

Liste. Stiva. Coada

SD 2015/2016

Conținut

- Tipuri de date de nivel înalt
 - Liste liniare
 - Liste liniare ordonate
 - Stiva
 - Coada
- Tipurile abstracte
 - LLin, LLinOrd, Stiva, Coada
 - Implementarea cu tablouri
 - Implementarea cu liste simplu înlănțuite
- Aplicație la conversii de expresii

Liste liniare - exemple

- Studenți
 - (Adriana, George, Luiza, Maria, Daniel)
- Examene
 - (Mate, Logica, SD, ACSO, ENG)
- Zile săptămânii
 - (L, M, Mi, J, V, S, D)
- Lunile anului
 - (Ian, Feb, Mar, Apr, ..., Dec)

Tipul abstract **LLin**

- **Obiecte:** $L = (e_0, \dots, e_{n-1}), n \geq 0$
- $e_i \in \mathbf{El}t$ (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:
 - $L = (e_0, \dots, e_{n-1})$, $k \in \mathbf{Nat}$, $e \in \mathbf{Elt}$
 - 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:
 - $L = (e_0, \dots, e_{n-1})$, $k \in \mathbf{Nat}$
 - ieșire
 - $L = (\dots e_{k-1}, e_{k+1} \dots)$ dacă $0 \leq k \leq n-1$
 - eroare în caz contrar

elimina() - example

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

- $\text{elimina}(L, 2) \Rightarrow$
 $L = (a, b, d, e, f, g)$
(Obs.: Indexul elementelor d, \dots, g descrește cu 1.)
- $\text{elimina}(L, 10) \Rightarrow$ eroare
- $\text{elimina}(L, -7) \Rightarrow$ eroare

LLin – operații

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

alKlea() – exemple

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

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

LLin – operații

- elimTotE()

- intrare:

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

- ieșire:

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

elimTotE () - exemple

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

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

LLin – operații

- 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 () - exemple

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

- $\text{parcure}(L, \text{oriDoi}()) \Rightarrow (2, 4, 6, 2, 4, 6)$
- $\text{parcure}(L, \text{incrementeaza}()) \Rightarrow (2, 3, 4, 2, 3, 4)$

LLin – operații

- poz()

- intrare:

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

- ieșire:

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

poz () - exemple

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

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

LLin – operații

- lung()

- intrare:

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

- ieșire:

- n – lungimea listei L

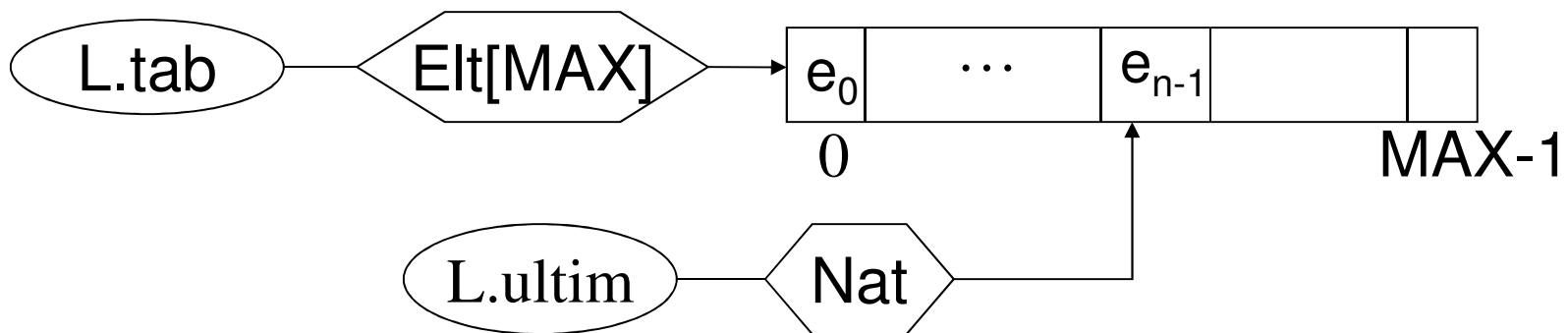
- Exemplu:

