

Liste. Stive. Cozi

SD 2020/2021

Tipurile abstracte LLin, LLinOrd, Stiva, Coadă

- Liste liniare

- Implementarea cu tablouri

- Implementarea cu liste simplu înlănțuite

- Liste liniare ordonate

- Stive

- Cozi

Aplicație la conversie de expresii aritmetice

Tipurile abstracte LLin, LLinOrd, Stiva, Coadă

Liste liniare

Implementarea cu tablouri

Implementarea cu liste simplu înlănțuite

Liste liniare ordonate

Stive

Cozi

Aplicație la conversie de expresii aritmetice

Liste liniare – exemple

- ▶ Studenți
 - ▶ (*Adriana, George, Luiza, Maria, Daniel*)
- ▶ Examene
 - ▶ (*Mate, Logică, SD, ACSO, IP, ENG*)
- ▶ Zilele săptămânii
 - ▶ (*L, M, Mi, J, V, S, D*)
- ▶ Lunile anului
 - ▶ (*Ian, Feb, Mar, Apr, Mai, Iun, Iul, Aug, Sep, Oct, Nov, Dec*)

Tipul abstract LLin

► **OBIECTE:** $L = (e_0, \dots, e_{n-1}), \quad n \geq 0$

► $e_i \in \text{Elt}$ (tipul abstract al elementelor)

► **Relații:**

- e_0 primul element al listei;
- e_{n-1} ultimul element al listei;
- e_i elementul predecesor lui e_{i+1} .

▶ listaVida()

▶ intrare: nimic

▶ ieșire: $L = ()$ (lista cu zero elemente)

▶ insereaza()

▶ intrare:

▶ $L = (e_0, \dots, e_{n-1}), \quad k \in \text{Nat}, \quad e \in \text{Elt}$

▶ ieșire:

▶ $L = (\dots, e_{k-1}, e, e_k, \dots)$, dacă $0 \leq k \leq n$

▶ eroare în caz contrar

insereaza() – exemple

$$L = (a, b, c, d, e, f, g)$$

► $\text{insereaza}(L, 0, x) \Rightarrow L = (x, a, b, c, d, e, f, g)$

Obs. indexul elementelor a, \dots, g crește cu 1.

► $\text{insereaza}(L, 2, x) \Rightarrow L = (a, b, x, c, d, e, f, g)$

► $\text{insereaza}(L, 7, x) \Rightarrow L = (a, b, c, d, e, f, g, x)$

► $\text{insereaza}(L, 10, x) \Rightarrow$ eroare

► $\text{insereaza}(L, -7, x) \Rightarrow$ eroare

LLin – operații

▶ elimina()

▶ intrare:

▶ $L = (e_0, \dots, e_{n-1}), \quad k \in \text{Nat}$

▶ ieșire:

▶ $L = (\dots, e_{k-1}, e_{k+1}, \dots)$, dacă $0 \leq k \leq n-1$

▶ eroare în caz contrar

LLin – operații

▶ elimina()

▶ intrare:

$$\text{▶ } L = (e_0, \dots, e_{n-1}), \quad k \in \text{Nat}$$

▶ ieșire:

$$\text{▶ } L = (\dots, e_{k-1}, e_{k+1}, \dots), \text{ dacă } 0 \leq k \leq n-1$$

▶ eroare în caz contrar

Exemple:

$$L = (a, b, c, d, e, f, g)$$

$$\text{▶ } \text{elimina}(L, 2) \Rightarrow L = (a, b, d, e, f, g)$$

Obs. indexul elementelor d, \dots, g descrește cu 1.

$$\text{▶ } \text{elimina}(L, 10) \Rightarrow \text{eroare}$$

$$\text{▶ } \text{elimina}(L, -7) \Rightarrow \text{eroare}$$

LLin – operații

▶ alKlea()

▶ intrare:

▶ $L = (e_0, \dots, e_{n-1}), \quad k \in \text{Nat}$

▶ ieșire:

▶ e_k , dacă $0 \leq k \leq n-1$

▶ eroare în caz contrar

LLin – operații

- ▶ `alKlea()`
 - ▶ intrare:
 - ▶ $L = (e_0, \dots, e_{n-1}), \quad k \in \text{Nat}$
 - ▶ ieșire:
 - ▶ e_k , dacă $0 \leq k \leq n-1$
 - ▶ eroare în caz contrar

Exemple:

$$L = (a, b, c, d, e, f, g)$$

- ▶ `alKlea(L, 0)` \Rightarrow a
- ▶ `alKlea(L, 2)` \Rightarrow c
- ▶ `alKlea(L, 6)` \Rightarrow g
- ▶ `alKlea(L, 20)` \Rightarrow eroare
- ▶ `alKlea(L, -2)` \Rightarrow eroare

LLin – operații

▶ elimTotE()

▶ intrare:

▶ $L = (e_0, \dots, e_{n-1}), \quad e \in \text{Elt}$

▶ ieșire:

▶ lista L din care s-au eliminat toate elementele egale cu e

▶ elimTotE()

▶ intrare:

▶ $L = (e_0, \dots, e_{n-1})$, $e \in \text{Elt}$

▶ ieșire:

▶ lista L din care s-au eliminat toate elementele egale cu e

Exemple:

$$L = (a, b, c, a, b, c, a)$$

▶ $\text{elimTotE}(L, a) \Rightarrow (b, c, b, c)$

▶ $\text{elimTotE}(L, c) \Rightarrow (a, b, a, b, a)$

▶ $\text{elimTotE}(L, d) \Rightarrow (a, b, c, a, b, c, a)$

- ▶ `parcurge()`

- ▶ intrare:

- ▶ $L = (e_0, \dots, e_{n-1})$, o procedură / funcție viziteaza()

- ▶ ieșire:

- ▶ lista L în care toate elementele au fost procesate aplicând viziteaza()

▶ `parcure()`

▶ intrare:

▶ $L = (e_0, \dots, e_{n-1})$, o procedură / funcție viziteaza()

▶ ieșire:

▶ lista L în care toate elementele au fost procesate aplicând viziteaza()

Exemple:

$$L = (1, 2, 3, 1, 2, 3)$$

▶ `parcure(L, oriDoi())` \Rightarrow (2, 4, 6, 2, 4, 6)

▶ `parcure(L, incrementeaza())` \Rightarrow (2, 3, 4, 2, 3, 4)

LLin – operații

▶ poz()

▶ intrare:

▶ $L = (e_0, \dots, e_{n-1}), \quad e \in \text{Elt},$

▶ ieșire:

- ▶ prima poziție pe care apare e în L sau
- ▶ -1 dacă e nu apare în L .

► poz()

► intrare:

$$\text{► } L = (e_0, \dots, e_{n-1}), \quad e \in \text{Elt},$$

► ieșire:

- prima poziție pe care apare e în L sau
- -1 dacă e nu apare în L .

Exemple:

$$L = (a, b, c, a, b, c, d)$$

- $\text{poz}(L, a) \Rightarrow 0$
- $\text{poz}(L, c) \Rightarrow 2$
- $\text{poz}(L, d) \Rightarrow 6$
- $\text{poz}(L, x) \Rightarrow -1$

LLin – operații

- ▶ `lung()`
 - ▶ intrare:
 - ▶ $L = (e_0, \dots, e_{n-1})$,
 - ▶ ieșire:
 - ▶ n – lungimea listei L .

- ▶ `lung()`
 - ▶ intrare:
 - ▶ $L = (e_0, \dots, e_{n-1})$,
 - ▶ ieșire:
 - ▶ n – lungimea listei L .

