# Liste. Stive. Cozi

SD 2017/2018

# Conținut

# Conținut

# 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: $L = (e_0, \cdots, e_{n-1}), \quad n \geq 0$
- $e_i \in$ `Elt` (tipul abstract al elementelor)
- Relații:
  - $e_0$ primul element al listei;
  - $e_{n-1}$ ultimul element al listei;
  - $e_i$ elementul predecesor lui $e_{i+1}$.

# LLin – operații

- `listaVida()`
  - intrare: nimic
  - ieșire: $L = ()$ (lista cu zero elemente)

- `insereaza()`
  - intrare:
    - $L = (e_0, \cdots, e_{n-1}), \quad k \in \texttt{Nat}, \quad e \in \texttt{Elt}$
  - ieșire:
    - $L = (\cdots, e_{k-1}, e, e_k, \cdots)$, dacă $0 \leq k \leq n$
    - eroare în caz contrar

# insereaza() – exemple

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

- insereaza$(L, 0, x)$ $\Rightarrow$ $L = (x, a, b, c, d, e, f, g)$

  Obs. indexul elementelor $a, \cdots, g$ crește cu 1.

- insereaza$(L, 2, x)$ $\Rightarrow$ $L = (a, b, x, c, d, e, f, g)$

- insereaza$(L, 7, x)$ $\Rightarrow$ $L = (a, b, c, d, e, f, g, x)$

- insereaza$(L, 10, x)$ $\Rightarrow$ eroare

- insereaza$(L, -7, x)$ $\Rightarrow$ eroare

# LLin – operații

- `elimina()`
  - intrare:
    - $L = (e_0, \cdots, e_{n-1}), \quad k \in \texttt{Nat}$
  - ieșire:
    - $L = (\cdots, e_{k-1}, e_{k+1}, \cdots)$, dacă $0 \leq k \leq n-1$
    - eroare în caz contrar

# LLin – operații

- `elimina()`
  - intrare:
    - $L = (e_0, \cdots, e_{n-1}), \quad k \in \texttt{Nat}$
  - ieșire:
    - $L = (\cdots, e_{k-1}, e_{k+1}, \cdots)$, dacă $0 \leq k \leq n-1$
    - eroare în caz contrar

---

Exemple:

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

- $\texttt{elimina}(L, 2) \quad \Rightarrow \quad L = (a, b, d, e, f, g)$
  Obs. indexul elementelor $d, \cdots, g$ descrește cu 1.
- $\texttt{elimina}(L, 10) \quad \Rightarrow \quad$ eroare
- $\texttt{elimina}(L, -7) \quad \Rightarrow \quad$ eroare

# LLin – operații

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

# LLin – operații

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

---

Exemple:

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

- $\texttt{alKlea}(L, 0) \quad \Rightarrow \quad a$
- $\texttt{alKlea}(L, 2) \quad \Rightarrow \quad c$
- $\texttt{alKlea}(L, 6) \quad \Rightarrow \quad g$
- $\texttt{alKlea}(L, 20) \quad \Rightarrow \quad$ eroare
- $\texttt{alKlea}(L, -2) \quad \Rightarrow \quad$ eroare

# LLin – operații

- `elimTotE()`
  - intrare:
    - $L = (e_0, \cdots, e_{n-1}), \quad e \in \texttt{Elt}$
  - ieșire:
    - lista $L$ din care s-au eliminat toate elementele egale cu $e$

# LLin – operații

- `elimTotE()`
  - intrare:
    - $L = (e_0, \cdots, e_{n-1}), \quad e \in \texttt{Elt}$
  - ieșire:
    - lista $L$ din care s-au eliminat toate elementele egale cu $e$

---

Exemple:

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

- $\texttt{elimTotE}(L, a) \quad \Rightarrow \quad (b, c, b, c)$
- $\texttt{elimTotE}(L, c) \quad \Rightarrow \quad (a, b, a, b, a)$
- $\texttt{elimTotE}(L, d) \quad \Rightarrow \quad (a, b, c, a, b, c, a)$

# LLin – operații

- `parcurge()`
  - intrare:
    - $L = (e_0, \cdots, e_{n-1})$, o procedură / funcție `viziteaza()`
  - ieșire:
    - lista $L$ în care toate elementele au fost procesate aplicând `viziteaza()`

# LLin – operații

- `parcurge()`
  - intrare:
    - $L = (e_0, \cdots, e_{n-1})$, o procedură / funcție `viziteaza()`
  - ieșire:
    - lista $L$ în care toate elementele au fost procesate aplicând `viziteaza()`

---

Exemple:

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

- `parcurge`$(L,$ `oriDoi()`$)$ $\Rightarrow$ $(2, 4, 6, 2, 4, 6)$
- `parcurge`$(L,$ `incrementeaza()`$)$ $\Rightarrow$ $(2, 3, 4, 2, 3, 4)$

# LLin – operații

- `poz()`
  - intrare:
    - $L = (e_0, \cdots, e_{n-1}), \quad e \in \texttt{Elt}$,
  - ieșire:
    - prima poziție pe care apare $e$ în $L$ sau
    - $-1$ dacă $e$ nu apare în $L$.

# LLin – operații

- `poz()`
  - intrare:
    - $L = (e_0, \cdots, e_{n-1}), \quad e \in \texttt{Elt}$,
  - 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:
    - $L = (e_0, \cdots, e_{n-1})$,
  - ieșire:
    - $n$ – lungimea listei $L$.

# LLin – operații

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

---

Exemple:

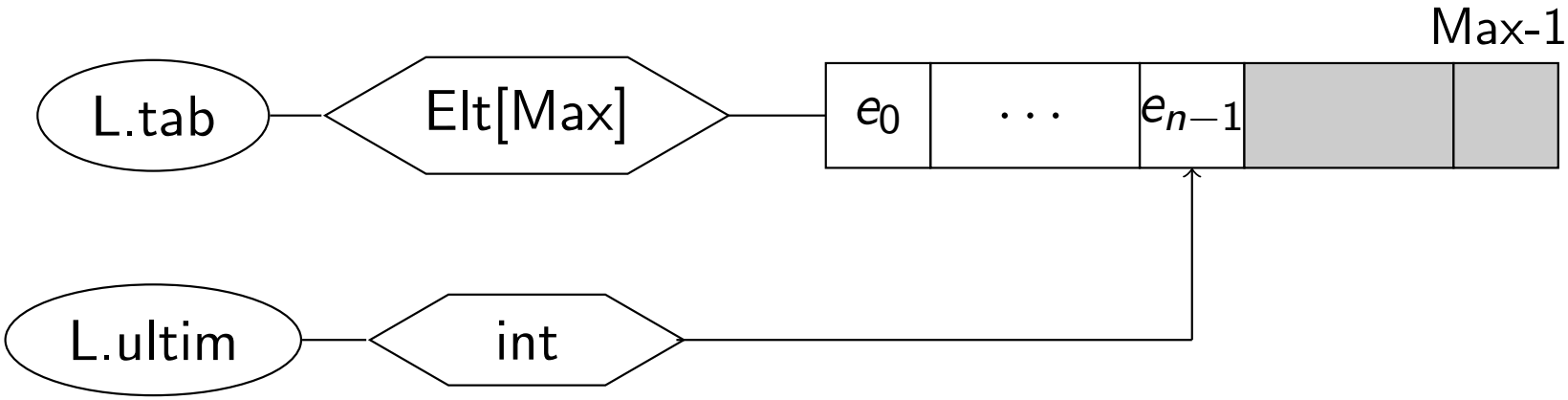
$$L = (a, b, c, a, b, c, d)$$

- `lung`$(L) \quad \Rightarrow \quad 7$

# Conținut

# LLin – implementarea cu tablouri

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




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

# LLin – implementarea cu tablouri

- `insereaza()`
  - deplasează elementele de pe pozițiile $k, k+1, \ldots$ 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** $(L.ultim >= Max - 1)$ **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: $O(n)$.

# LLin – implementarea cu tablouri

- `parcurge()`

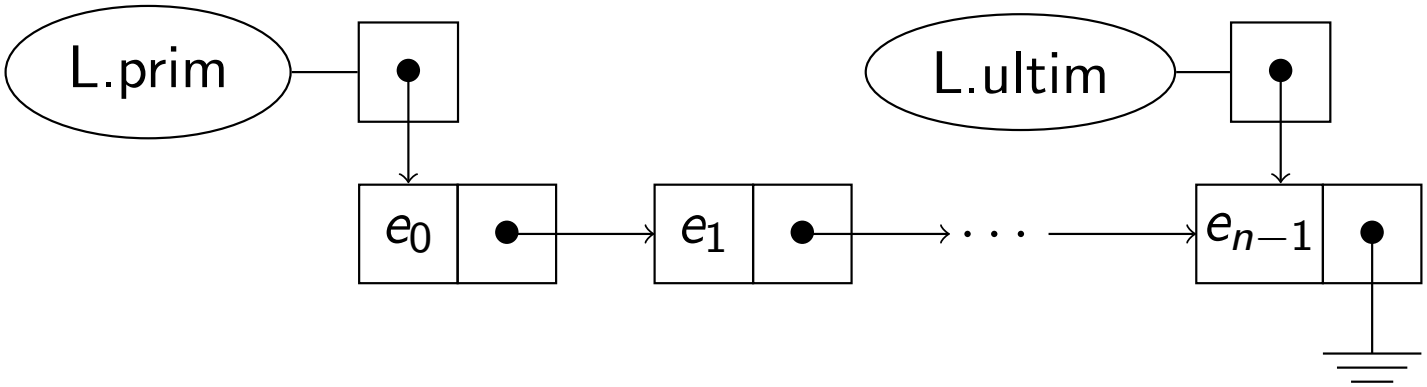
**procedure** *parcurge(L, viziteaza())*
**begin**
    **for** $i \leftarrow 0$ **to** $L.ultim$ **do**
        viziteaza($L.tab[i]$)
**end**

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

# Conținut

# LLin – implementarea cu structuri înlănțuite

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



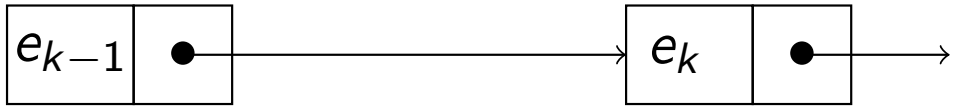

- $L$ este o *structură* cu două câmpuri
  - $L.prim$ – pointer la primul element al listei;
  - $L.ultim$ – pointer la ultimul element al listei.
- un nod $*$ p (aflat la adresa din p) are două câmpuri:
  - $p->elt(= e_i)$ – memorează informația utilă din nod;
  - $p->succ$ – memorează adresa nodului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

# LLin – implementarea cu structuri înlănțuite

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

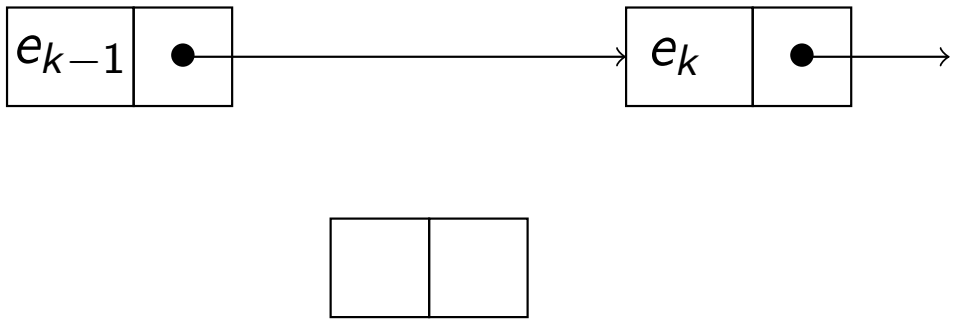
# LLin – implementarea cu structuri înlănțuite

- Cazul general



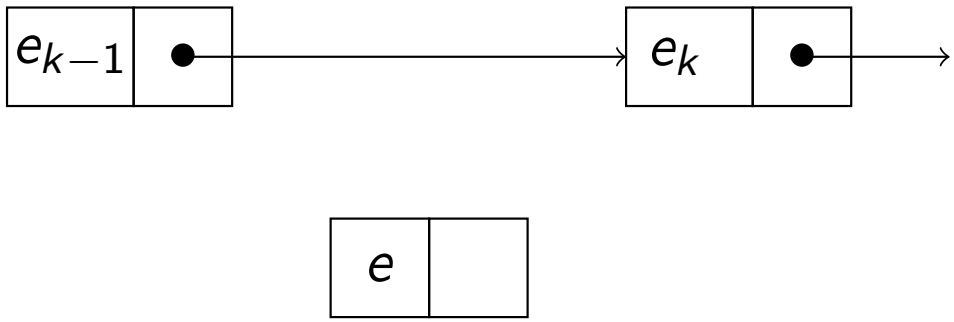
# LLin – implementarea cu structuri înlănțuite

- Cazul general



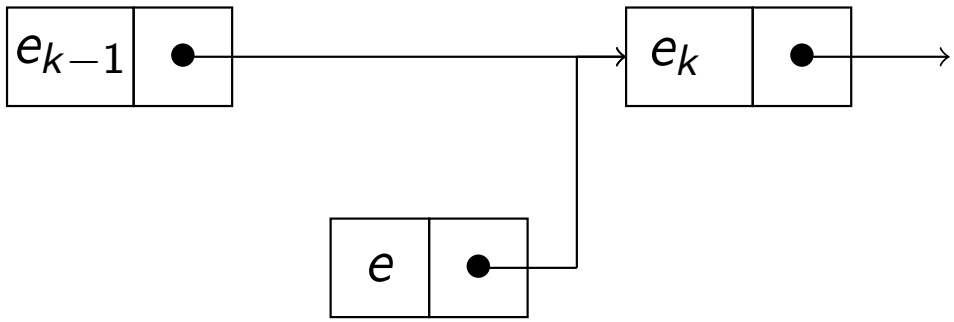
# LLin – implementarea cu structuri înlănțuite

- Cazul general



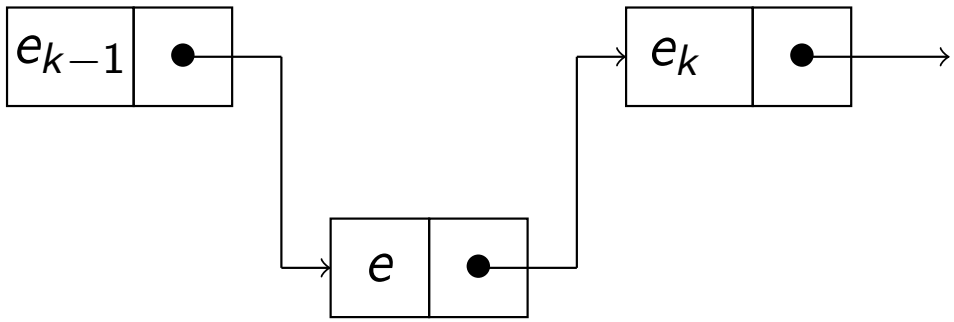
# LLin – implementarea cu structuri înlănțuite

- Cazul general



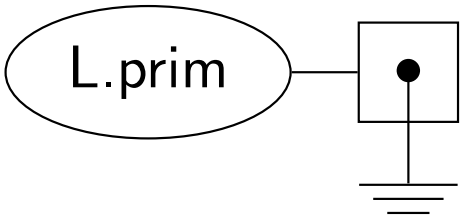
# LLin – implementarea cu structuri înlănțuite