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

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

LLin: implementare cu tablouri

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



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

LLin: implementare cu tablouri

- insereaza()

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

LLin: implementare 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 ← L.ultim downto k do
        L.tab[j+1] ← L.tab[j]
    L.tab[k] ← e
    L.ultim ← L.ultim + 1
end
```

– timpul de executie: $O(n)$

LLin: implementare cu tablouri

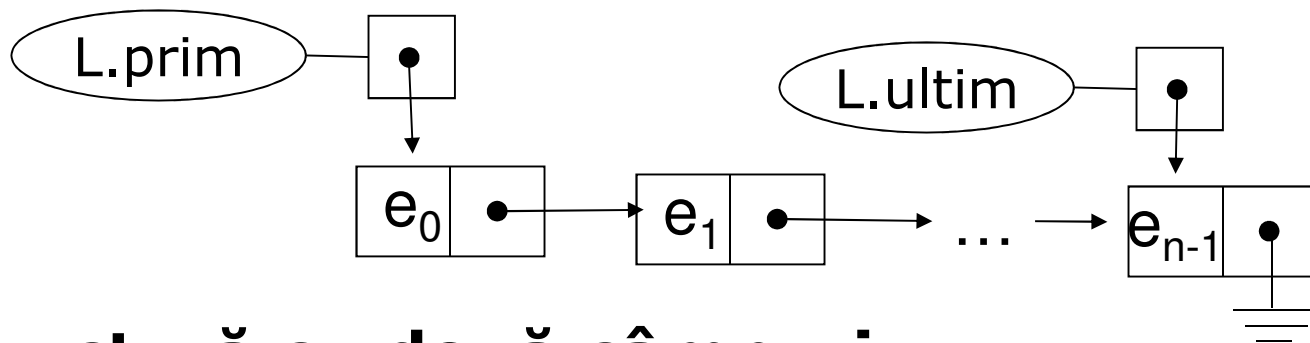
- `parcure()`