Exemple:

$$L = (a, b, c, a, b, c, d)$$

$$\text{▶ } \text{lung}(L) \Rightarrow 7$$

Tipurile abstracte LLin, LLinOrd, Stiva, Coadă

Liste liniare

Implementarea cu tablouri

Implementarea cu liste simplu înlănțuite

Liste liniare ordonate

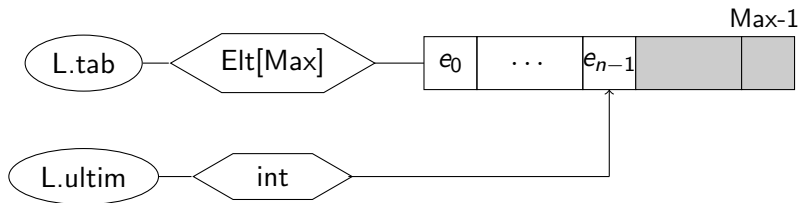
Stive

Cozi

Aplicație la conversie de expresii aritmetice

LLin – implementarea cu tablouri

- ▶ Reprezentarea obiectelor $L = (e_0, \dots, e_{n-1})$



- ▶ L este o *structură*
 - ▶ $L.tab$ – un câmp de tip tablou pentru memorarea elementelor;
 - ▶ $L.ultim$ – un câmp pentru memorarea poziției ultimului element.

► insereaza()

- deplasează elementele de pe pozițiile $k, k + 1, \dots$ la dreapta cu o poziție;
- inserează e pe poziția k ;
- excepții:
 - $k < 0, \quad k > L.ultim + 1 \quad (n)$
 - $L.ultim = Max - 1.$

LLin – implementarea cu tablouri

```
procedure insereaza(L, k, e)  
begin  
    if ( $k < 0$  or  $k > L.ultim + 1$ ) then  
        throw "eroare-pozitie incorecta"  
    if ( $L.ultim \geq Max - 1$ ) then  
        throw "eroare-spatiu insuficient"  
    for  $j \leftarrow L.ultim$  downto  $k$  do  
         $L.tab[j + 1] \leftarrow L.tab[j]$   
     $L.tab[k] \leftarrow e$   
     $L.ultim \leftarrow L.ultim + 1$   
end
```

► Timpul de execuție: $O(n)$.

► `parcurge()`

```
procedure parcurge(L, viziteaza())  
begin  
    for  $i \leftarrow 0$  to L.ultim do  
        viziteaza(L.tab[i])  
end
```

- Dacă *viziteaza*() procesează un element în $O(1)$, atunci *parcurge*() procesează lista în $O(n)$ (n este numărul elementelor listei).

Tipurile abstracte LLin, LLinOrd, Stiva, Coadă

- Liste liniare

- Implementarea cu tablouri

- Implementarea cu liste simplu înlănțuite**

- Liste liniare ordonate

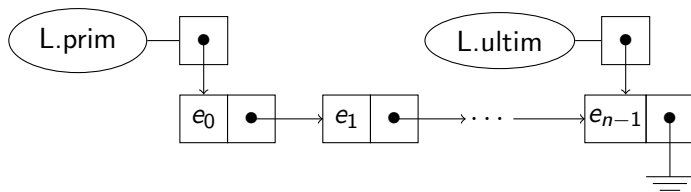
- Stive

- Cozi

Aplicație la conversie de expresii aritmetice

LLin – implementarea cu structuri înlănțuite

- ▶ Reprezentarea obiectelor $L = (e_0, \dots, e_{n-1})$



- ▶ L este o *structură* cu două câmpuri
 - ▶ $L.prim$ – pointer la primul element al listei;
 - ▶ $L.ultim$ – pointer la ultimul element al listei.
- ▶ un $nod * p$ (aflat la adresa din p) are *două câmpuri*:
 - ▶ $p \rightarrow elt (= e_i)$ – memorează informația utilă din nod;
 - ▶ $p \rightarrow succ$ – memorează adresa nodului succesor.

LLin – implementarea cu structuri înlănțuite

▶ `insereaza()`

- ▶ parcurge elementele de pe pozițiile $0, 1, \dots, k - 1$;
- ▶ inserează un nou element după poziția $k - 1$;
 - ▶ creează noul nod;
 - ▶ memorează informații;
 - ▶ reface legături.
- ▶ excepții:
 - ▶ lista vidă;
 - ▶ $k = 0$;
 - ▶ $k = n$;
 - ▶ $k < 0, k > n$.

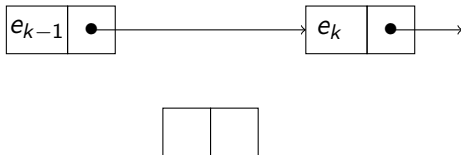
LLin – implementarea cu structuri înlănțuite

► Cazul general



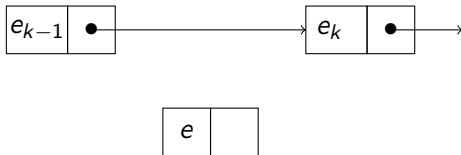
LLin – implementarea cu structuri înlănțuite

► Cazul general



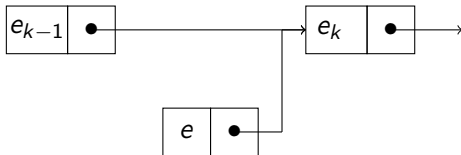
LLin – implementarea cu structuri înlănțuite

► Cazul general



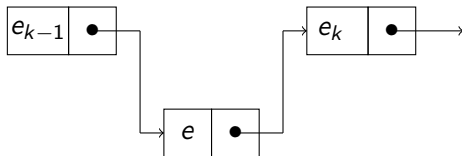
LLin – implementarea cu structuri înlănțuite

► Cazul general



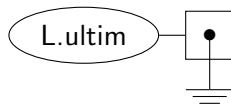
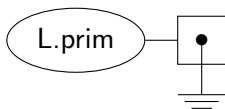
LLin – implementarea cu structuri înlănțuite

► Cazul general



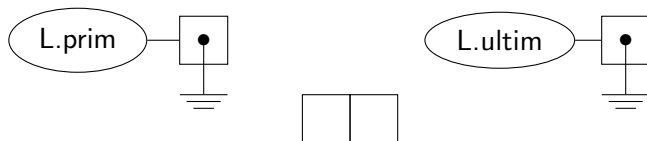
LLin – implementarea cu structuri înlănțuite

- Cazul particular: lista vidă



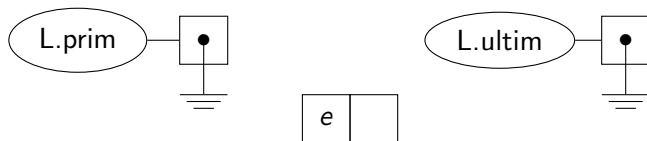
LLin – implementarea cu structuri înlanțuite

- Cazul particular: lista vidă



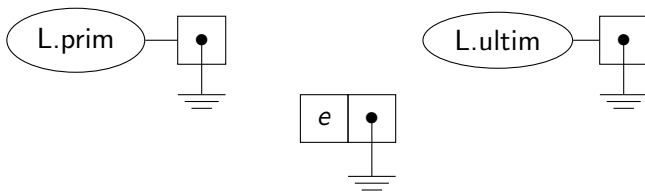
LLin – implementarea cu structuri înlănțuite

- Cazul particular: lista vidă



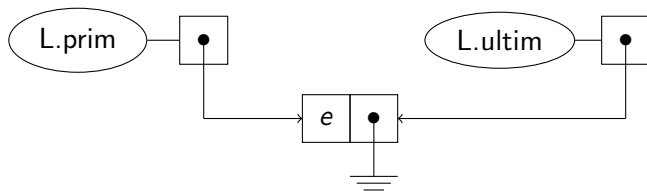
LLin – implementarea cu structuri înlanțuite

- Cazul particular: lista vidă



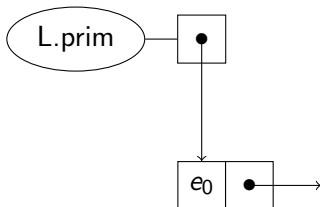
LLin – implementarea cu structuri înlănțuite