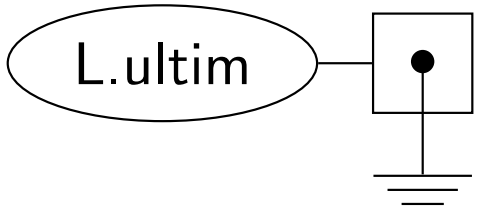
- Cazul general



# LLin – implementarea cu structuri înlănțuite

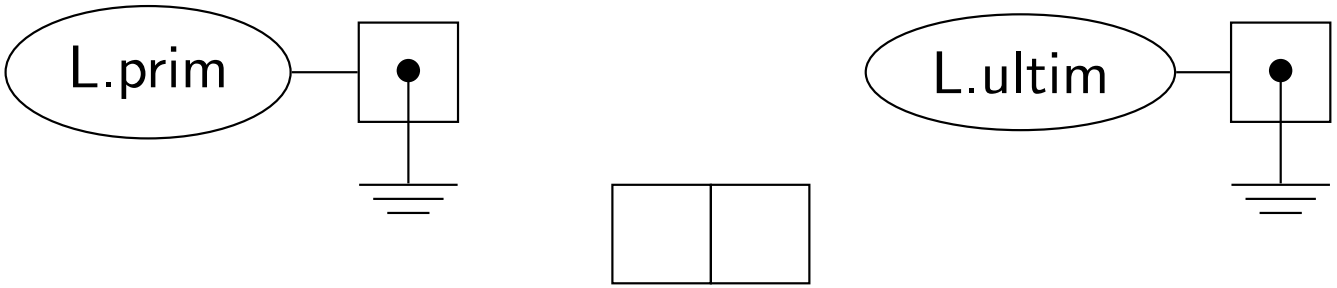
- Cazul particular: lista vidă



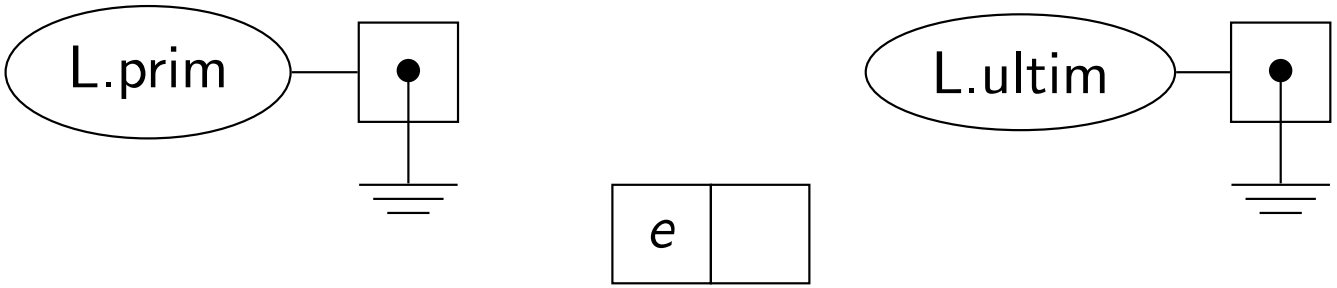
# LLin – implementarea cu structuri înlănțuite

- Cazul particular: lista vidă



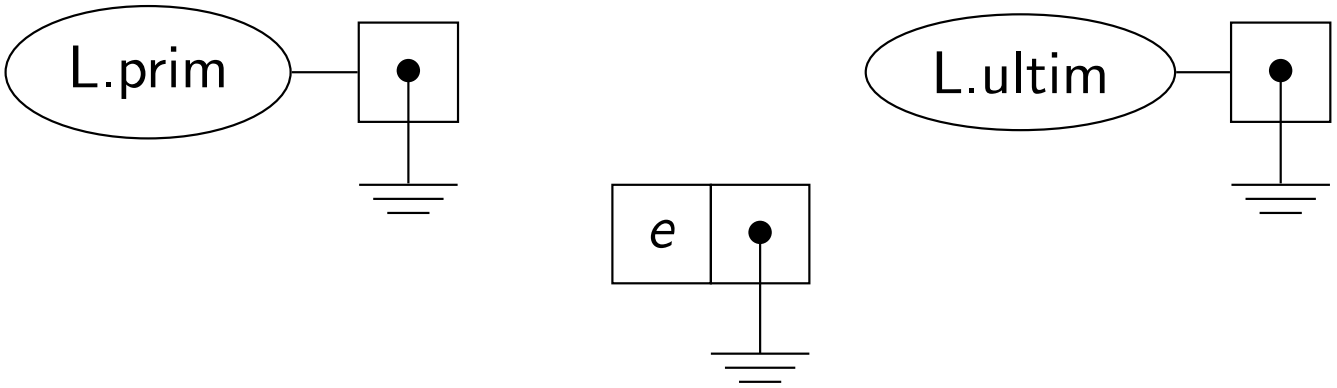
# LLin – implementarea cu structuri înlănțuite

- Cazul particular: lista vidă



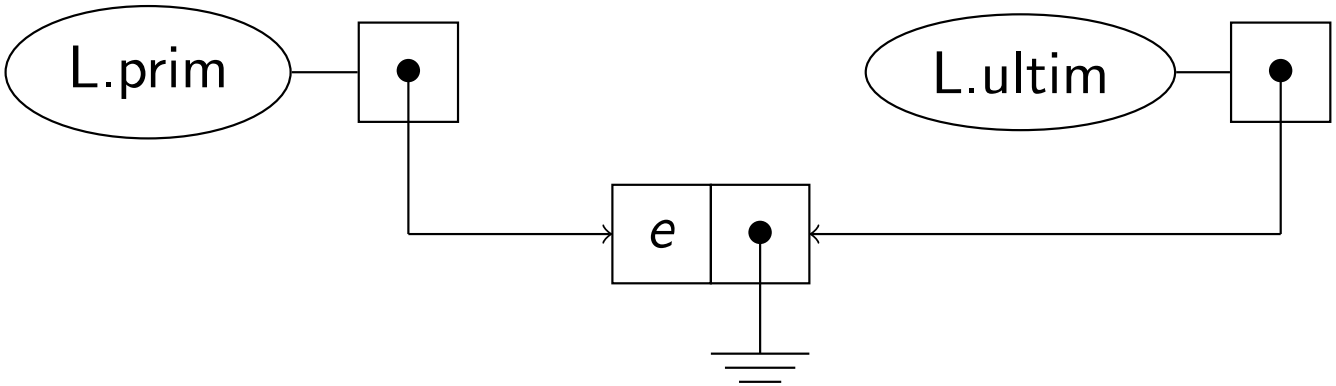
# LLin – implementarea cu structuri înlănțuite

- Cazul particular: lista vidă



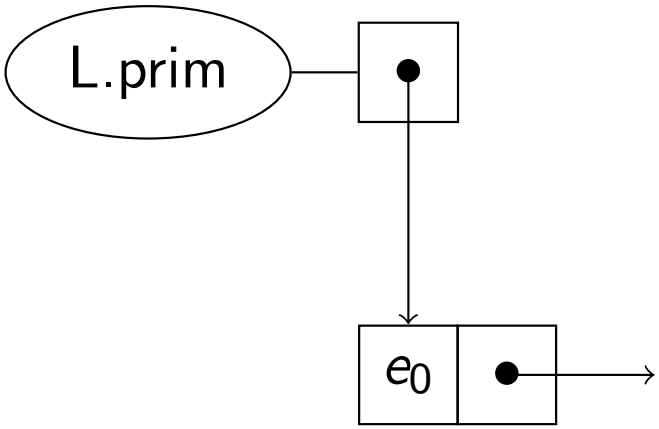
# LLin – implementarea cu structuri înlănțuite

- Cazul particular: lista vidă



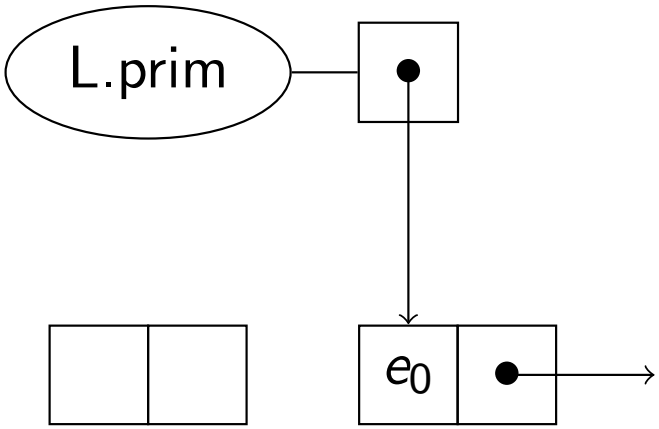
# LLin – implementarea cu structuri înlănțuite

- Cazul particular: inserarea la începutul listei



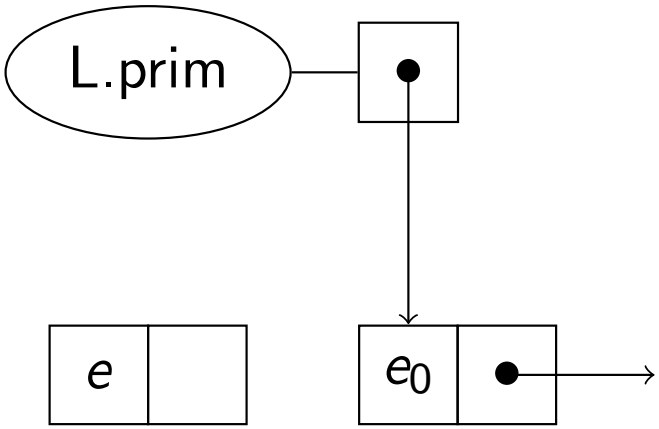
# LLin – implementarea cu structuri înlănțuite

- Cazul particular: inserarea la începutul listei



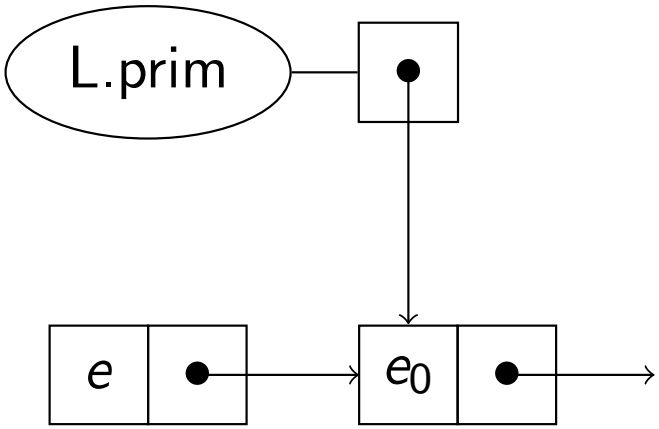
# LLin – implementarea cu structuri înlănțuite

- Cazul particular: inserarea la începutul listei



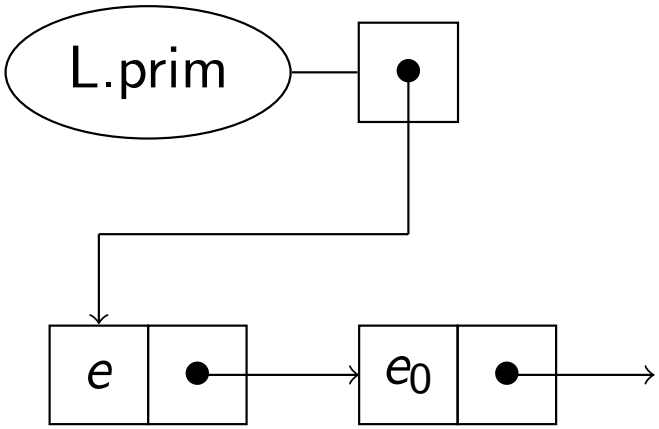
# LLin – implementarea cu structuri înlănțuite