```
procedure parcure(L, viziteaza())  
begin  
    for i ← 0 to L.ultim do  
        viziteaza(L.tab[i])  
    end
```

- dacă `viziteaza()` procesează un element în $O(1)$, atunci `parcure()` procesează lista în $O(n)$ (n este numărul elementelor listei)

LLin: implementare cu structuri înlănțuite

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

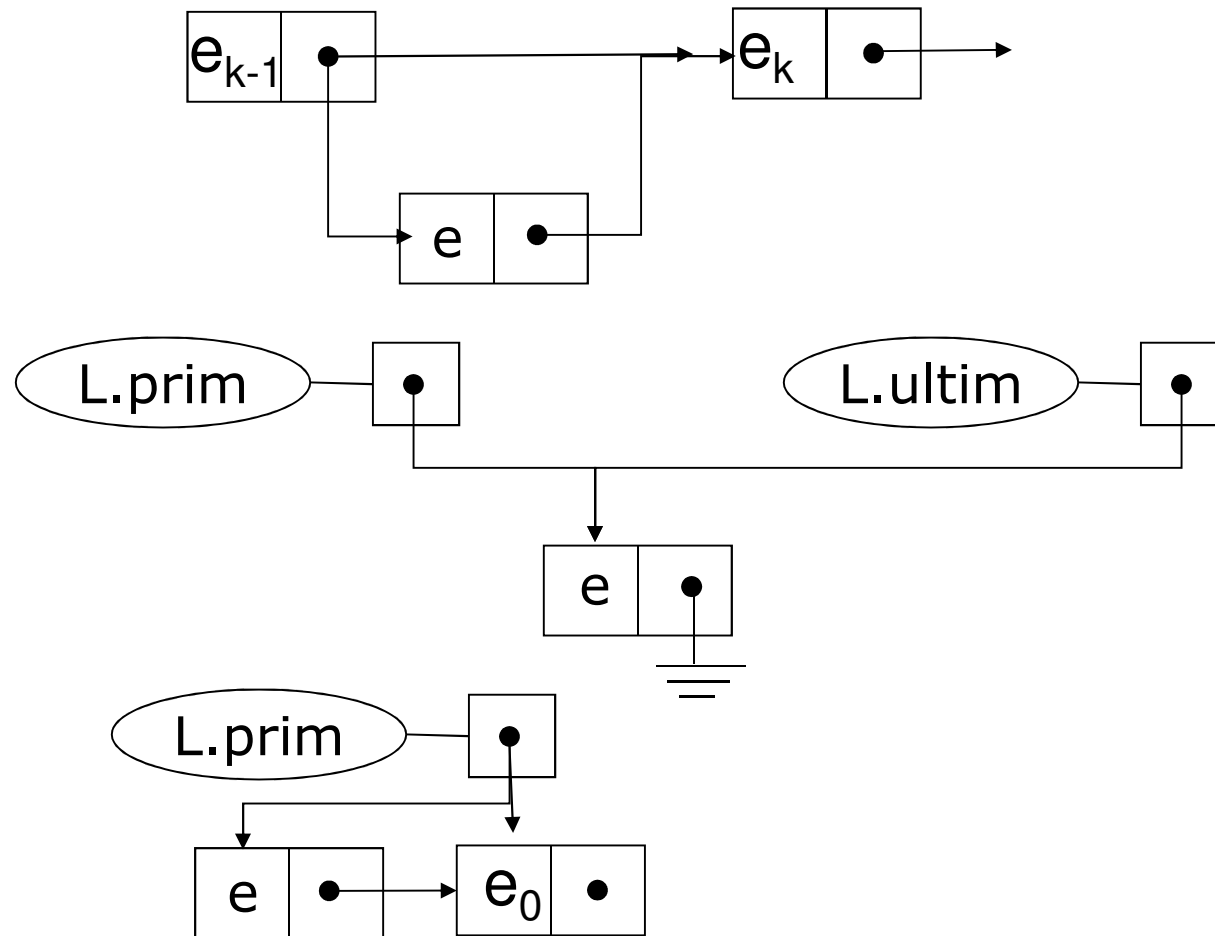


- **L structură cu două câmpuri**
 - pointer la primul element
 - pointer la ultimul element
- **un nod $*p$ (aflat la adresa din p) are două câmpuri:**
 - unul pentru memorarea informației: $p \rightarrow \text{elt} = e_i$
 - unul pentru nodul succesor: $p \rightarrow \text{succ}$

LLin: implementare cu structuri înlănțuite

- insereaza()
 - parcurge elementele $0, 1, \dots, k-1$
 - inserează un nou element după $k-1$
 - creează nodul
 - memorează informații
 - reface legături
 - excepții
 - lista vida
 - $k=0$
 - $k=n$
 - $k<0, k > n$

LLin: implementare cu structuri înlănțuite



LLin: implementare 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();
    /* citește n */
    for i ← 0 to n-1 do
        /* citește p.x, p.y */
        insereaza(L, 0, p)
    end
```

- atenție: timpul de execuție depinde de implementare

LLin: aplicație

- multiplică cu 2 coordonatele unui punct

```
procedure ori2Punct (p)
```

```
begin
```

```
    p.x ← p.x * 2
```

```
    p.y ← p.y * 2
```

```
end
```

- multiplică cu 2 coordonatele unei linii poligonale

```
procedure ori2Linie (L)
```

```
begin
```

```
    parcurge (L, ori2Punct ())
```

```
end
```

LLin: aplicație

– translatează punct

```
procedure trPunct(p, dx, dy)
begin
```

```
    p.x ← p.x + dx
```

```
    p.y ← p.y + dy
```

```
end
```

– translatează linie poligonală

```
procedure trLinie(L, dx, dy)
begin
```

```
    parcurge(L, trPunct())
```

```
end
```