- Cazul particular: lista vidă



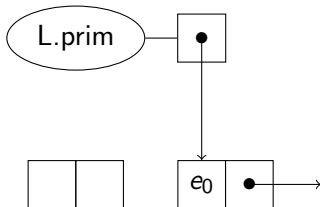
LLin – implementarea cu structuri înlănțuite

- Cazul particular: inserarea la începutul listei



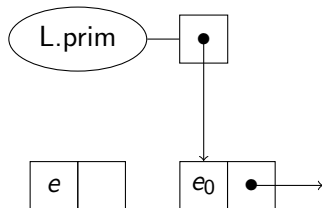
LLin – implementarea cu structuri înlănțuite

- Cazul particular: inserarea la începutul listei



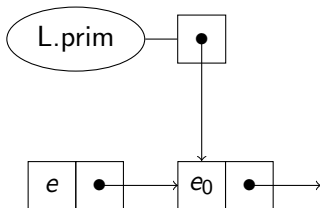
LLin – implementarea cu structuri înlănțuite

- Cazul particular: inserarea la începutul listei



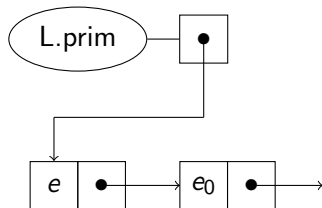
LLin – implementarea cu structuri înlănțuite

- Cazul particular: inserarea la începutul listei



LLin – implementarea cu structuri înlănțuite

- Cazul particular: inserarea la începutul listei



LLin – implementarea cu structuri înlănțuite

```
procedure insereaza(L, k, e)  
begin  
  if (k < 0) or (k ≠ 0 and L.prim == NULL) then  
    throw "eroare-pozitie incorecta"  
  new(q);  q → elt ← e  
  if (k == 0 or L.prim == NULL) then  
    q → succ ← L.prim;  L.prim ← q  
    if (L.ultim == NULL) then  
      L.ultim ← q  
  else  
    p ← L.prim;  j ← 0  
    while (j < k - 1 and p ≠ L.ultim) do  
      p ← p → succ;  j ← j + 1  
    if (j < k - 1) then  
      throw "eroare-pozitie incorecta"  
    q → succ ← p → succ;  p → succ ← q  
    if (p == L.ultim) then  
      L.ultim ← q  
end
```

- ▶ Linie poligonală de puncte.
 - ▶ Punct: structură cu două câmpuri x și y ;
- ▶ crearea unei liste

```
procedure creeazaLista( $L$ )  
begin  
     $L \leftarrow listaVida()$   
    /* citeste  $n$  */  
    for  $i \leftarrow 0$  to  $n - 1$  do  
        /* citeste  $p.x$ ,  $p.y$  */  
        insereaza( $L, 0, p$ )  
end
```

- ▶ Obs. Timpul de execuție depinde de implementare.

- ▶ Multiplică cu 2 coordonatele unui punct:

```
procedure ori2Punct(p)  
begin  
     $p.x \leftarrow p.x * 2$   
     $p.y \leftarrow p.y * 2$   
end
```

- ▶ Multiplică cu 2 coordonatele unei linii poligonale:

```
procedure ori2Linie(p)  
begin  
    parcurge(L, ori2Punct())  
end
```

- ▶ translatează punct:

```
procedure trPunct(p, dx, dy)  
begin  
    p.x  $\leftarrow$  p.x + dx  
    p.y  $\leftarrow$  p.y + dy  
end
```

- ▶ translatează linie poligonală:

```
procedure trLinie(L, dx, dy)  
begin  
    parcurge(L, trPunct())  
end
```

Tipurile abstracte LLin, LLinOrd, Stiva, Coadă

- Liste liniare

- Implementarea cu tablouri

- Implementarea cu liste simplu înlănțuite

- Liste liniare ordonate

- Stive

- Cozi

Aplicație la conversie de expresii aritmetice

Liste liniare ordonate: LLinOrd

► OBIECTE:

$$L = (e_0, \dots, e_{n-1}), \quad n \geq 0, \quad e_i \in \text{Elt}, \quad e_0 \leq e_1 \leq \dots \leq e_{n-1}$$

► Operații:

► listaVida()

► intrare: nimic

► ieșire: $L = ()$ (lista cu zero elemente)

► insereaza()

► intrare: $L = (e_0, \dots, e_{n-1})$, $e \in \text{Elt}$

► ieșire: $L = (\dots, e_{k-1}, e, e_k, \dots)$, dacă $e_{k-1} \leq e \leq e_k$
($e_{-1} = -\infty$, $e_n = +\infty$)

Liste liniare ordonate: LLinOrd

▶ elimina()

- ▶ intrare: $L = (e_0, \dots, e_{n-1})$, $e \in \text{Elt}$
- ▶ ieșire: $L = (\dots, e_{k-1}, e_{k+1}, \dots)$, dacă $e = e_k$
eroare în caz contrar

▶ alKlea()

▶ parcurge()

▶ poz()

LLinOrd – implementarea cu tablouri

```
function poz(L, e)  
begin  
     $p \leftarrow 0$ ;     $q \leftarrow L.ultim$   
     $m \leftarrow (p + q)/2$   
    while ( $L.tab[m] \neq e$  and  $p < q$ ) do  
        if ( $e < L.tab[m]$ ) then  
             $q \leftarrow m - 1$   
        else  
             $p \leftarrow m + 1$   
             $m \leftarrow (p + q)/2$   
    if ( $L.tab[m] == e$ ) then  
        return  $m$   
    else  
        return  $-1$   
end
```

- ▶ Implementarea cu tablouri: $O(\log_2 n)$;
- ▶ Implementarea cu liste înlănțuite: $O(n)$;

Tipurile abstracte LLin, LLinOrd, Stiva, Coadă

- Liste liniare

- Implementarea cu tablouri

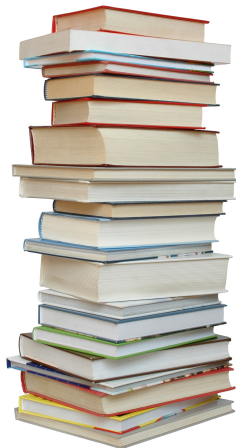
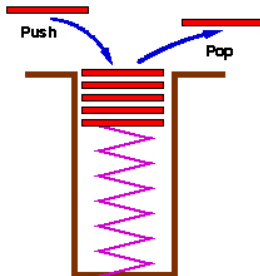
- Implementarea cu liste simplu înlănțuite

- Liste liniare ordonate

- Stive

- Cozi

Aplicație la conversie de expresii aritmetice



▶ Aplicații directe

- ▶ Istoricul paginilor web vizitate într-un browser;
- ▶ Sevența “undo” într-un editor de text;
- ▶ Șiruri de apeluri recursive ale unui subprogram.

▶ Aplicații indirecte

- ▶ Structură de date auxiliară în anumiți algoritmi;
- ▶ Componentă a altor structuri de date.

Tipul abstract Stiva

► OBIECTE:

Liste în care se cunoaște vechimea elementelor introduse:
liste LIFO (*Last-In-First-Out*).

► Operații:

► `stivaVida()`

► intrare: nimic

► ieșire: $S = ()$ (lista vidă)

► `esteVida()`

► intrare: $S \in \text{Stiva}$

► ieșire:

– **true** dacă S este vidă;

– **false** dacă S nu este vidă.

Tipul abstract Stiva

▶ Operații:

▶ push()

- ▶ intrare: $S \in \text{Stiva}$, $e \in \text{Elt}$
- ▶ ieșire: S la care s-a adăugat e ca ultim element introdus (cel cu vechimea cea mai mică).

▶ pop()

- ▶ intrare: $S \in \text{Stiva}$
- ▶ ieșire:
 - S din care s-a eliminat ultimul element introdus (cel cu vechimea cea mai mică);
 - eroare dacă S este vidă.