- 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) 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
```

# LLin – aplicație

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

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

- Obs. Timpul de execuție depinde de implementare.

# LLin – aplicație

- 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**

# LLin – aplicație

- 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**

# Conținut

# Liste liniare ordonate: `LLinOrd`

- OBIECTE:
  $L = (e_0, \cdots, e_{n-1}), \quad n \geq 0, \quad e_i \in \texttt{Elt}, \quad e_0 \leq e_1 \leq \cdots \leq e_{n-1}$

- Operații:
  - `listaVida()`
    - intrare: nimic
    - ieșire: $L = ()$ (lista cu zero elemente)
  - `insereaza()`
    - intrare: $L = (e_0, \ldots, e_{n-1}), \quad e \in \texttt{Elt}$
    - ieșire: $L = (\cdots, e_{k-1}, e, e_k, \cdots)$, dacă $e_{k-1} \leq e \leq e_k$
      $(e_{-1} = -\infty, e_n = +\infty)$

# Liste liniare ordonate: `LLinOrd`

- `elimina()`
  - intrare: $L = (e_0, \ldots, e_{n-1}), \quad e \in$ `Elt`
  - ieșire: $L = (\cdots, e_{k-1}, e_{k+1}, \cdots)$, dacă $e = e_k$
    eroare în caz contrar
- `alKlea()`
- `parcurge()`
- `poz()`

# LLinOrd – implementarea cu tablouri

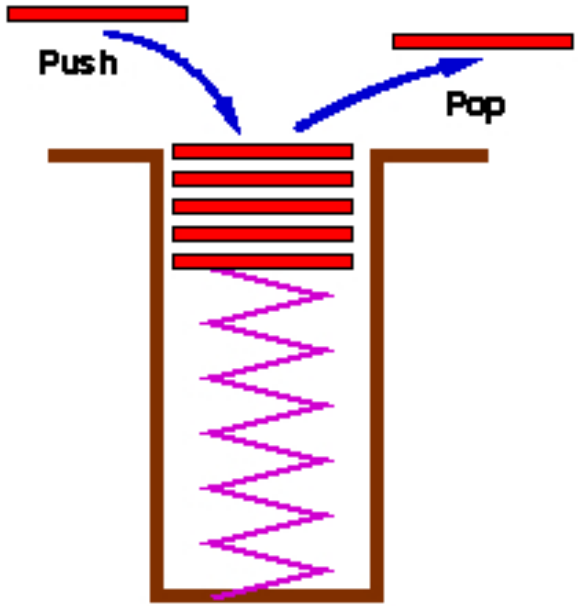
```
function poz(L, e)
begin
    p ← 0;    q ← L.ultim
    m ← (p + q)/2
    while (L.tab[m]! = e and p < q) do
        if (e < L.tab[m]) then
            q ← m–1
        else
            p ← m + 1
        m ← (p + q)/2
    if (L.tab[m] == e) then
        return m
    else
        return −1
end
```

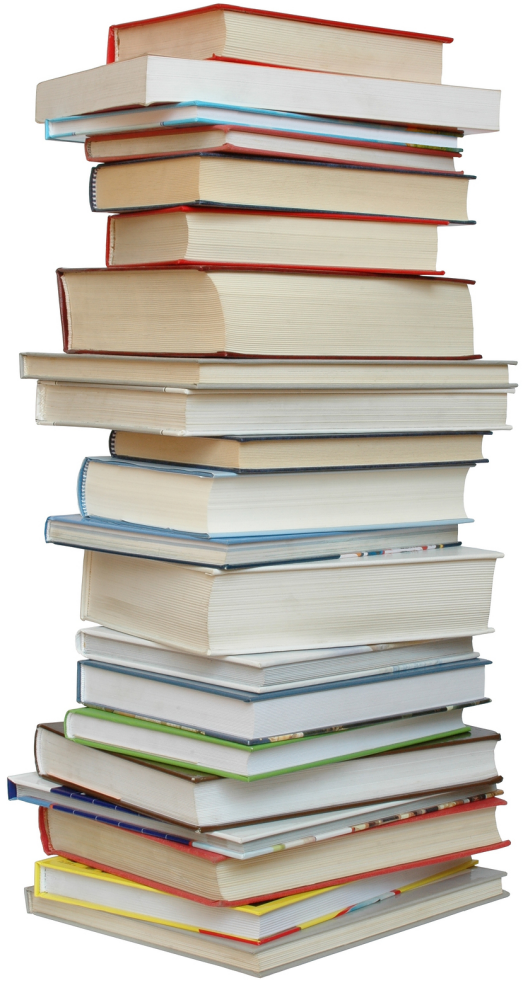
# LLinOrd – complexitatea căutării

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

# Conținut

# Stiva

# Stiva – aplicații

- 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: $S \in$ `Stiva`
    - ieșire:
      – **true** dacă $S$ este vidă;
      – **false** dacă $S$ nu este vidă.

# Tipul abstract `Stiva`

- Operații:
  - `push()`
    - intrare: $S \in$ `Stiva`, $e \in$ `Elt`
    - ieșire: $S$ la care s-a adăugat $e$ ca ultim element introdus (cel cu vechimea cea mai mică).
  - `pop()`
    - intrare: $S \in$ `Stiva`
    - ieșire:
      – $S$ din care s-a eliminat ultimul element introdus (cel cu vechimea cea mai mică);
      – eroare dacă $S$ este vidă.
  - `top()`
    - intrare: $S \in$ `Stiva`
    - ieșire:
      – ultimul element introdus în $S$ (cel cu vechimea cea mai mică);
      – eroare dacă $S$ este vidă.

# Stiva – implementare cu liste

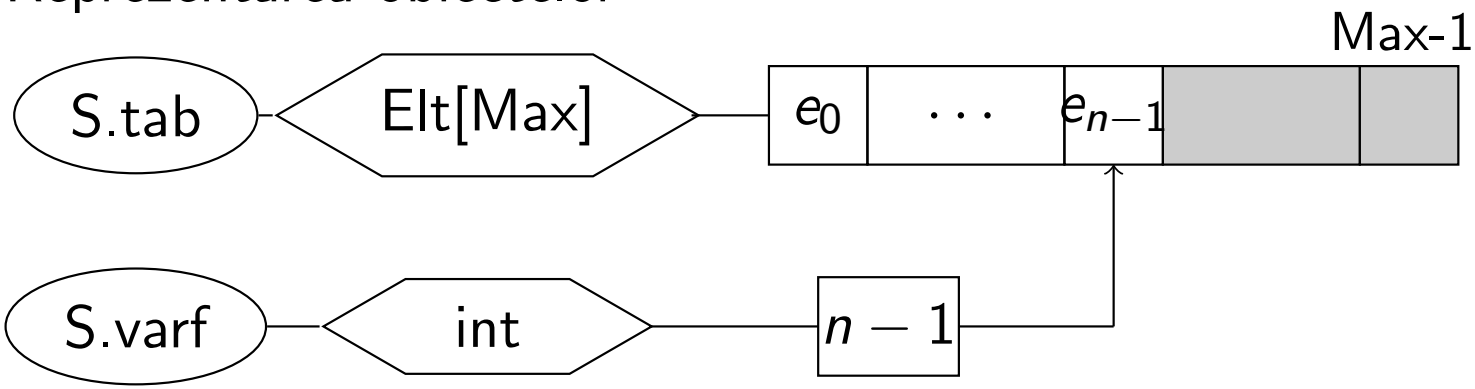
| tipul Stiva | | tipul LLin |
|---|---|---|
| $push(S, e)$ | $=$ | $insereaza(S, 0, e)$ |
| $pop(S, e)$ | $=$ | $elimina(S, 0)$ |
| $top(S)$ | $=$ | $alKlea(S, 0)$ |

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

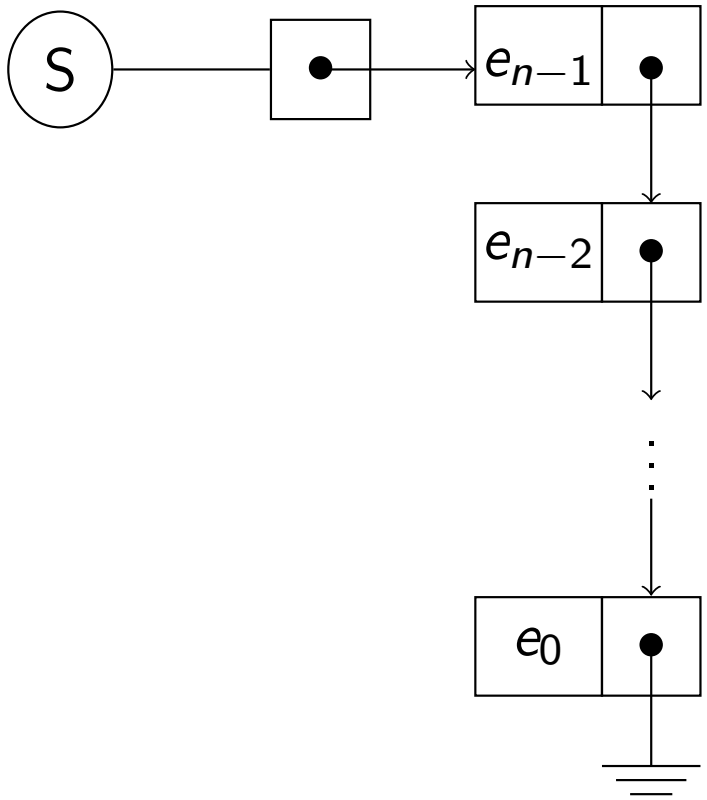



- implementarea operațiilor