Liste liniare ordonate: **LLinOrd**

- obiecte
 - $L = (e_0, \dots, e_{n-1}), n \geq 0, e_i \in \mathbf{Elt}, e_0 < \dots < e_{n-1}$
- operații
 - listaVida()
 - intrare: nimic
 - iesire
 - $()$ (lista cu zero elemente)
 - insereaza()
 - intrare:
 - $L = (e_0, \dots, e_{n-1}), e \in \mathbf{Elt}$
 - 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:

- $L = (e_0, \dots, e_{n-1}), e \in \mathbf{El}t$

- ieșire

- $L = (\dots e_{k-1}, e_{k+1} \dots)$ dacă $e = e_k$

- eroare în caz contrar

– `alKlea()`

– `parcurge()`

– `poz()`

LLin: implementare cu tablouri

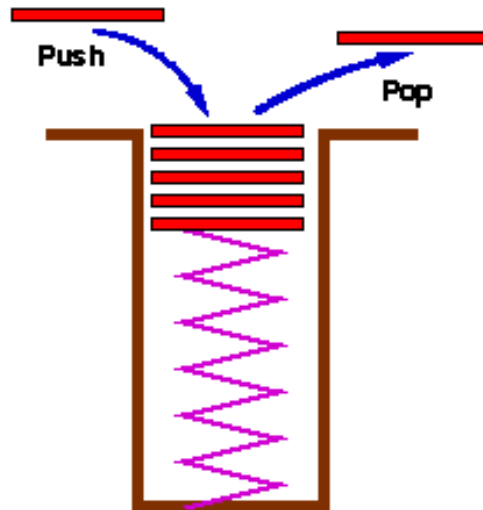
- **căutare binară**

```
function Poz(L, e)
begin
    p ← 0; q ← L.ultim
    m ← [(p+q)/2]
    while(L.tab[m] != e && 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
```

LLin: complexitatea căutării

- implementare cu tablouri
 - timpul de execuție $O(\log n)$
- implementarea cu liste înlănțuite
 - timpul de execuție $O(n)$ (căutare liniară)

Stiva



Stiva – aplicații

- Aplicații directe
 - istoricul paginilor web vizitate într-un browser;
 - secvența “undo” într-un editor text;
 - șirul 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)
- operații
 - **stivaVida()**
 - intrare: nimic
 - ieșire
 - $S = ()$, lista vida
 - **push()**
 - intrare
 - $S \in \text{Stiva}, e \in \text{Elt}$
 - ieșire
 - S la care s-a adăugat e ca ultimul element introdus (cel cu vechimea cea mai mică)

Tipul abstract **Stiva**

– **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

– **esteVida()**

- intrare
 - $S \in \text{Stiva}$
- ieșire
 - **true** dacă S este vidă
 - **false** dacă S nu este vidă

Tipul abstract **Stiva**

– top()

- intrare

- $S \in \text{Stiva}$

- ieșire

- ultimul element introdus (cel cu vechimea cea mai mică)
 - eroare dacă S este vidă

Stiva: implementare cu liste

$\text{push}(S, e) = \text{insereaza}(S, 0, e)$

$\text{pop}(S) = \text{elimina}(S, 0)$

$\text{top}(S) = \text{alKlea}(S, 0)$

sau

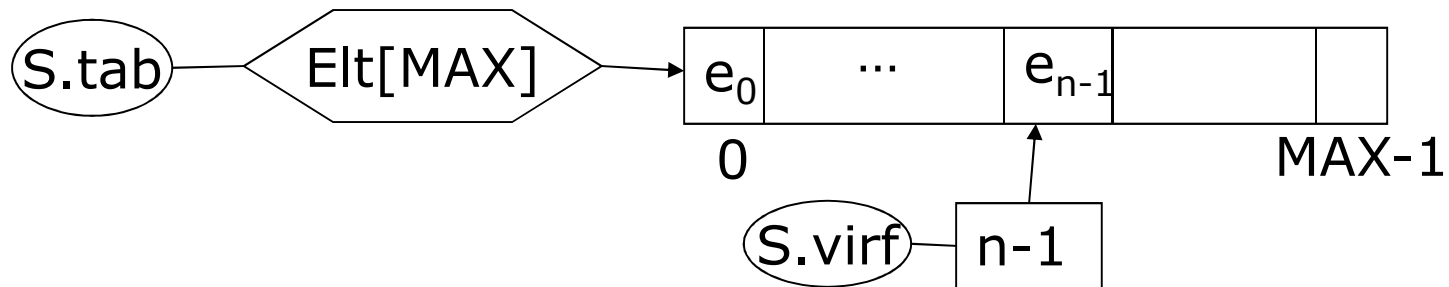
$\text{push}(S, e) = \text{insereaza}(S, \text{lung}(S), e)$