▶ top()

- ▶ intrare: $S \in \text{Stiva}$
- ▶ ieșire:
 - ultimul element introdus în S (cel cu vechimea cea mai mică);
 - eroare dacă S este vidă.

Stiva – implementare cu liste

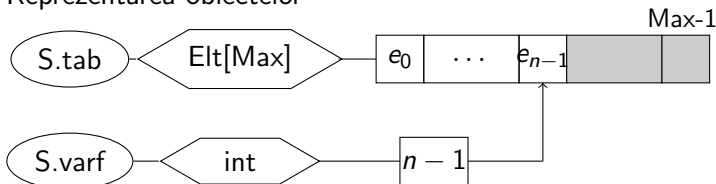
tipul Stiva		tipul LLin
$push(S, e)$	=	$insereaza(S, 0, e)$
$pop(S, e)$	=	$elimina(S, 0)$
$top(S)$	=	$alKlea(S, 0)$

sau

tipul Stiva		tipul LLin
$push(S, e)$	=	$insereaza(S, lung(S), e)$
$pop(S, e)$	=	$elimina(S, lung(S) - 1)$
$top(S)$	=	$alKlea(S, lung(S) - 1)$

Stiva – implementarea cu tablouri

► Reprezentarea obiectelor

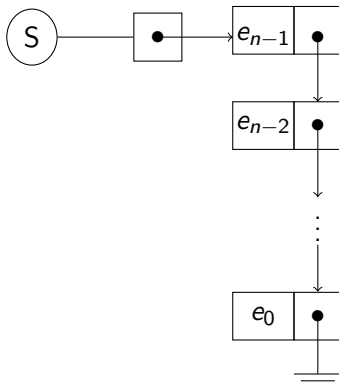


► implementarea operațiilor

```
procedure push( $S, e$ )  
begin  
    if  $S.varf == Max - 1$  then  
        throw "eroare"  
    else  
         $S.varf \leftarrow S.varf + 1$   
         $S.tab[varf] \leftarrow e$   
end
```

Stiva – implementarea cu structuri înlănțuite

► Reprezentarea obiectelor



Stiva – implementarea cu structuri înlănțuite

► Implementarea operațiilor

► `push()`

```
procedure push(S, e)  
begin  
    new(q)  
    q → elt ← e  
    q → succ ← S  
    S ← q  
end
```

► `pop()`

```
procedure pop(S)  
begin  
    if S == NULL then  
        throw "eroare"  
    q ← S  
    S ← S → succ  
    delete(q) end
```

Tipurile abstracte LLin, LLinOrd, Stiva, Coadă

- Liste liniare

- Implementarea cu tablouri

- Implementarea cu liste simplu înlănțuite

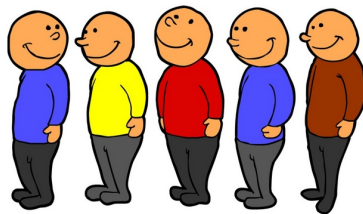
- Liste liniare ordonate

- Stive

- Cozi

Aplicație la conversie de expresii aritmetice

Coadă



- ▶ Aplicații directe

- ▶ Liste / fire de așteptare;
- ▶ Accesul la resurse partajate.
Exemplu: imprimante.

- ▶ Aplicații indirecte

- ▶ Structură de date auxiliară în anumiți algoritmi.

Tipul abstract Coadă

► OBIECTE:

Liste în care se cunoaște vechimea elementelor introduse:
liste FIFO (*First-In-First-Out*).

► Operații:

► `coadaVida()`

► intrare: nimic

► ieșire: $C = ()$ (lista vidă)

► `esteVida()`

► intrare: $C \in \text{Coadă}$

► ieșire:

– **true** dacă C este vidă;

– **false** dacă C nu este vidă.

Tipul abstract Coadă

► Operații:

► `insereaza()`

- intrare: $C \in \text{Coadă}$, $e \in \text{Elt}$
- ieșire: C la care s-a adăugat e ca ultim element introdus (cel cu vechimea cea mai mică).

► `elimina()`

- intrare: $C \in \text{Coadă}$
- ieșire:
 - C din care s-a eliminat primul element introdus (cel cu vechimea cea mai mare);
 - eroare dacă C este vidă.

► `citeste()`

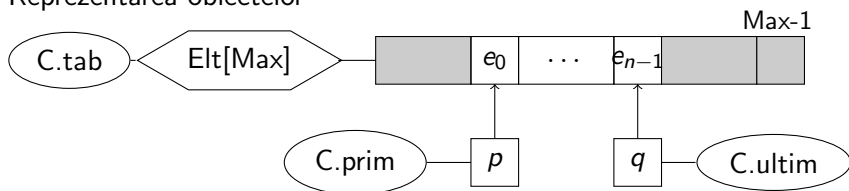
- intrare: $C \in \text{Coadă}$
- ieșire:
 - primul element introdus în C (cel cu vechimea cea mai mare);
 - eroare dacă C este vidă.

Coadă – implementare cu liste

tipul Coadă		tipul LLin
<i>insereaza</i> (C, e)	=	<i>insereaza</i> ($C, \text{lung}(C), e$)
<i>elimina</i> (C)	=	<i>elimina</i> ($C, 0$)
<i>citeste</i> (S)	=	<i>alKlea</i> ($C, 0$)

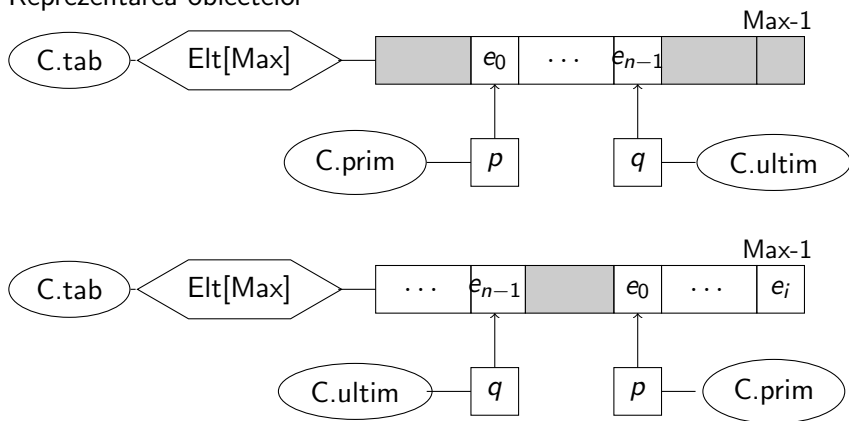
Coadă – implementarea cu tablouri

► Reprezentarea obiectelor



Coadă – implementarea cu tablouri

► Reprezentarea obiectelor



Coadă – implementarea cu tablouri

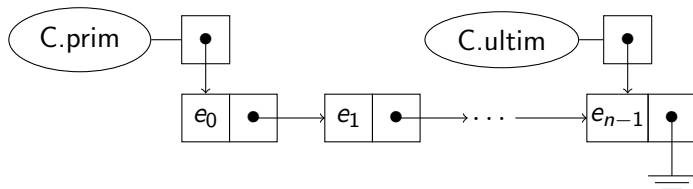
► Implementarea operațiilor

► `insereaza()`

```
procedure insereaza(C, e)  
begin  
    if (C.ultim + 1)%Max == C.prim then  
        throw "eroare"  
    else  
        C.ultim  $\leftarrow$  (C.ultim + 1)%Max  
        C.tab[ultim]  $\leftarrow$  e  
end
```

Coadă – implementarea cu structuri înlănțuite

► Reprezentarea obiectelor



Coadă – implementarea cu structuri înlănțuite

► Implementarea operațiilor

► `insereaza()`

```
procedure insereaza(C, e)  
begin  
    new(q)  
    q → elt ← e  
    q → succ ← NULL  
    if C.ultim == NULL then  
        C.prim ← q  
        C.ultim ← q  
    else  
        C.ultim → succ ← q  
        C.ultim ← q  
end
```

