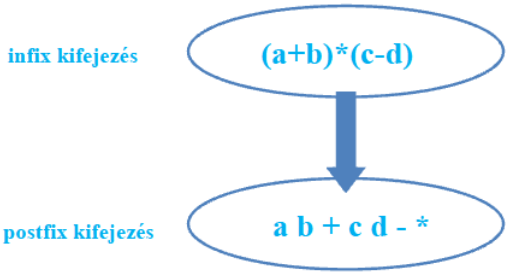
Lehetne az is, hogy a darabszámot tároljuk.

Láncolt ábrázolás is lehet



Lengyelforma

Infix kifejezésből postfix kifejezést alkot:

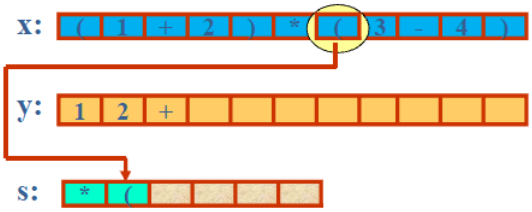


Előnye, hogy a műveleti jelek olyan sorrendben követik egymást, amilyen sorrendben végre kell hajtani azokat, és a műveleti jel (operátor) közvetlenül az operandusai után áll.

Menete:

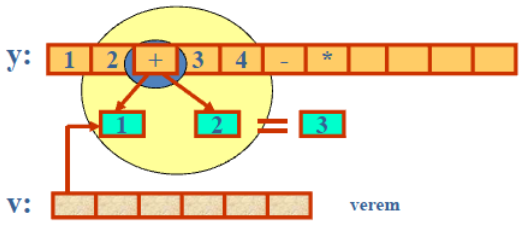
Adott egy x, y sorozat és egy s verem.

- Az x sorozatot balról jobbra dolgozzuk fel. A sorozat végét ; jelzi.
- Ha a következő szimbólum nyitózárójel '(', betesszük a verembe.
- az Ha operandus: az y sorozatba tesszük, kiírjuk.
- Ha operátor: legfeljebb a nyitózárójelig kivesszük a veremből az operátornál nagyobb prioritású operátorokat és kiírjuk azokat, az operátort pedig betesszük a verembe.
- Ha csukó zárójel ')', kiírjuk y-ba a verem tetején lévő elemeket a nyitózárójelig ')'. Utána a verem tetejéből kiszedjük a nyitózárójelet is.
- a kifejezés végén: kiírjuk, ami a veremben maradt az y-ba.



Kiértékelés. Van egy y lengyelformára alakított forumula, illetve a v verem.

- végigmegyünk y-n.
- Ha az adott elem operandus - betesszük a verembe.
- Ha operátor - kivesszük a veremből a 2. és az 1. operandust, majd elvégzem a műveletet és az eredményt beteszem a verembe.
- A kifejezés végén az eerdmény a verem tetején lesz.



Algoritmusok

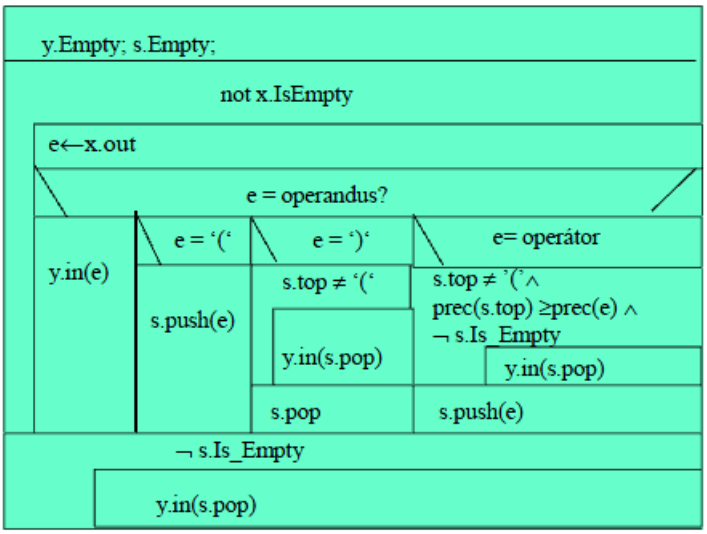
Az x sor tartalmazza a szintaktikailag helyes kifejezést, ún. token-ekből áll. Egy token lehet

- operandus
- (bináris) operátor
- nyitó, csukó zárójel ( )

Az y sorba hozzuk létre a postfix formájú kifejezést.

Közben felhasználva az s vermet, amely operátorokat és nyitó zárójeleket tartalmazhat.

Lengyel formára hozás



Kiértékelés