**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()`

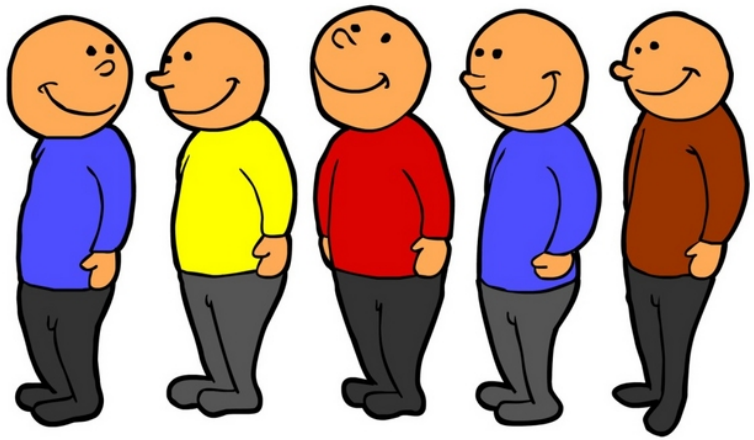
    **procedure** *push*$(S, e)$
    **begin**
    **new**$(q)$
    $q->elt \leftarrow e$
    $q->succ \leftarrow S$
    $S \leftarrow q$
    **end**

  - `pop()`

    **procedure** *pop*$(S)$
    **begin**
    **if** $S == NULL$ **then**
    **throw** "eroare"
    $q \leftarrow S$
    $S \leftarrow S->succ$
    **delete**$(q)$ **end**

# Conținut

# Coada



# Coada – aplicații

- Aplicații directe
  - Liste / fire de așteptare;
  - Accesul la resurse partajate.
    Exemplu: imprimante.
- Aplicații indirecte
  - Structură de date auxiliară în anumiți algoritmi.

# Tipul abstract Coada

- 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: $C \in$ `Coada`
    - ieșire:
      – **true** dacă $C$ este vidă;
      – **false** dacă $C$ nu este vidă.

# Tipul abstract Coada

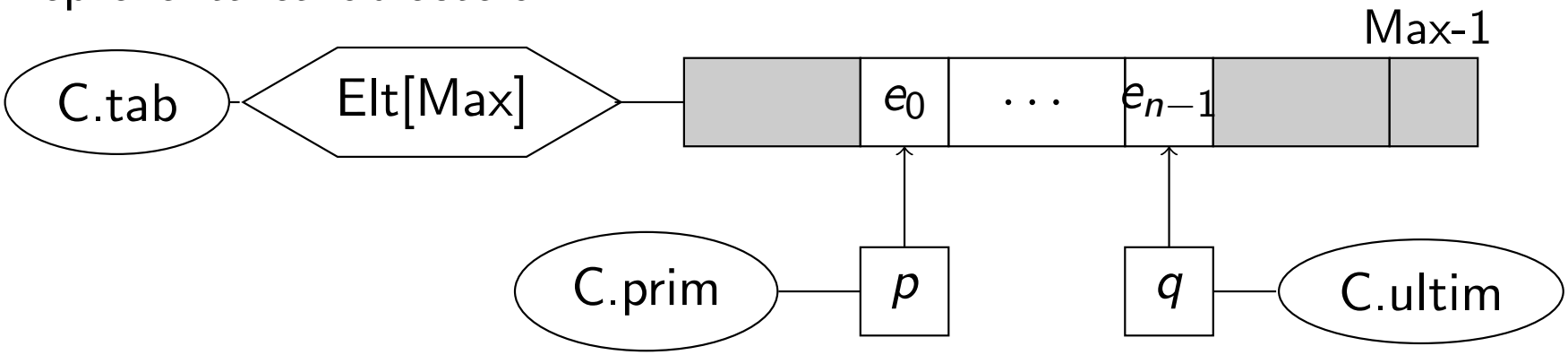
- Operații:
  - `insereaza()`
    - intrare: $C \in$ `Coada`, $e \in$ `Elt`
    - ieșire: $C$ la care s-a adăugat $e$ ca ultim element introdus (cel cu vechimea cea mai mică).
  - `elimina()`
    - intrare: $C \in$ `Coada`
    - ieșire:
      – $C$ din care s-a eliminat primul element introdus (cel cu vechimea cea mai mare);
      – eroare dacă $C$ este vidă.
  - `citeste()`
    - intrare: $C \in$ `Coada`
    - ieșire:
      – primul element introdus în $C$ (cel cu vechimea cea mai mare);
      – eroare dacă $C$ este vidă.

# Coada – implementare cu liste

| tipul Coada | | tipul LLin |
|---|---|---|
| $insereaza(C, e)$ | $=$ | $insereaza(C, lung(C), e)$ |
| $elimina(C)$ | $=$ | $elimina(C, 0)$ |
| $citeste(S)$ | $=$ | $alKlea(C, 0)$ |

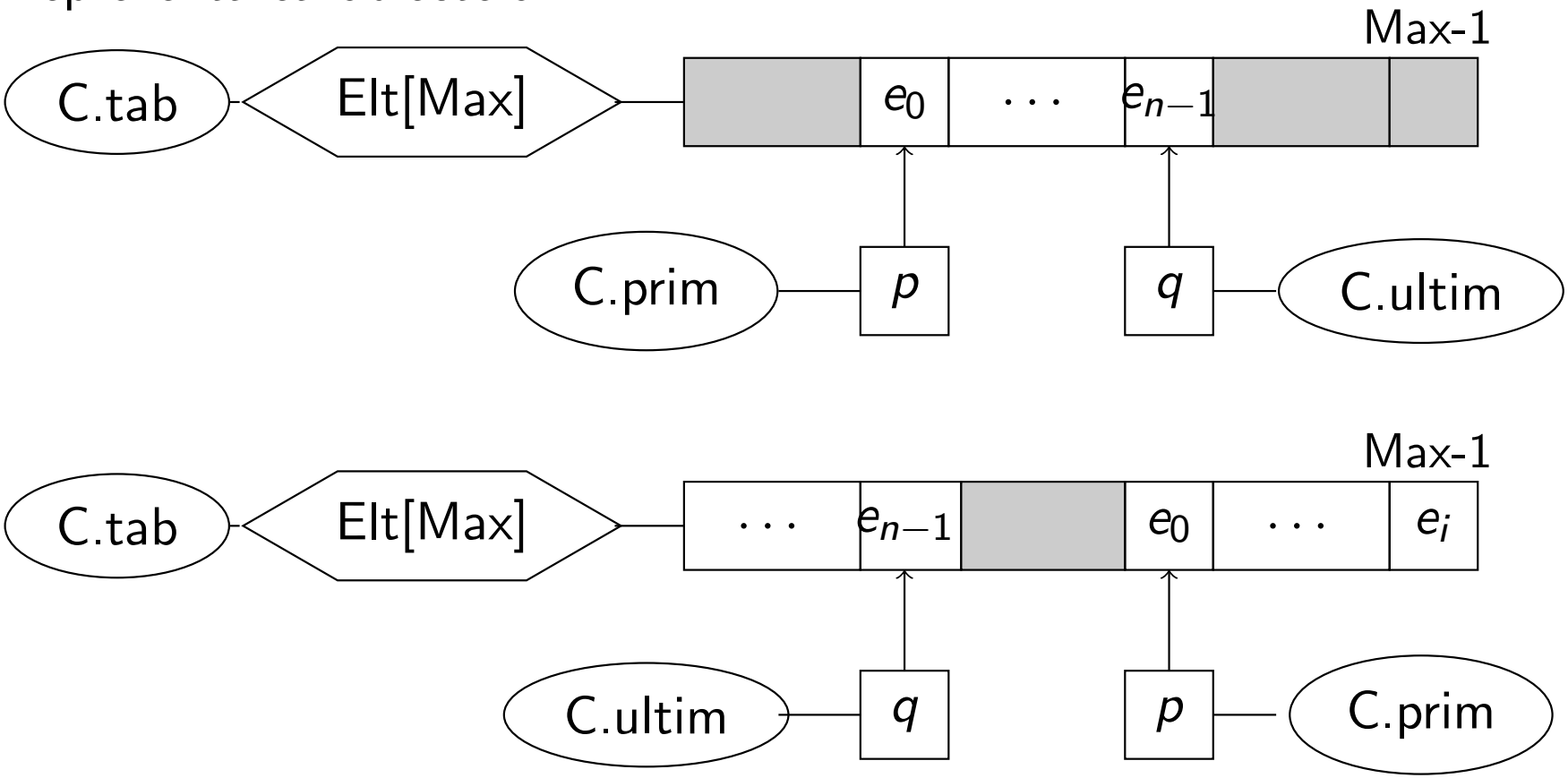
# Coada – implementarea cu tablouri

- Reprezentarea obiectelor



# Coada – implementarea cu tablouri

- Reprezentarea obiectelor



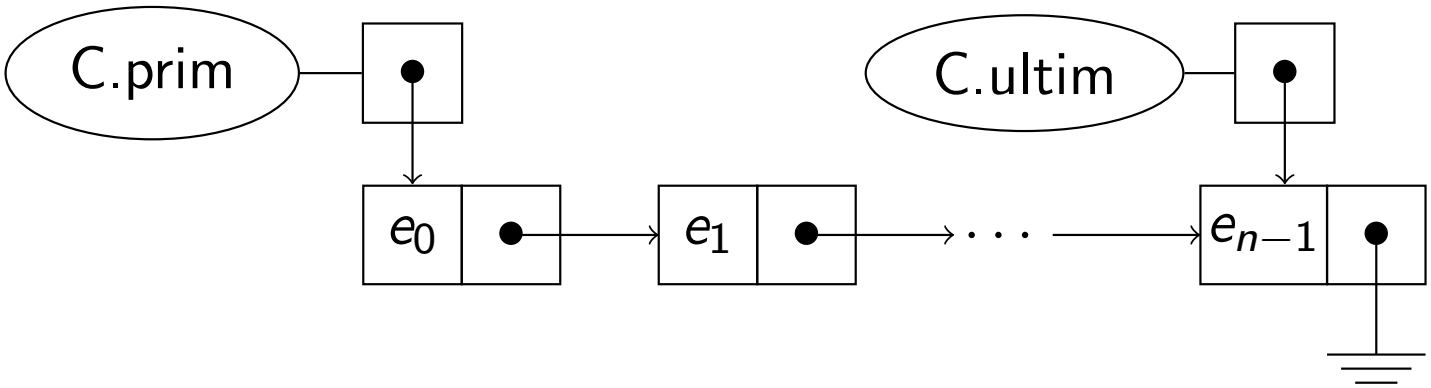
# Coada – implementarea cu tablouri

- 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**

# Coada – implementarea cu structuri înlănțuite

- Reprezentarea obiectelor



# Coada – implementarea cu structuri înlănțuite

- Implementarea operațiilor
  - `insereaza()`

**procedure** *insereaza*$(C, e)$
**begin**
    **new**(q)
    $q->elt \leftarrow e$
    $q->succ \leftarrow NULL$
    **if** $C.ultim == NULL$ **then**
        $C.prim \leftarrow q$
        $C.ultim \leftarrow q$
    **else**
        $C.ultim->succ \leftarrow q$
        $C.ultim \leftarrow q$
**end**

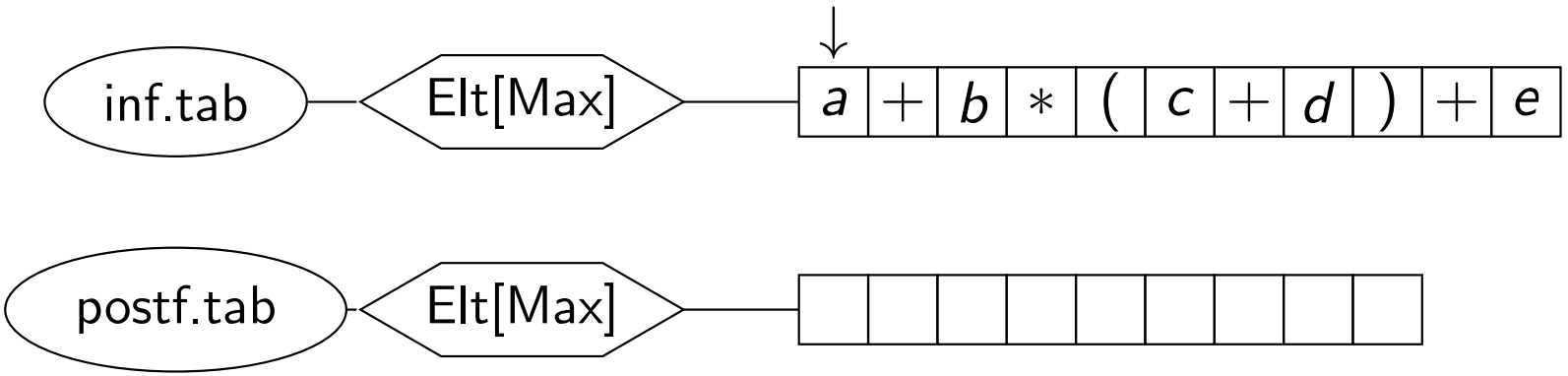
# Conținut

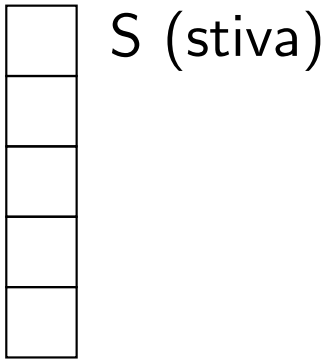
# Aplicație – notația postfixată a expresiilor

- 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

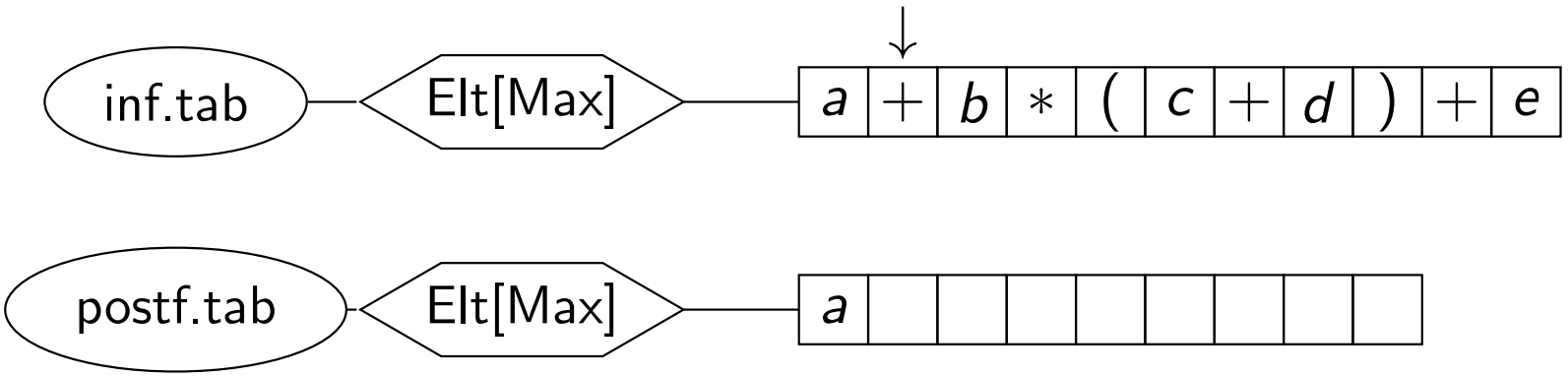
Exemplu: $a + b * (c + d) + e$

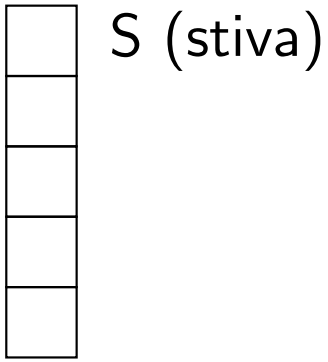


# Conversia infixat $\rightarrow$ postixat

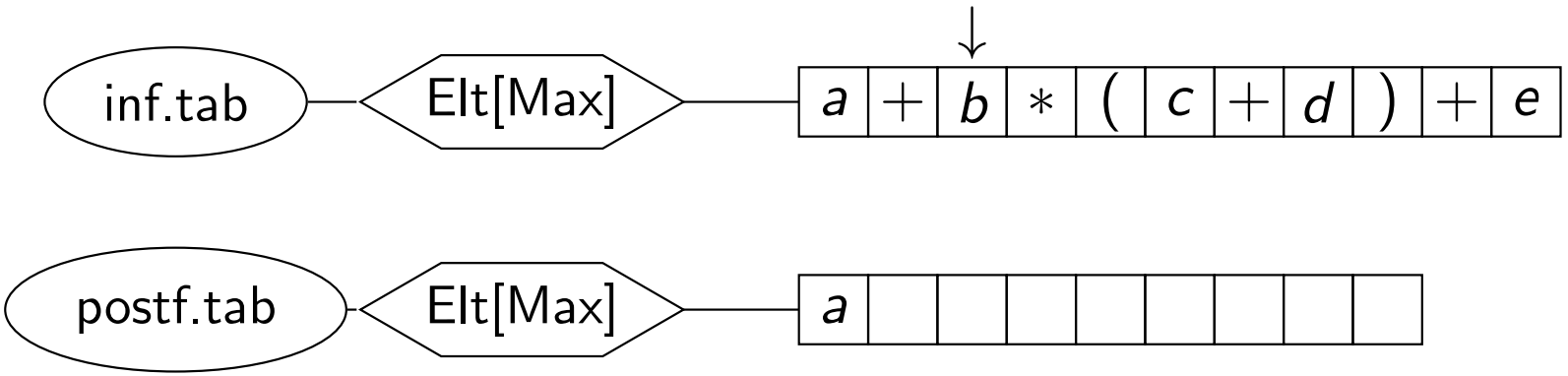