Tipurile abstracte LLin, LLinOrd, Stiva, Coadă

- Liste liniare

- Implementarea cu tablouri

- Implementarea cu liste simplu înlănțuite

- Liste liniare ordonate

- Stive

- Cozi

Aplicație la conversie de expresii aritmetice

Aplicație – notația postfixată a expresiilor

- ▶ notația infixată

- ▶ $a + b$
- ▶ $a + (b * 2)$

- ▶ notația postfixată

- ▶ $a \ b \ +$
- ▶ $a \ b \ 2 \ * \ +$

- ▶ reguli de precedență

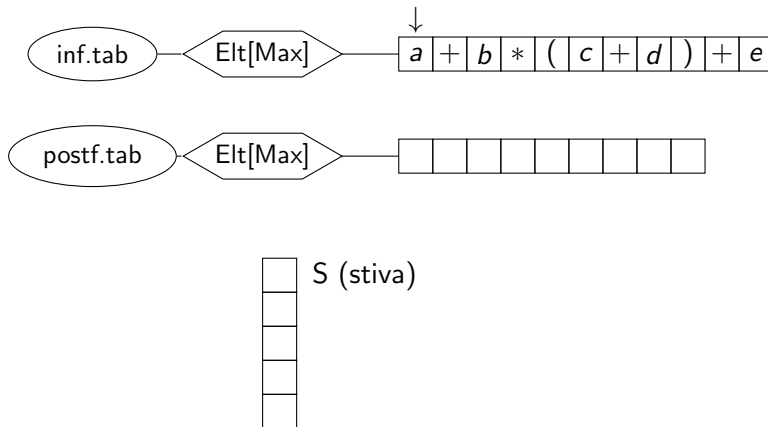
- ▶ $a + b * 2$

- ▶ reguli de asociere: $7/3 * 2$

- ▶ la stânga: $(7/3) * 2$
- ▶ la dreapta: $7/(3 * 2)$

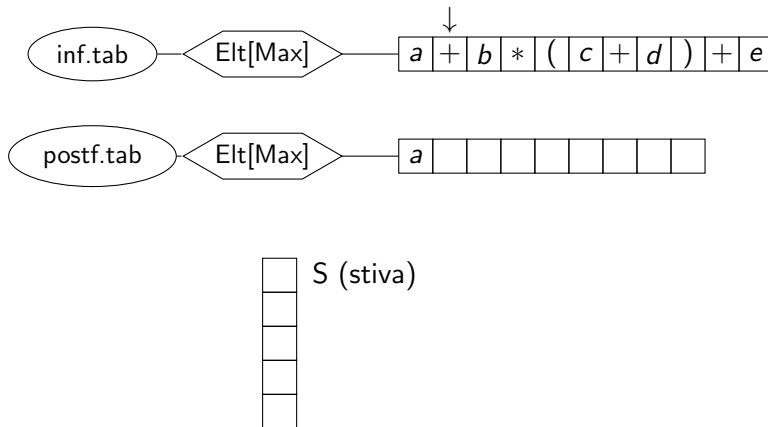
Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



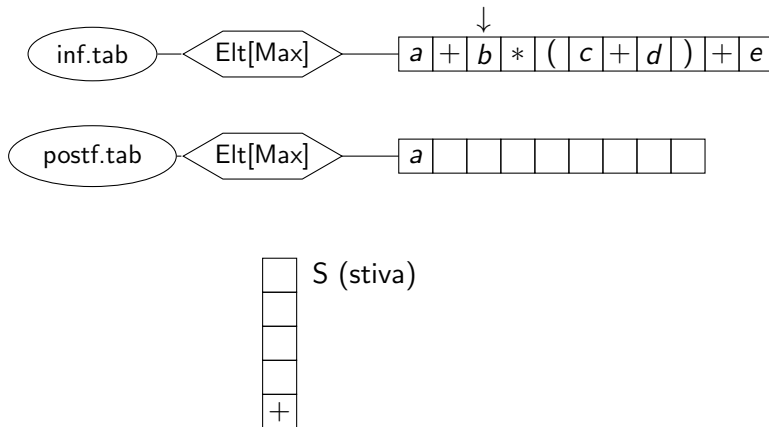
Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



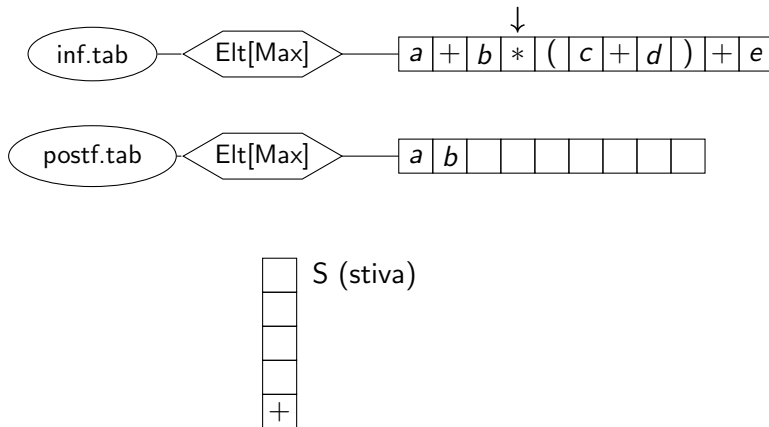
Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



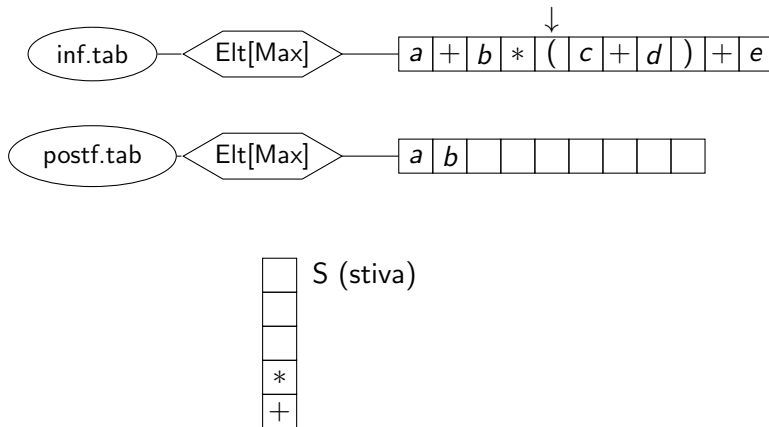
Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



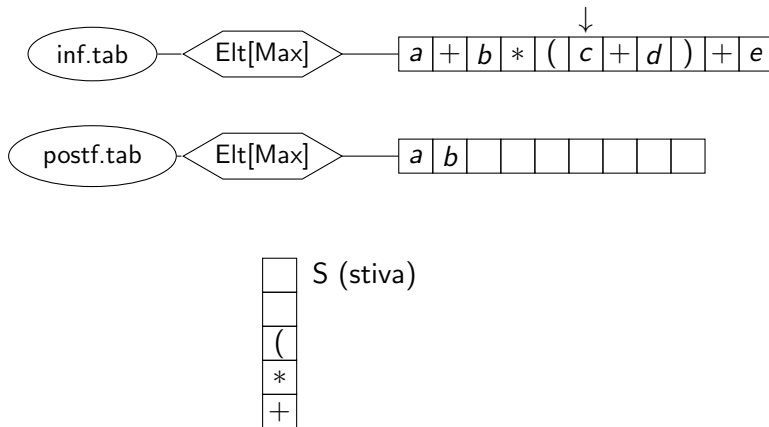
Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