$\text{pop}(S) = \text{elimina}(S, \text{lung}(S)-1)$

$\text{top}(S) = \text{alKlea}(S, \text{lung}(S)-1)$

Stiva: implementare cu tablouri

- reprezentarea obiectelor



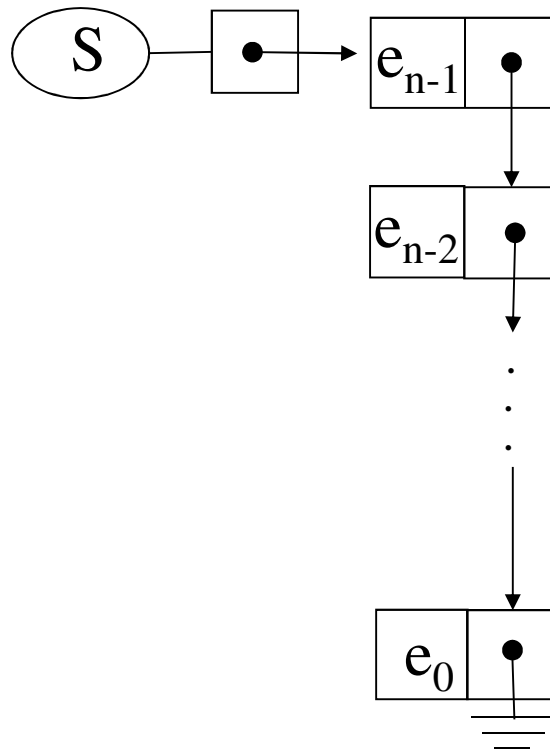
- implementarea operațiilor

– `push()`

```
procedure push(S, e)
begin
    if (S.virf = MAX-1)
    then throw "eroare"
    S.virf ← S.virf+1
    S.tab[virf] ← e
end
```

Stiva: implementare cu structuri înlanțuite

- reprezentarea obiectelor



Stiva: implementare cu structuri înlănțuite

- implementarea operațiilor

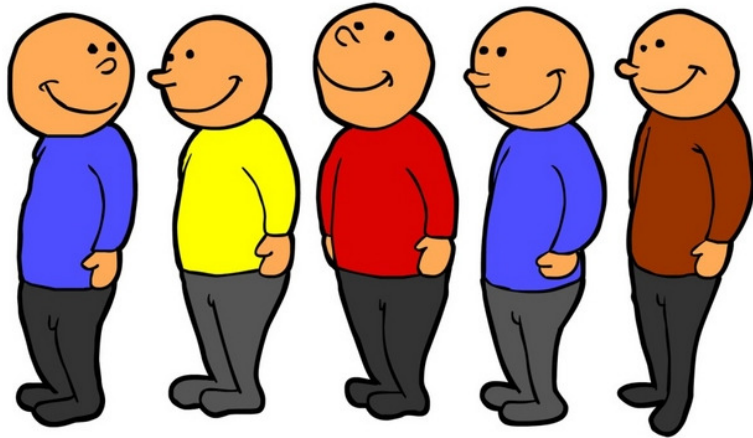
– push()

```
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
```

Coada



Coada – aplicații

- Aplicații directe
 - Liste / fire de așteptare;
 - Accesul la resurse partajate (ex: imprimare).
- Aplicații indirecte
 - structură de date auxiliară în anumiți algoritmi.