Exemplu: $a + b * (c + d) + e$

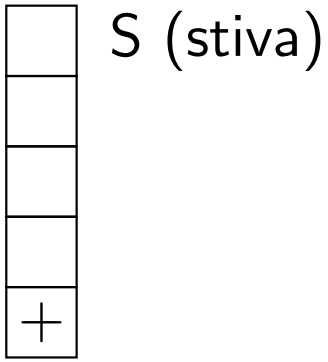


# Conversia infixat $\rightarrow$ postixat

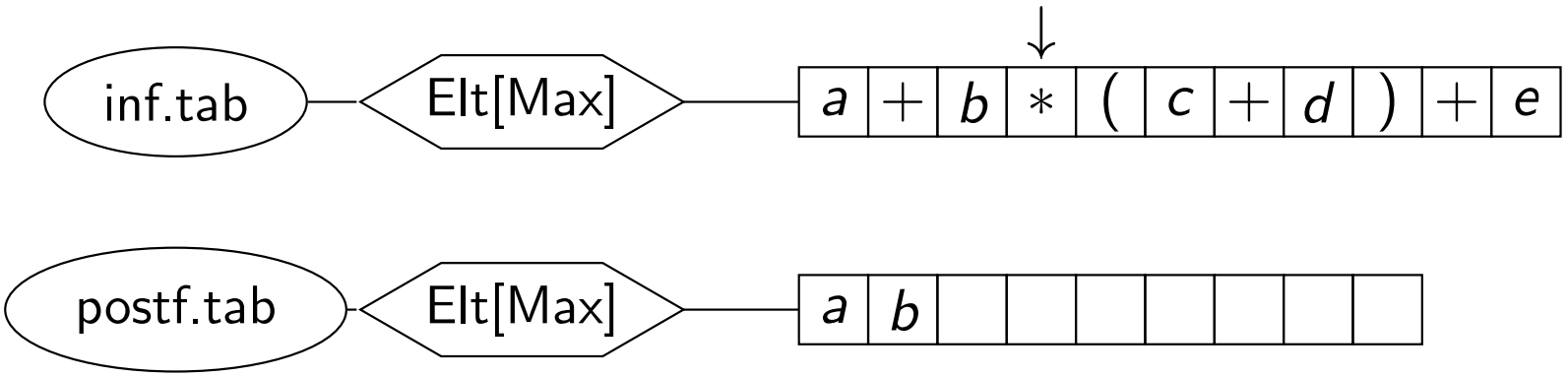
Exemplu: $a + b * (c + d) + e$

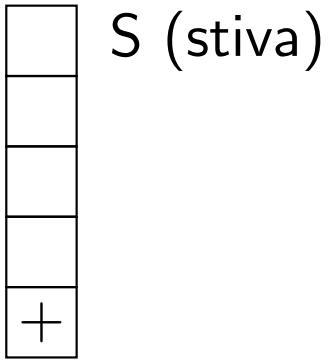


# Conversia infixat $\rightarrow$ postixat

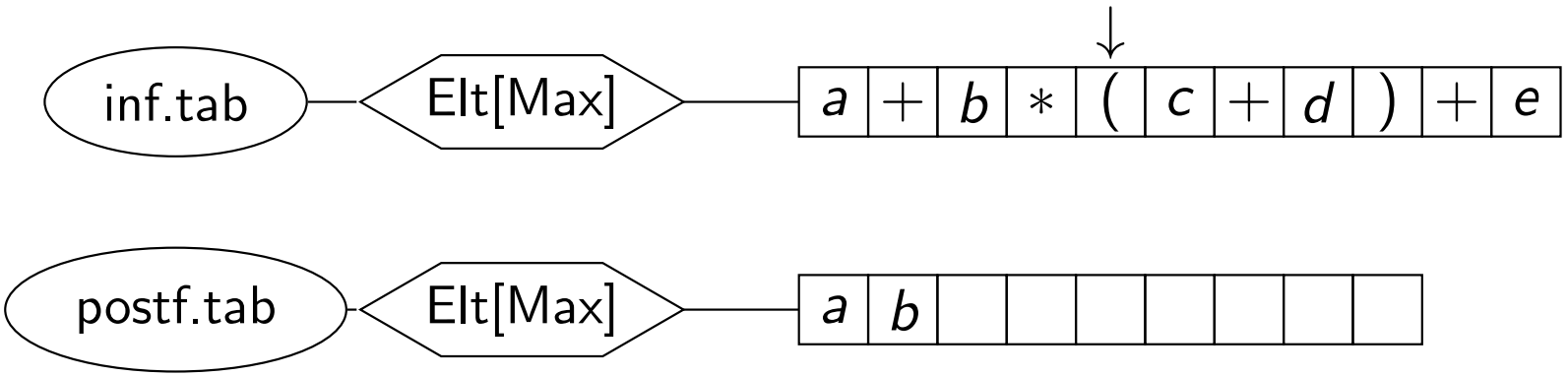
Exemplu: $a + b * (c + d) + e$

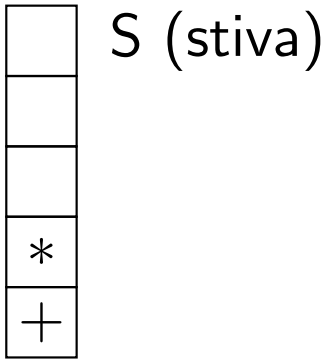


# Conversia infixat $\rightarrow$ postixat

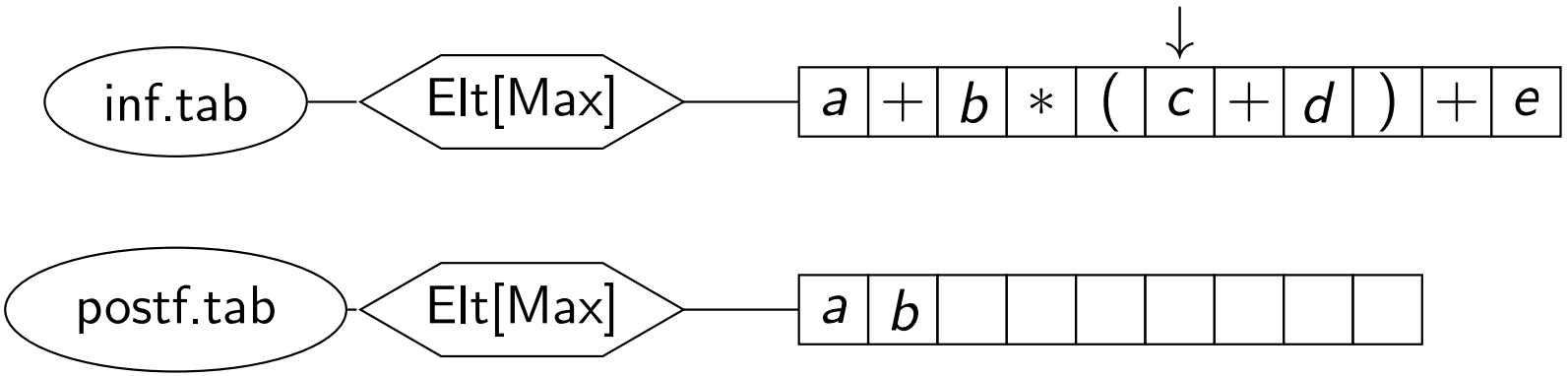
Exemplu: $a + b * (c + d) + e$

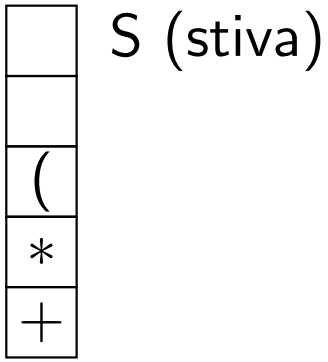


# Conversia infixat $\rightarrow$ postixat

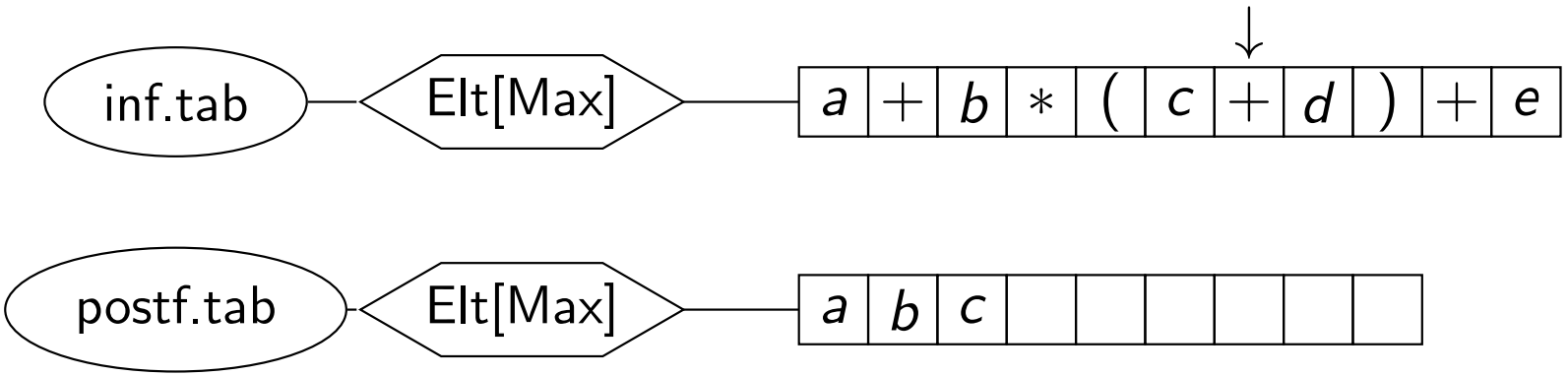
Exemplu: $a + b * (c + d) + e$

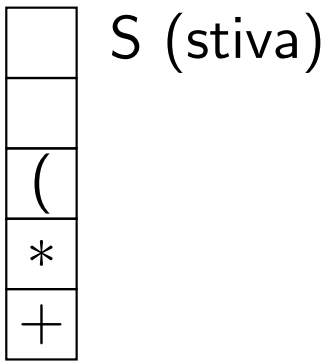


# Conversia infixat $\rightarrow$ postixat

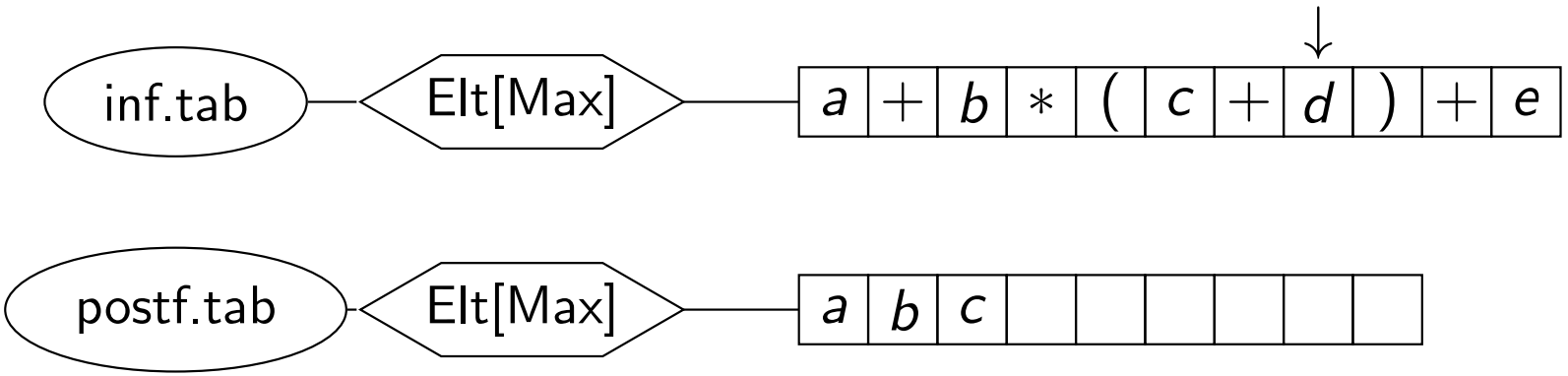
Exemplu: $a + b * (c + d) + e$

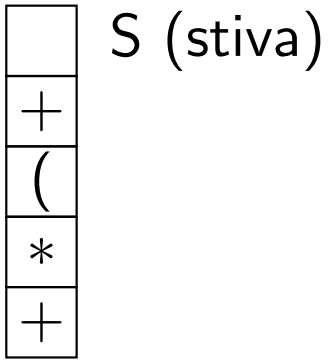


# Conversia infixat $\rightarrow$ postixat

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

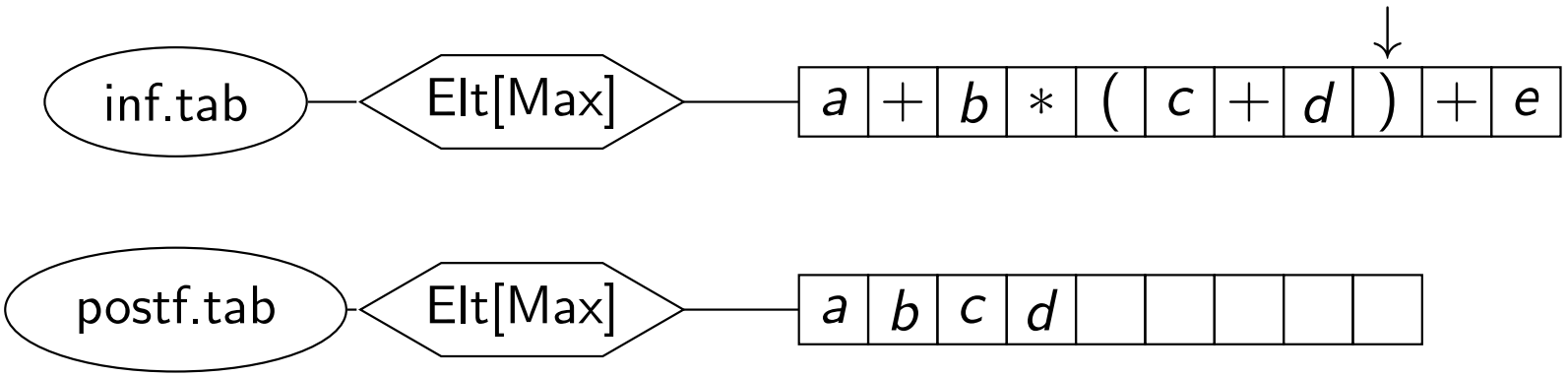


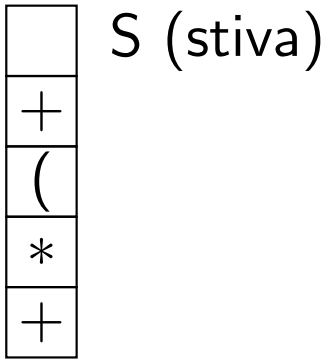




S (stiva)

# Conversia infixat $\rightarrow$ postixat

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

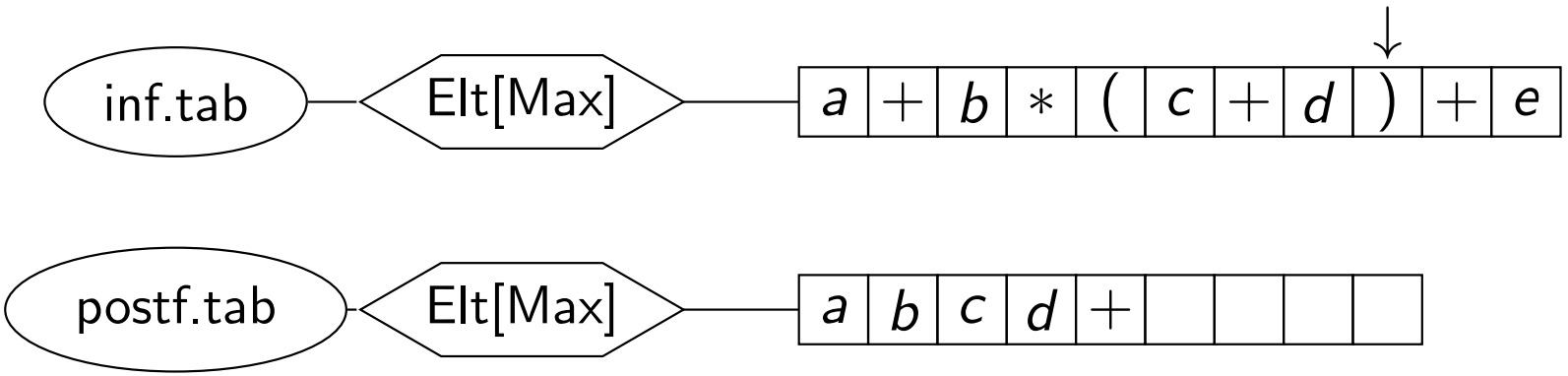


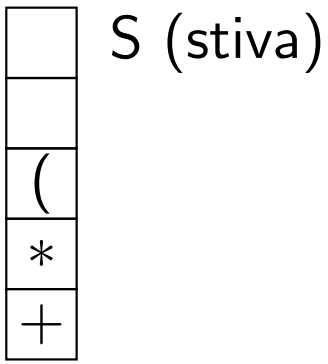




S (stiva)

# Conversia infixat $\rightarrow$ postixat

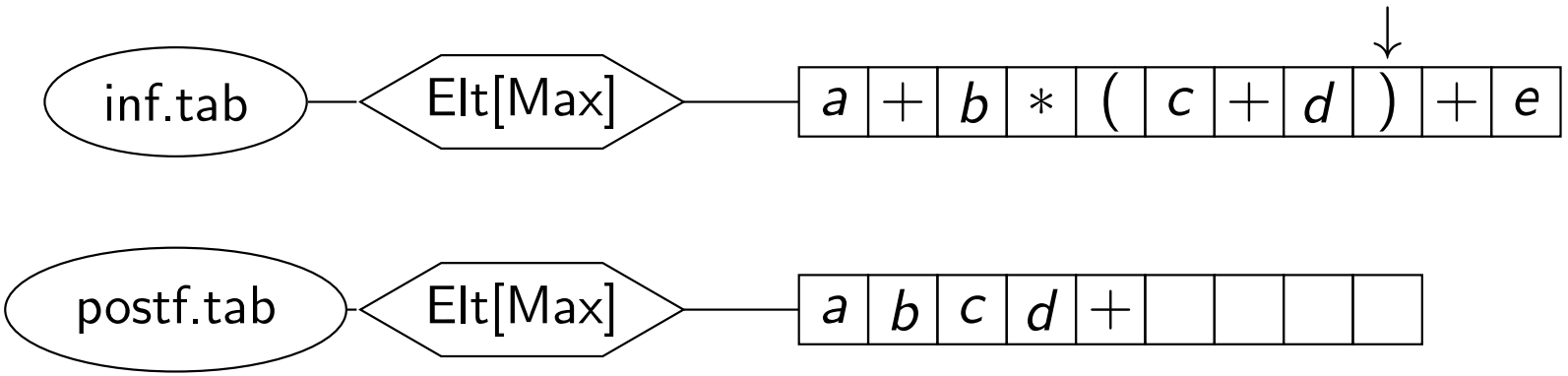
Exemplu: $a + b * (c + d) + e$

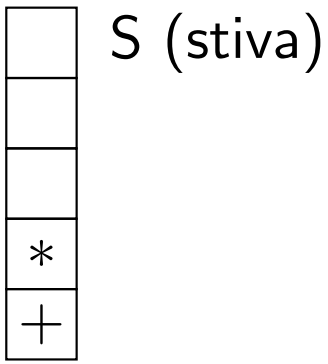


# Conversia infixat $\rightarrow$ postixat

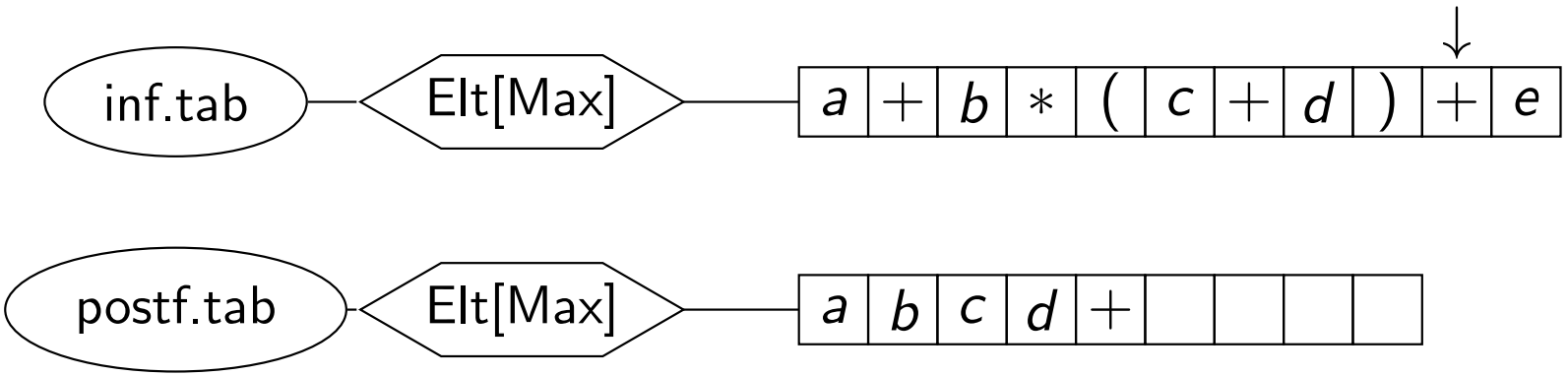
Exemplu: $a + b * (c + d) + e$

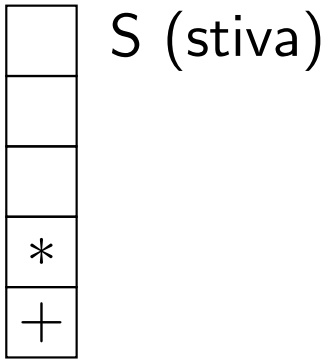


# Conversia infixat $\rightarrow$ postixat

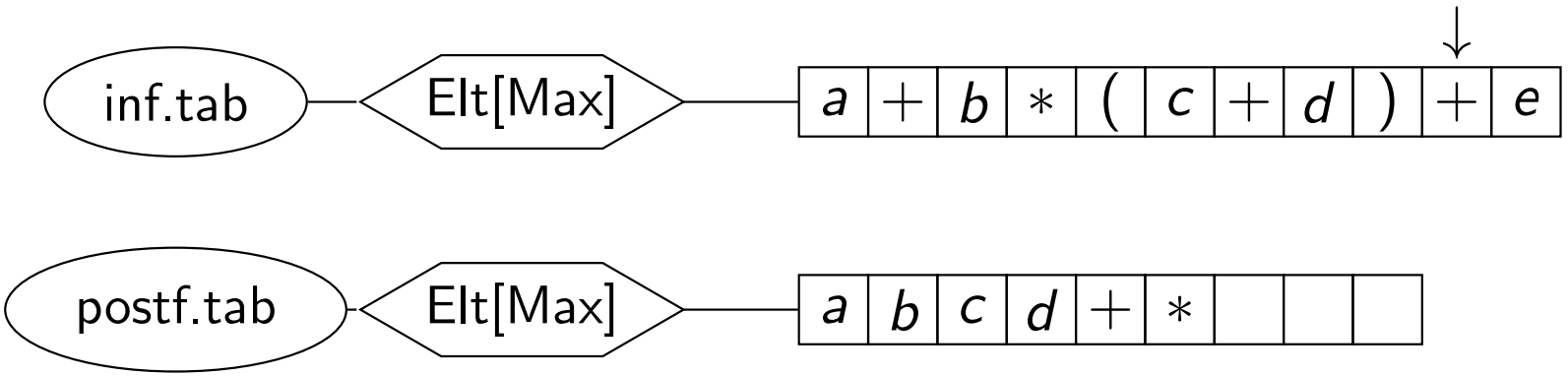
Exemplu: $a + b * (c + d) + e$

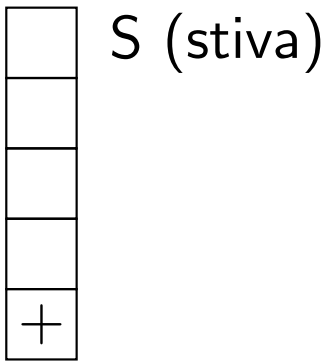


# Conversia infixat $\rightarrow$ postixat

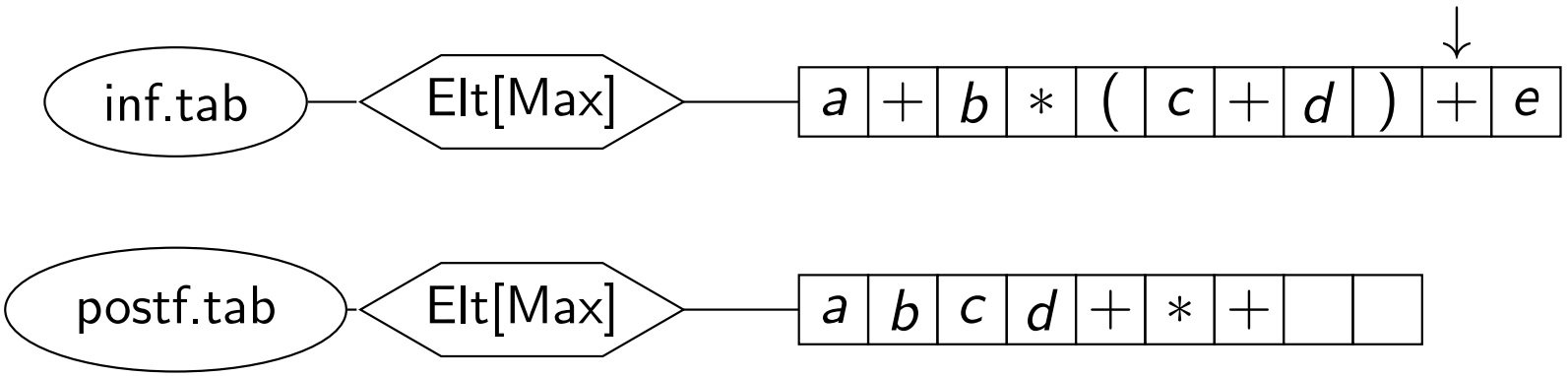
Exemplu: $a + b * (c + d) + e$

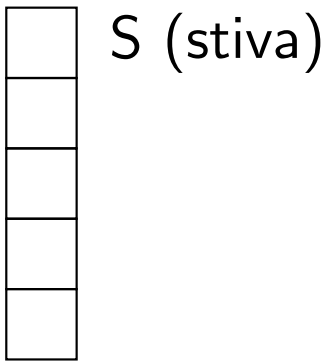


# Conversia infixat $\rightarrow$ postixat

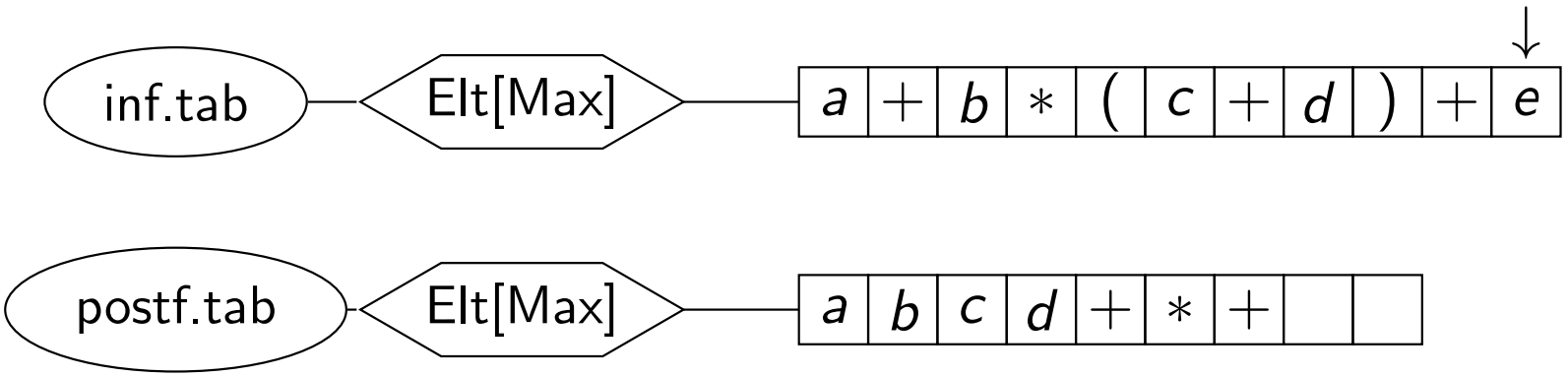
Exemplu: $a + b * (c + d) + e$

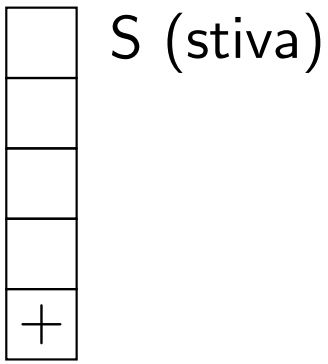


# Conversia infixat $\rightarrow$ postixat

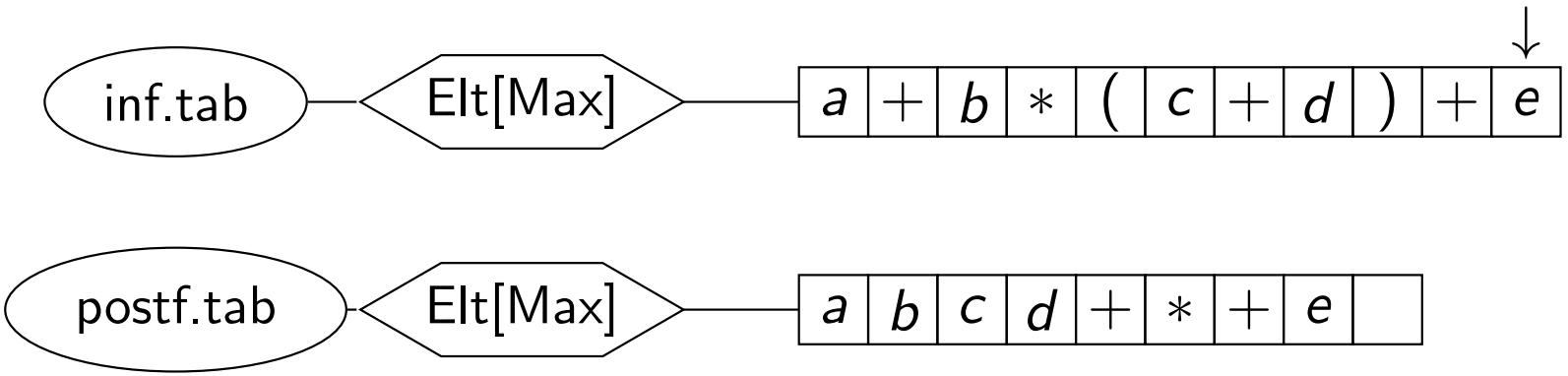
Exemplu: $a + b * (c + d) + e$

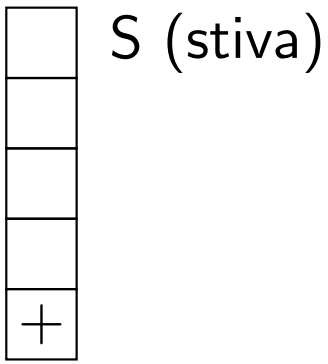


# Conversia infixat $\rightarrow$ postixat

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

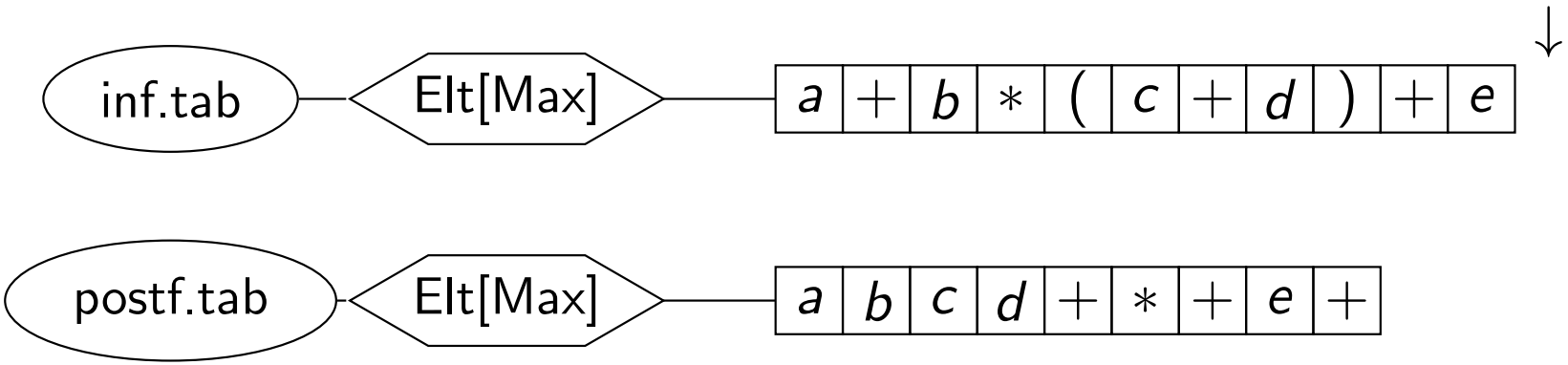


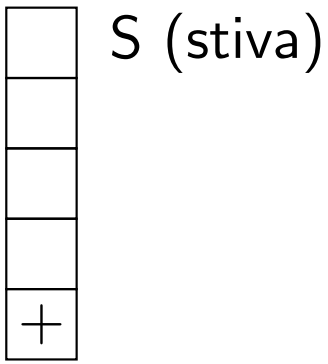




S (stiva)

# Conversia infixat $\rightarrow$ postixat

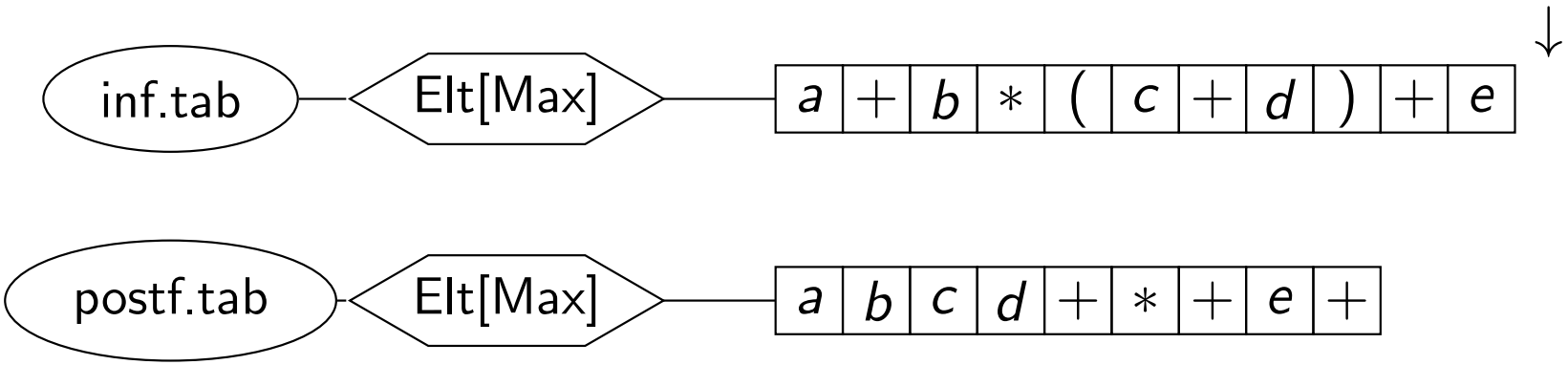
Exemplu: $a + b * (c + d) + e$

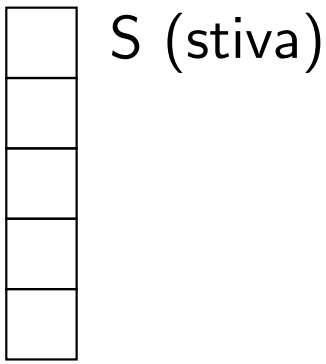


# Conversia infixat $\rightarrow$ postixat

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



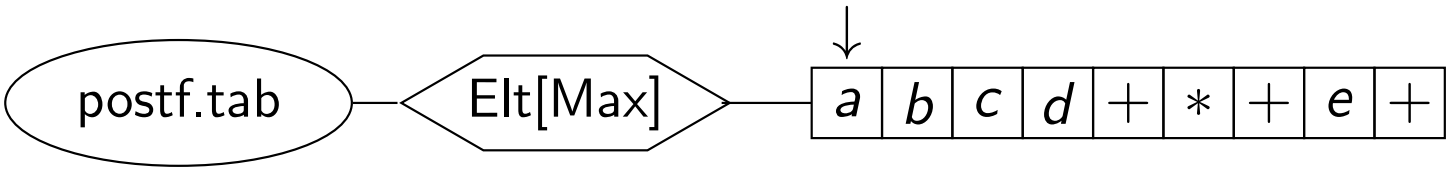



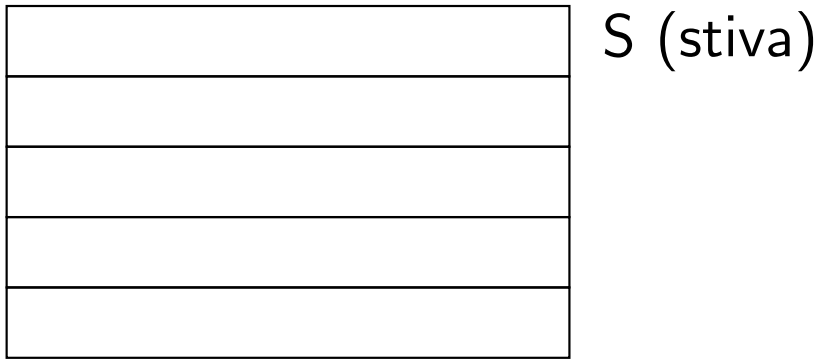
# Conversia infixat $\rightarrow$ postfixat