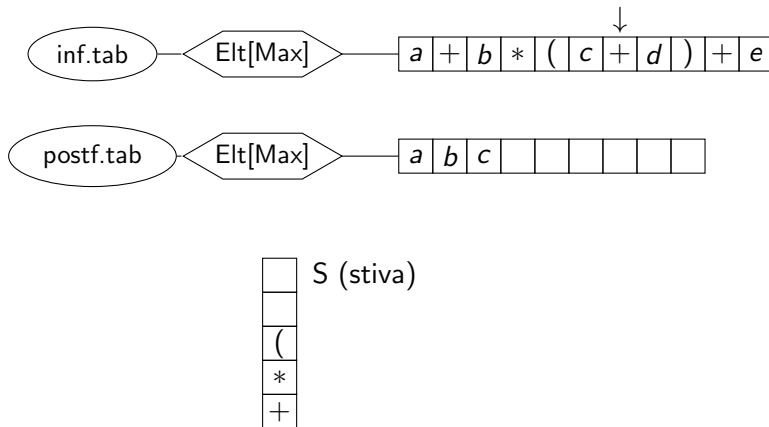
Conversia infixat \rightarrow postfixat

Exemplu: $a + b * (c + d) + e$



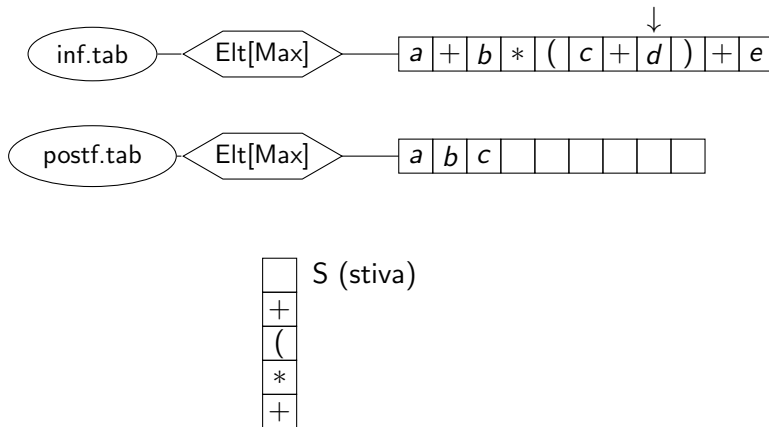
Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



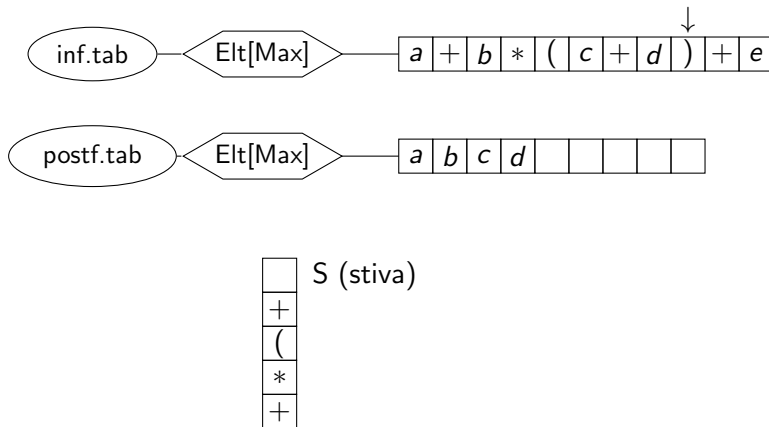
Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



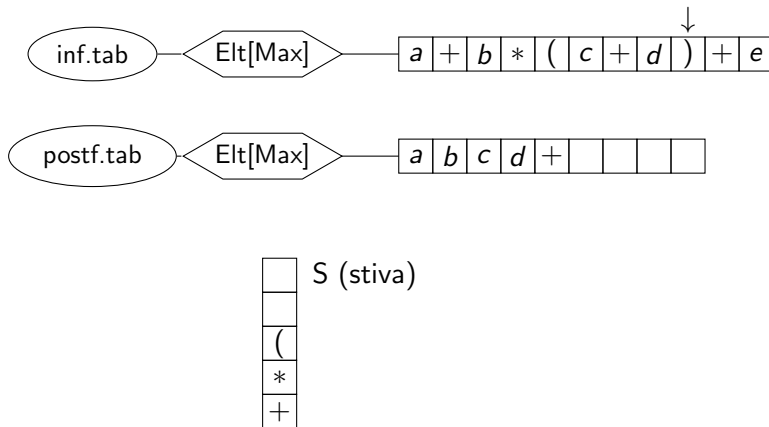
Conversia infixat \rightarrow postfixat

Exemplu: $a + b * (c + d) + e$



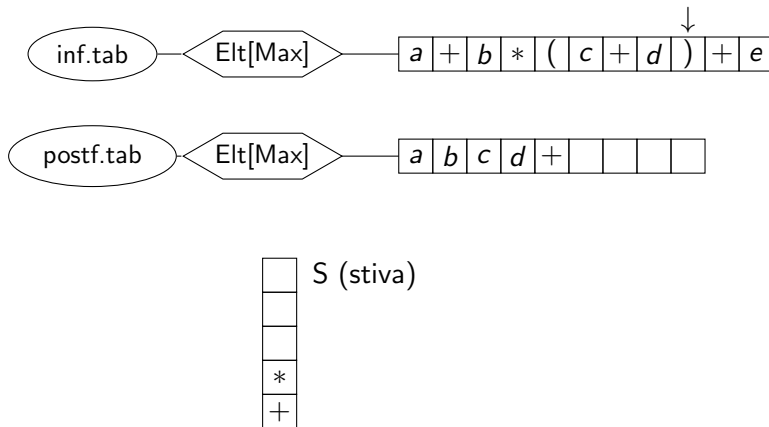
Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



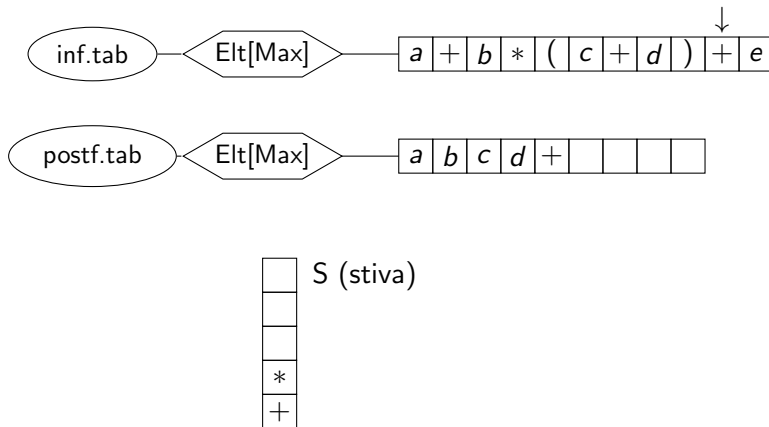
Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



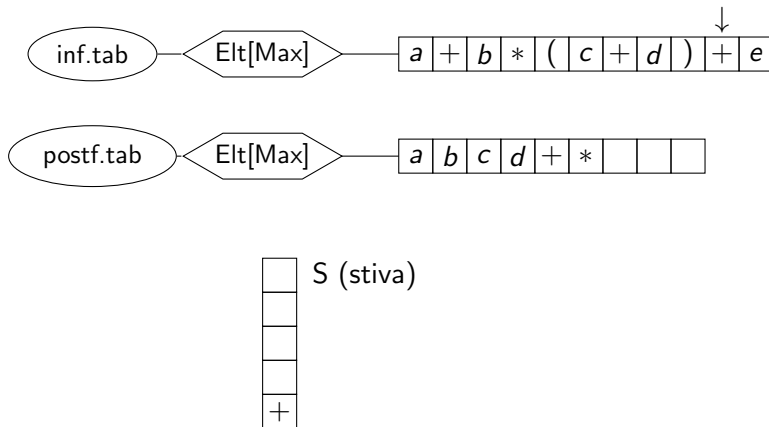
Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



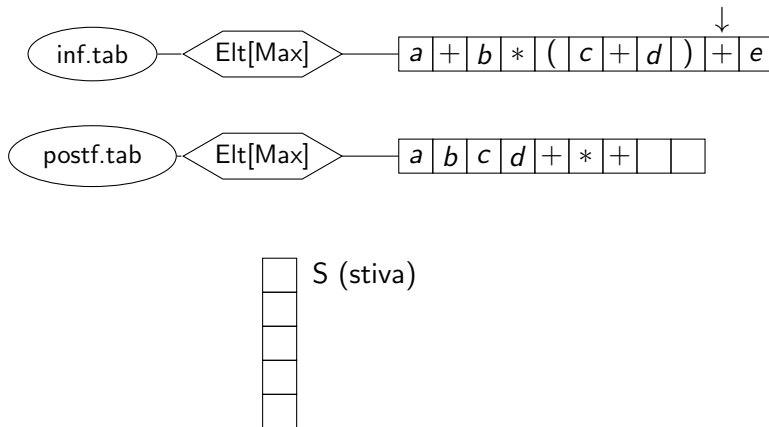
Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



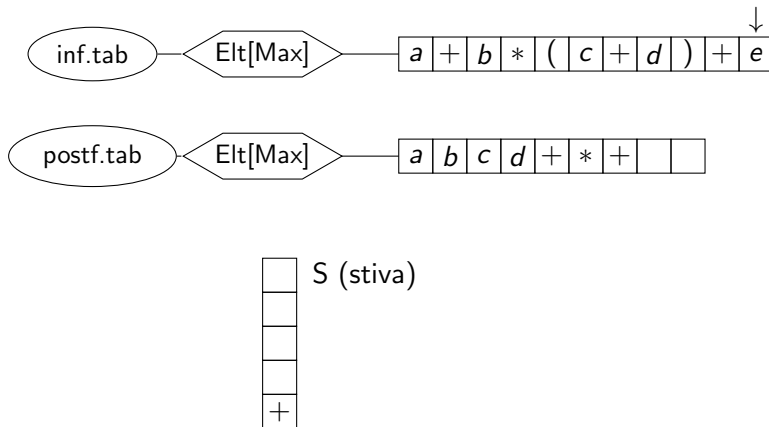
Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



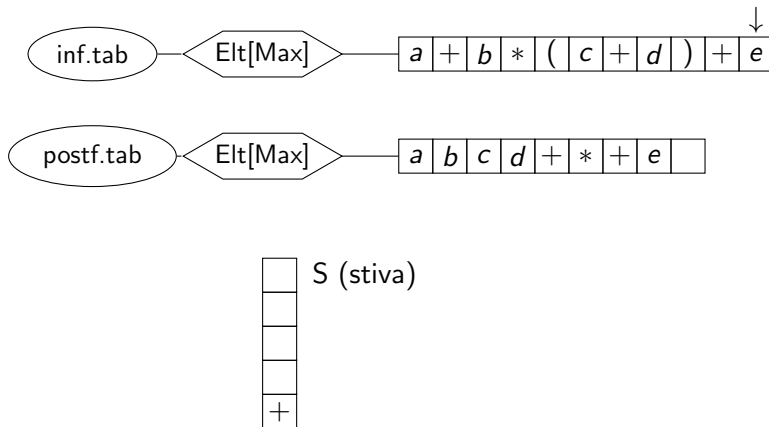
Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



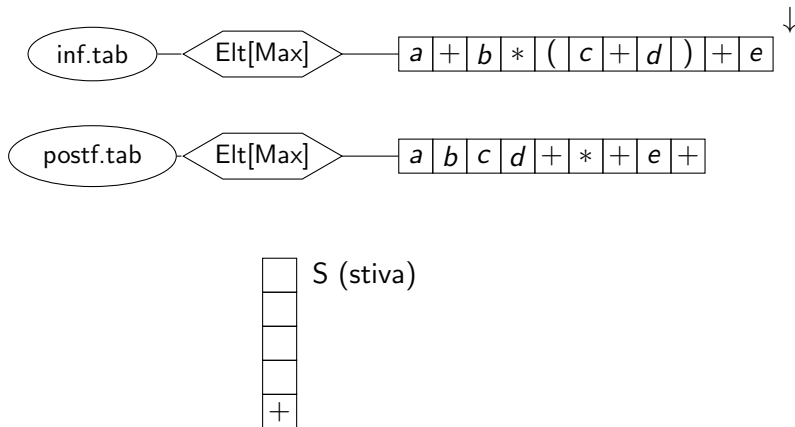
Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



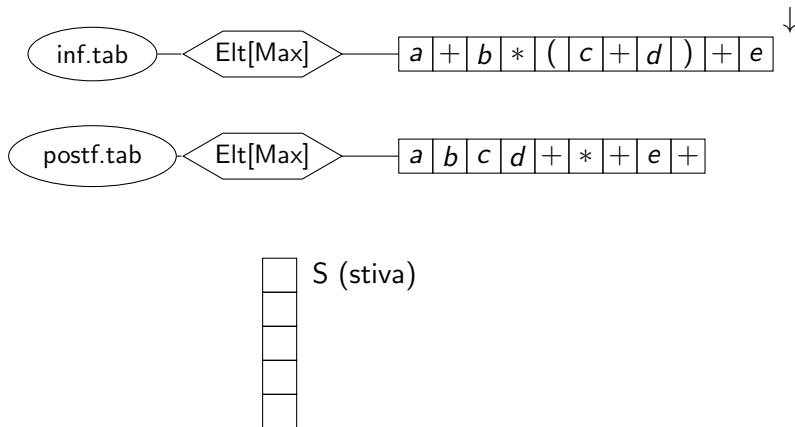
Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



Conversia infixat \rightarrow postixat

Exemplu: $a + b * (c + d) + e$



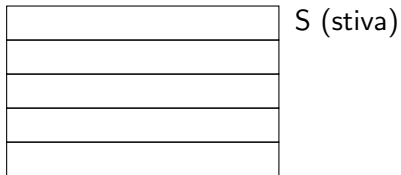
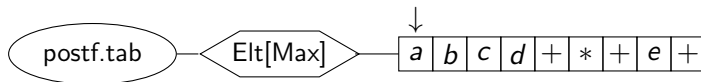
Conversia infixat → postfixat

```
procedure convInfix2Postfix(infix, postfix)
/* infix și postfix sunt cozi */
begin
    S ← stivaVida()
    while (not esteVida(infix)) do
        x ← citește(infix);   elimina(infix)
        if (operand(x)) then
            insereaza(postfix, x)
        else
            if (x == '(') then
                while (top(S) != '(') do
                    insereaza(postfix, top(S));   pop(S)
                pop(S)
            else
                while (not esteVida(S) and top(S) != '(' and
                    priorit(top(S)) >= priorit(x)) do
                    insereaza(postfix, top(S));   pop(S)
                push(S, x)
    while (not esteVida(S)) do
        insereaza(postfix, top(S));   pop(S)
```

end

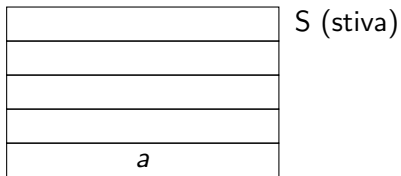
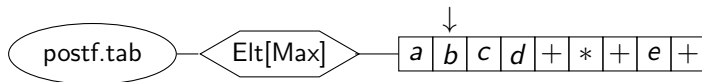
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



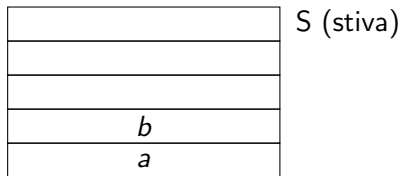
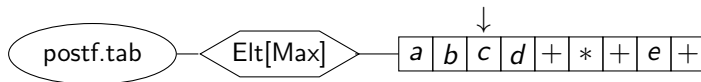
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