```
procedure convInfix2Postfix(infix, postfix)
/* infix și postfix sunt cozi*/
begin
    S ← stivaVida()
    while (not esteVida(infix)) do
        x ← citeste(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 postixate

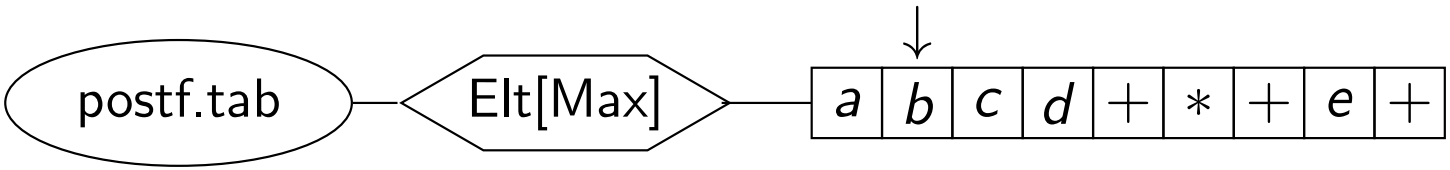
Exemplu: $a\ b\ c\ d\ +*+e+$

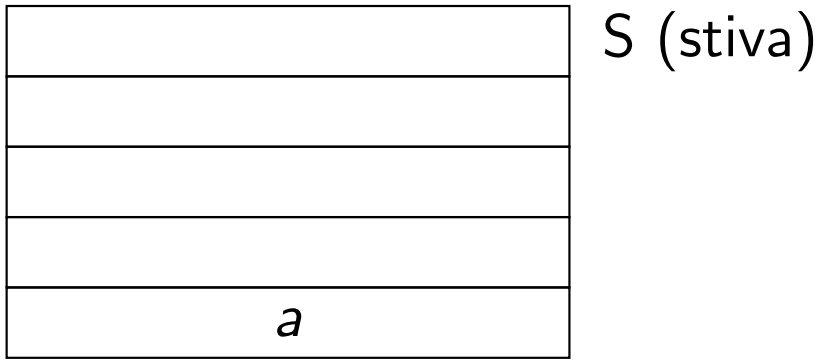


# Evaluarea expresiilor postixate

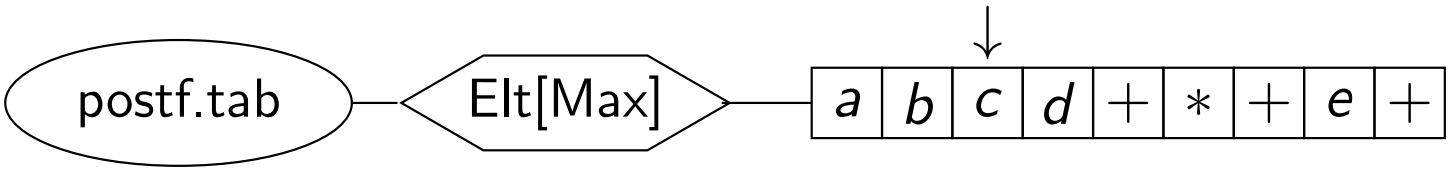
Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$

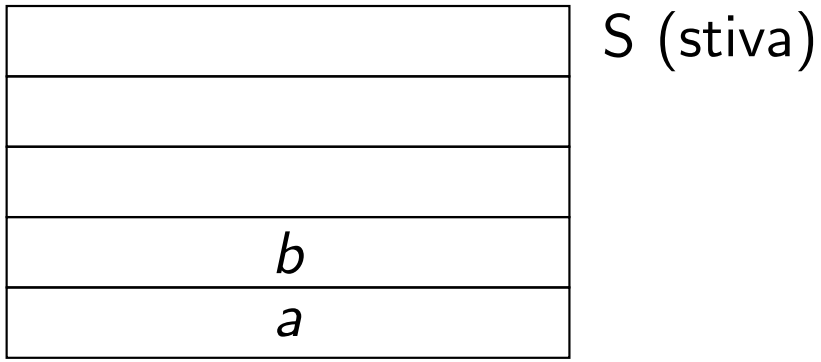


# Evaluarea expresiilor postixate

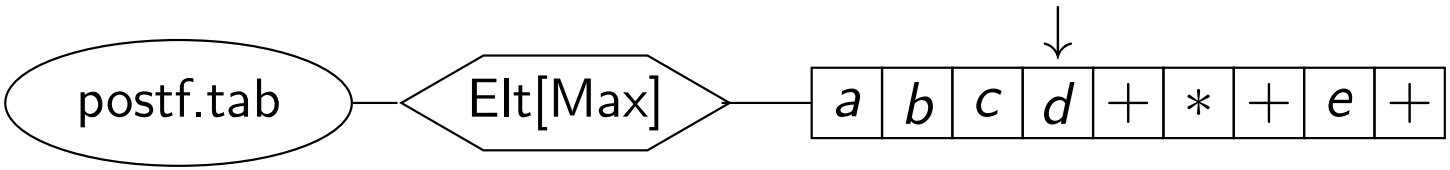
Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$

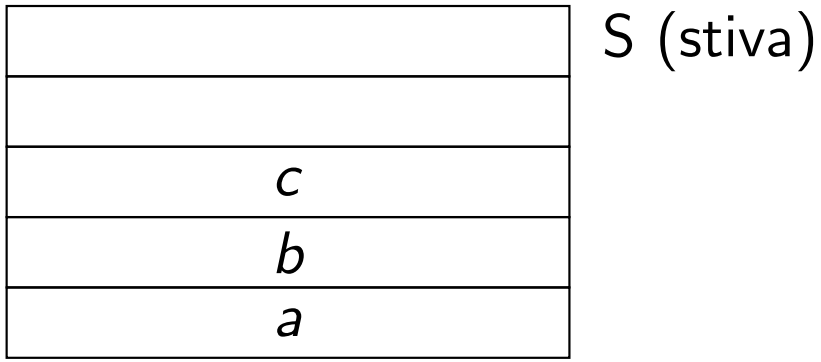


# Evaluarea expresiilor postixate

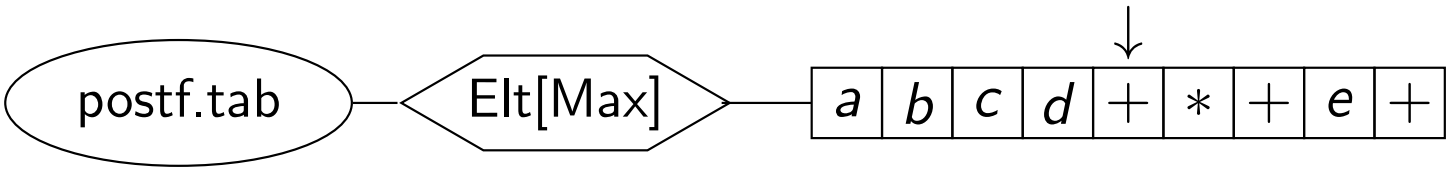
Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$

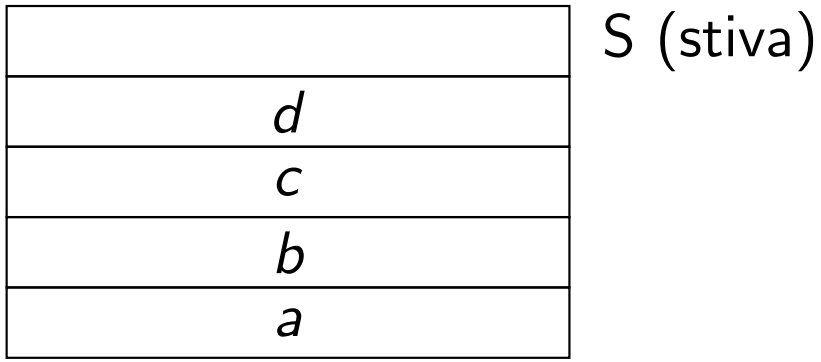


# Evaluarea expresiilor postixate

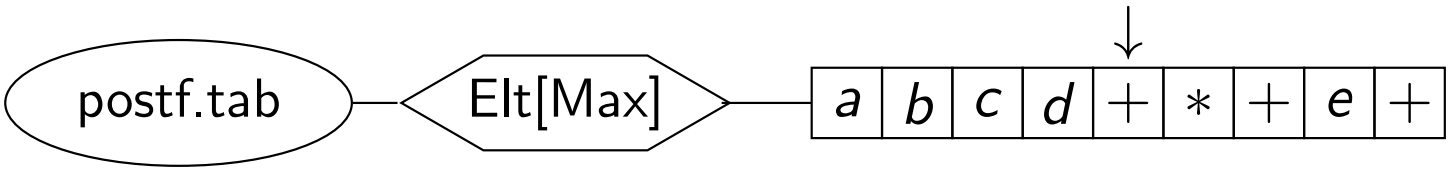
Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$

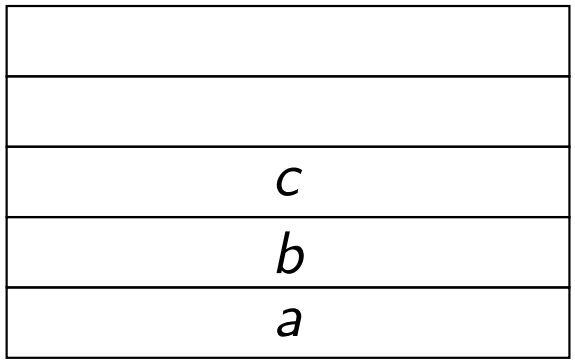


# Evaluarea expresiilor postixate

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

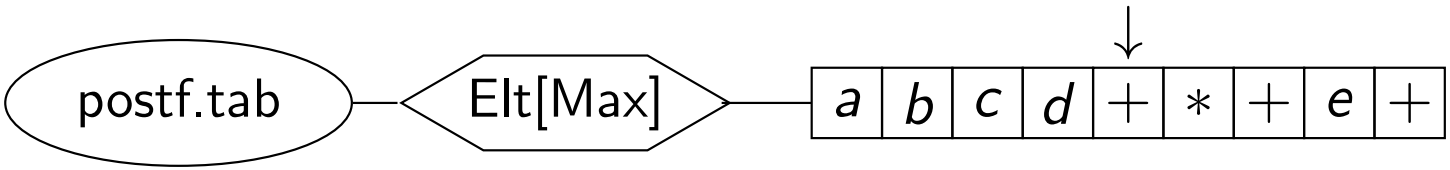


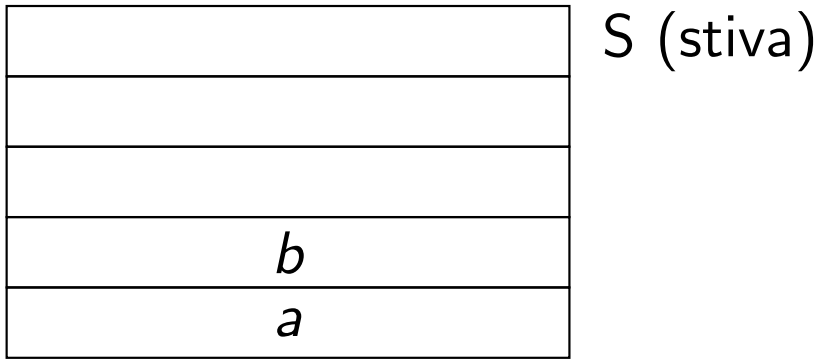




S (stiva)

# Evaluarea expresiilor postixate

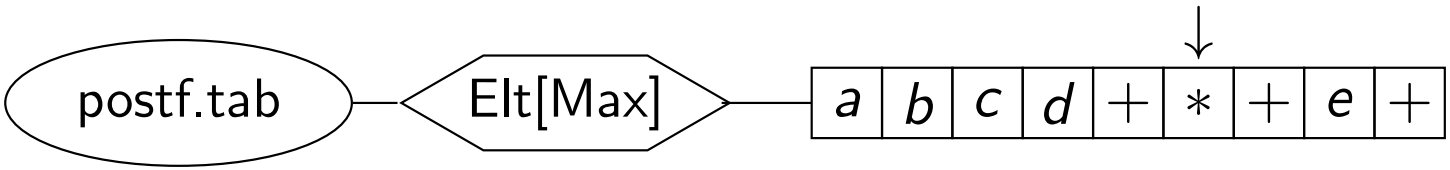
Exemplu: $a\ b\ c\ d\ +*+e+$

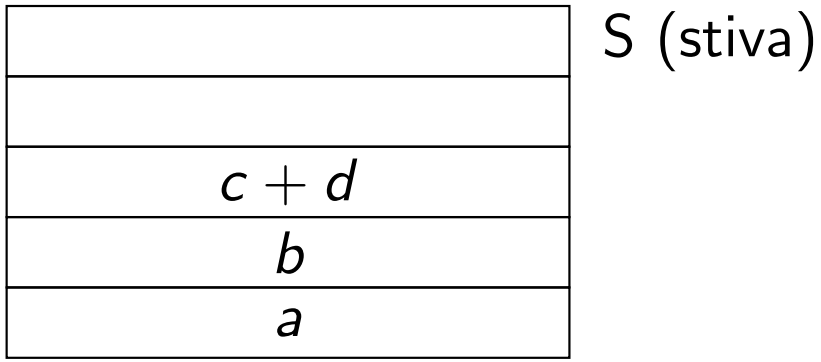


# Evaluarea expresiilor postixate

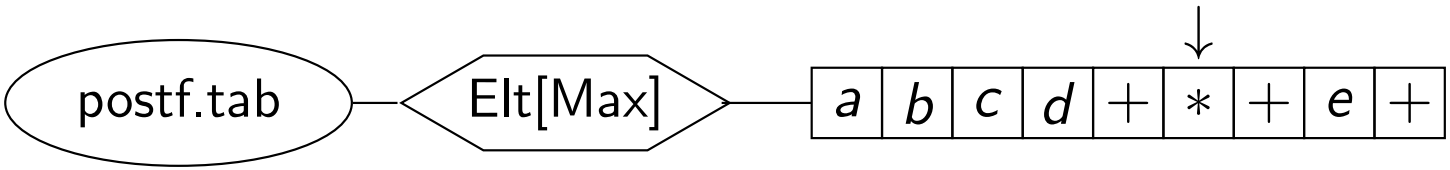
Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$

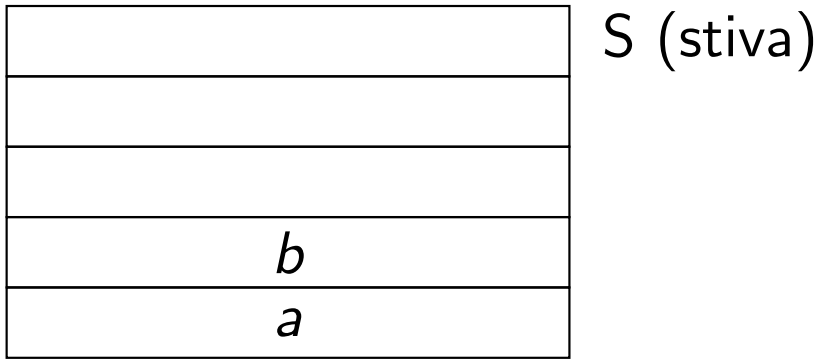


# Evaluarea expresiilor postixate

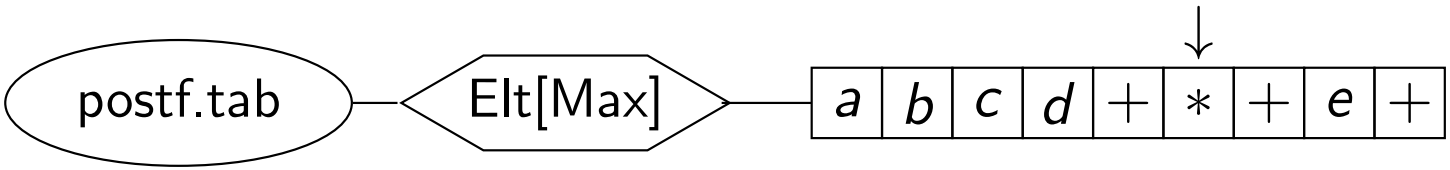
Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$