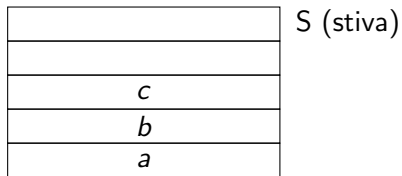
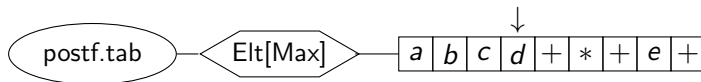
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



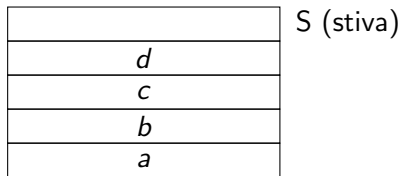
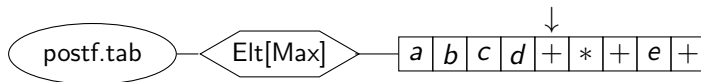
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



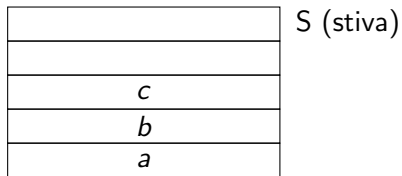
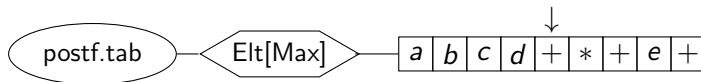
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



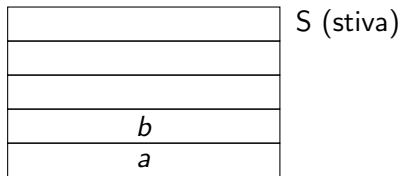
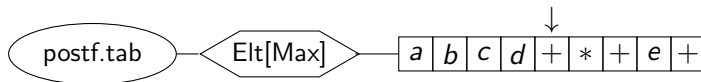
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



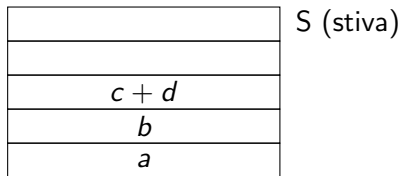
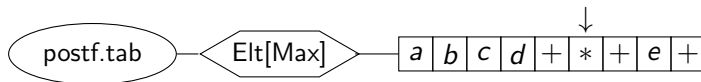
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



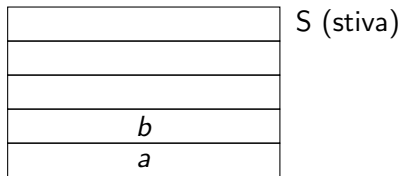
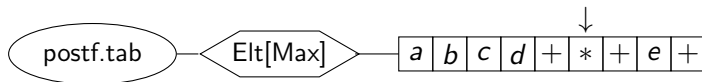
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



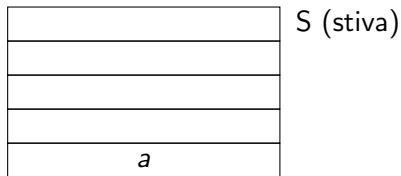
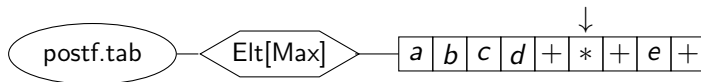
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



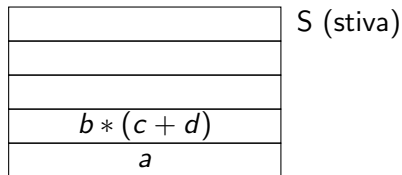
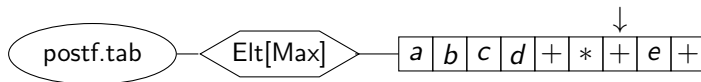
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



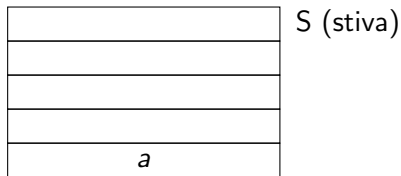
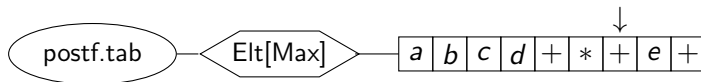
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



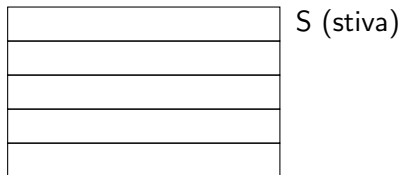
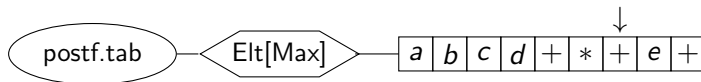
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



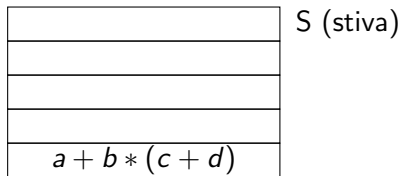
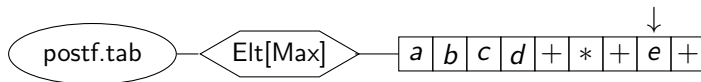
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



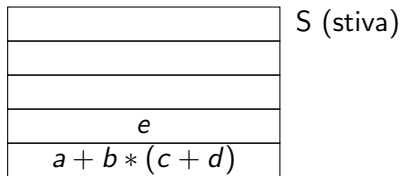
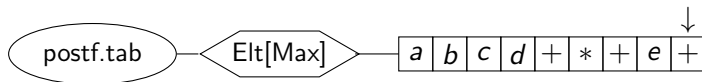
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



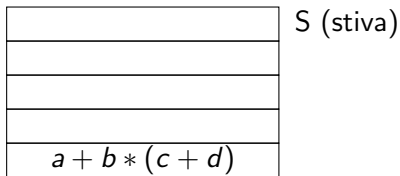
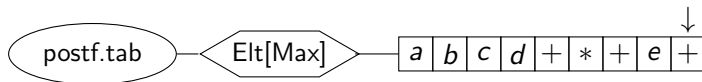
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



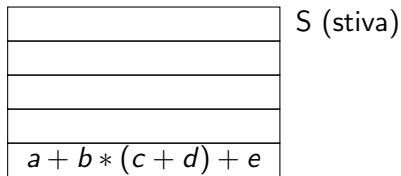
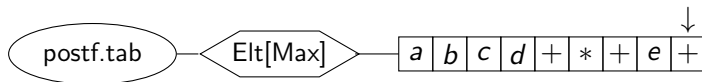
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



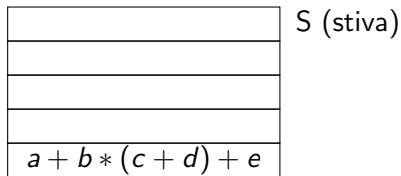
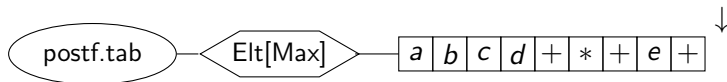
Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



Evaluarea expresiilor postfixate

Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$



Evaluarea expresiilor postfixate

```
function valPostfix(postfix)  
begin  
     $S \leftarrow \text{stivaVida}()$   
    while (not esteVida(postfix)) do  
         $x \leftarrow \text{citeste}(\text{postfix}); \quad \text{elimina}(\text{infix})$   
        if (operand( $x$ ) then  
             $\text{push}(S, x)$   
        else  
             $\text{drp} \leftarrow \text{top}(S); \quad \text{pop}(S)$   
             $\text{stg} \leftarrow \text{top}(S); \quad \text{pop}(S)$   
             $\text{val} \leftarrow \text{stg op}(x) \text{ drp}$   
             $\text{push}(S, \text{val})$   
     $\text{val} = \text{top}(S); \quad \text{pop}(S)$   
    return val  
end
```