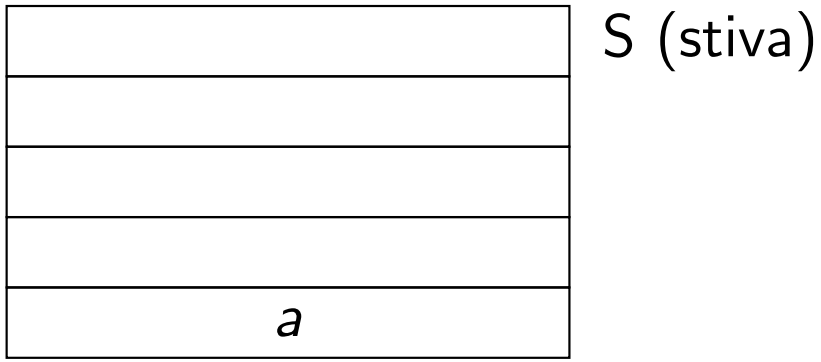


# Evaluarea expresiilor postixate

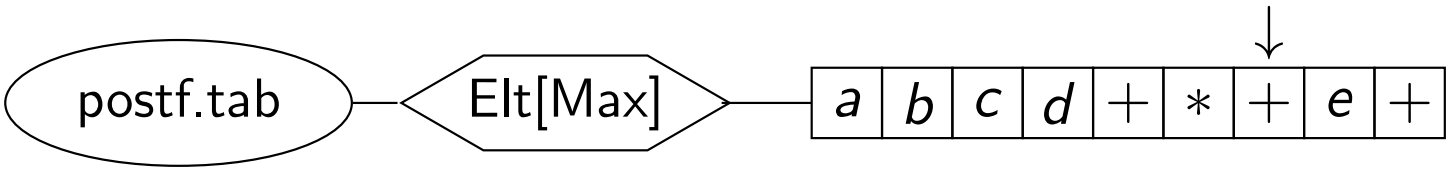
Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$

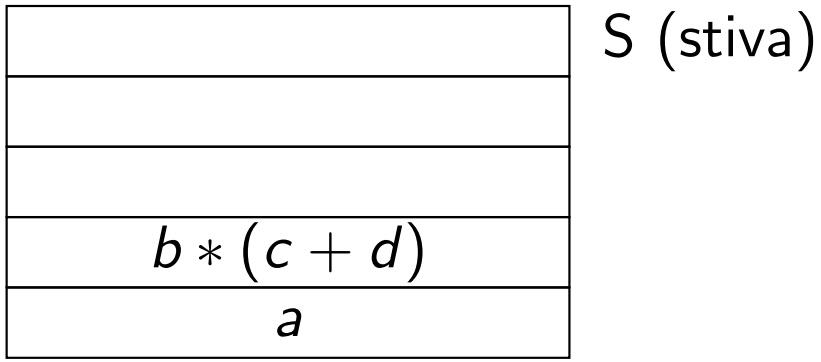


# Evaluarea expresiilor postixate

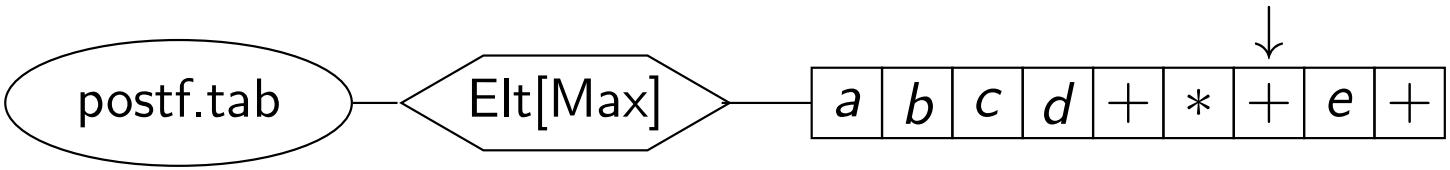
Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$

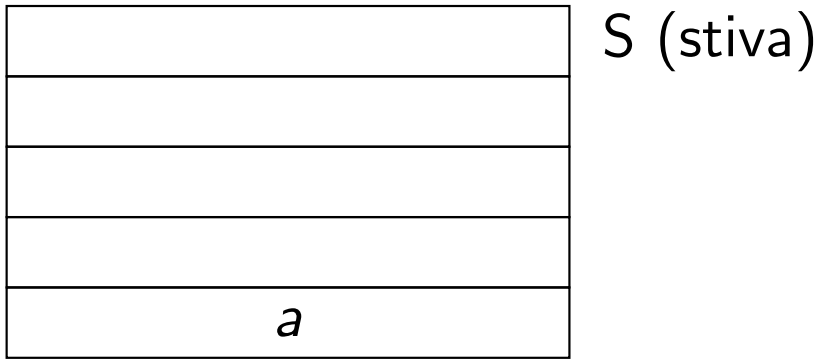


# Evaluarea expresiilor postixate

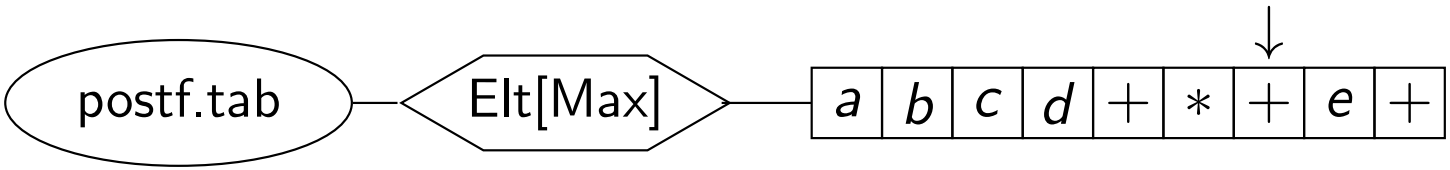
Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$

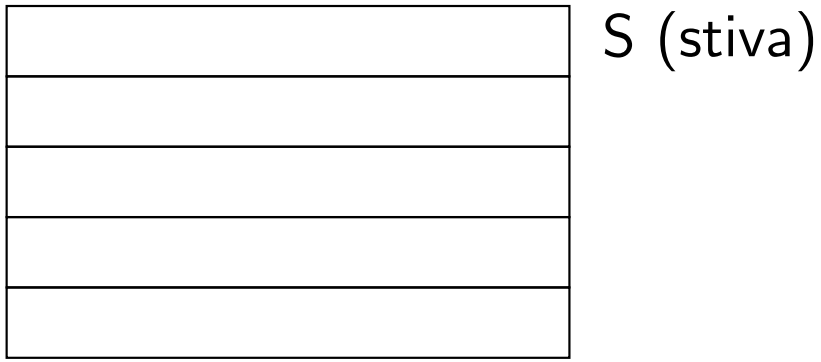


# Evaluarea expresiilor postixate

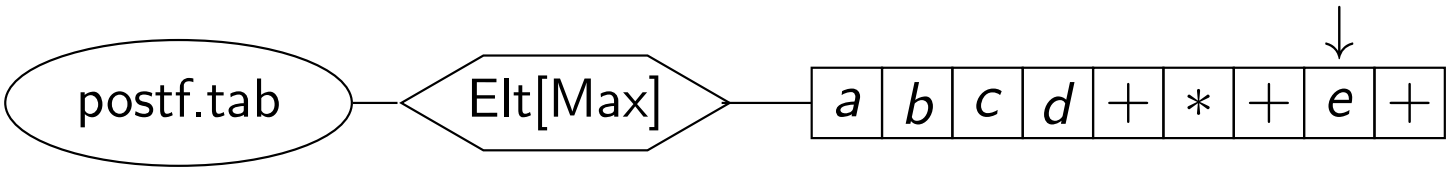
Exemplu: $a\ b\ c\ d\ +*+\ e+$

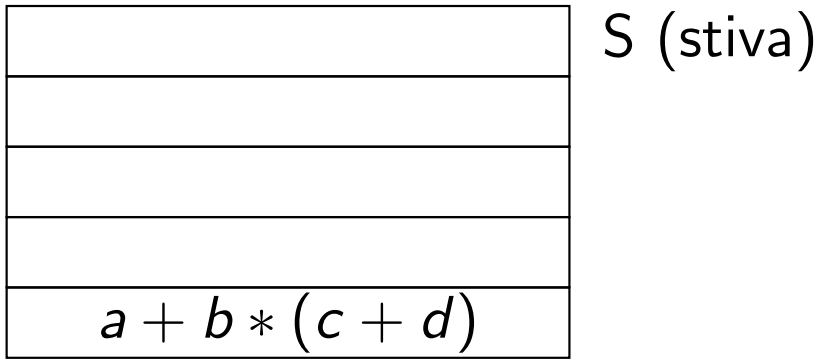


# Evaluarea expresiilor postixate

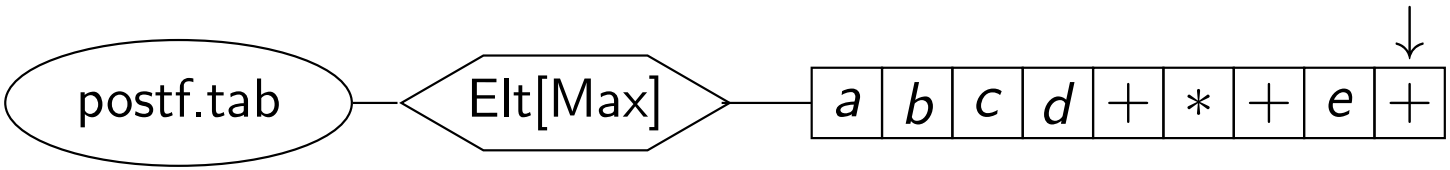
Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$

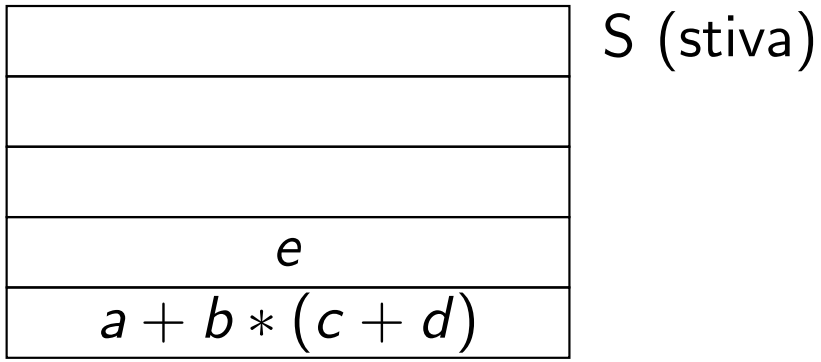


# Evaluarea expresiilor postixate

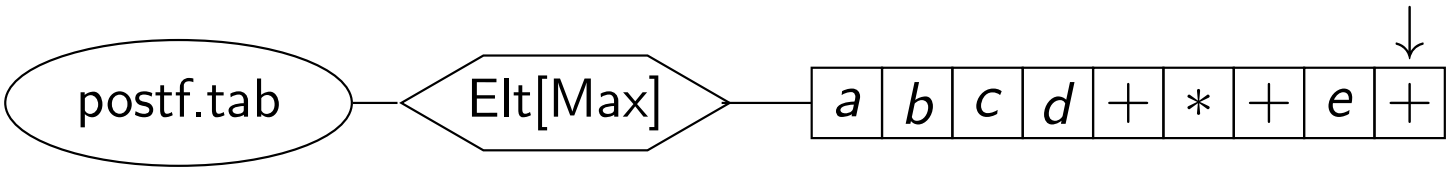
Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$

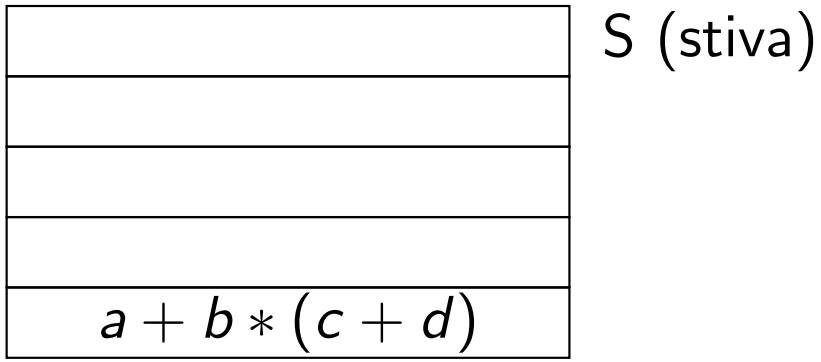


# Evaluarea expresiilor postixate

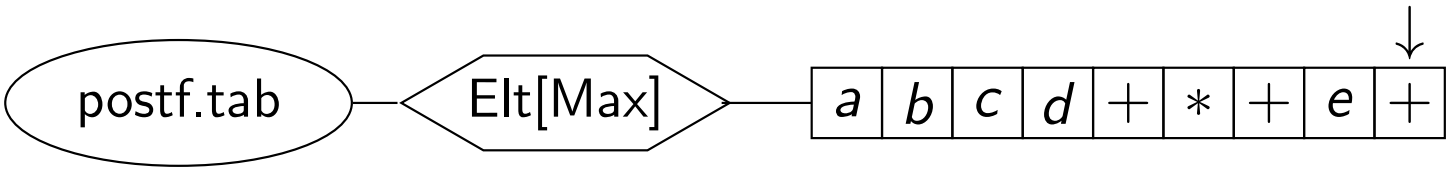
Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$

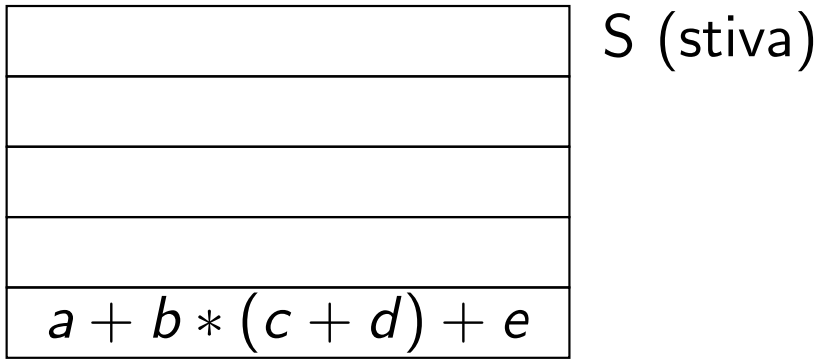


# Evaluarea expresiilor postixate

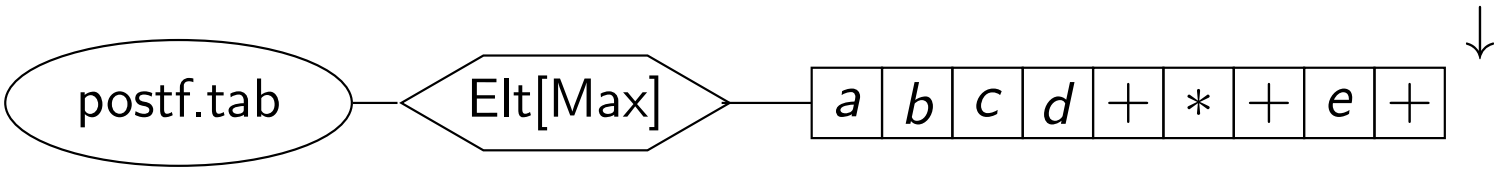
Exemplu: $a\ b\ c\ d\ +\ *\ +\ e\ +$

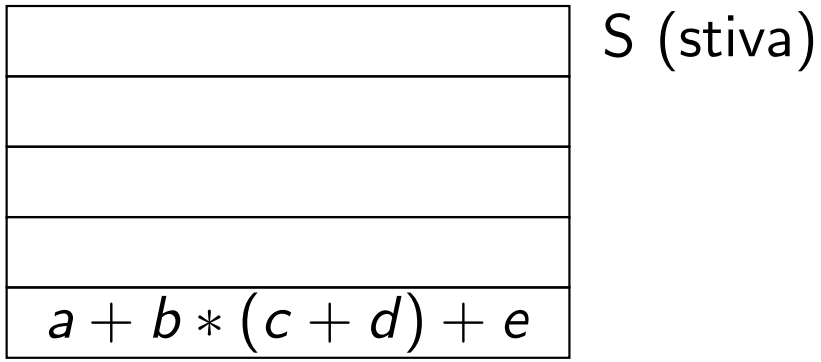


# Evaluarea expresiilor postixate

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



# Evaluarea expresiilor postfixate

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