Tipul abstract Coadă

- obiecte
 - liste în care se cunoaște vechimea elementelor introduse (liste FIFO)
- operații
 - coadaVida()
 - intrare: nimic
 - ieșire
 - lista vidă
 - insereaza()
 - intrare
 - $C \in \text{Coadă}, e \in \text{Elt}$
 - ieșire
 - C la care s-a adăugat e ca ultimul element introdus (cel cu vechimea cea mai mică)

Tipul abstract Coadă

– 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

– esteVidă()

- intrare
 - $C \in \text{Coadă}$
- ieșire
 - **true** dacă C este vidă
 - **false** dacă C nu este vidă

Tipul abstract Coadă

– citeste()

- intrare

- $C \in \text{Coadă}$

- ieșire

- primul element introdus (cel cu vechimea cea mai mare)
 - eroare dacă C este vidă

Coadă: implementare cu liste

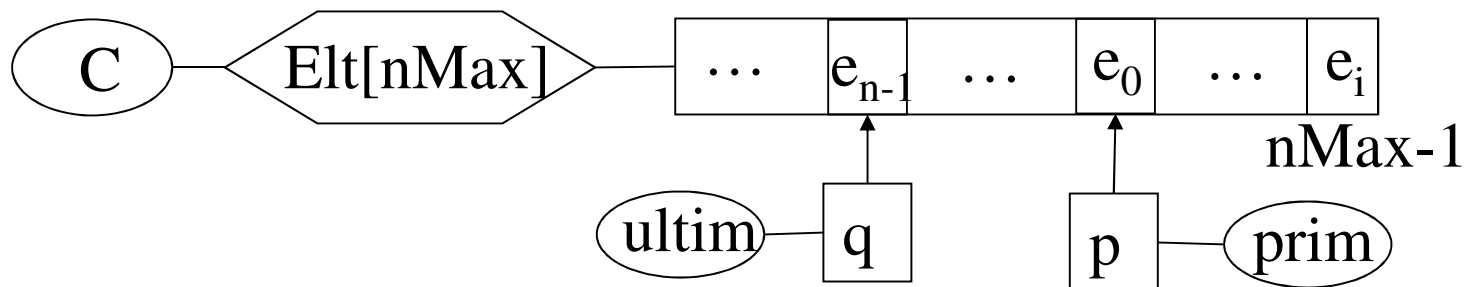
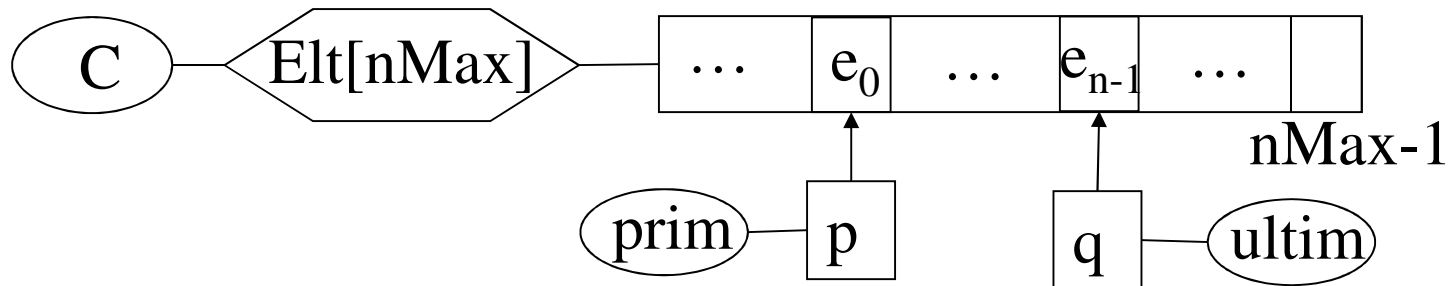
`insereaza(C, e) = LLin.insereaza(C, lung(C), e)`

`elimina(C) = LLin.elimina(C, 0)`

`citeste(C) = LLin.alKlea(C, 0)`

Coadă: implementare cu tablouri

- reprezentarea obiectelor



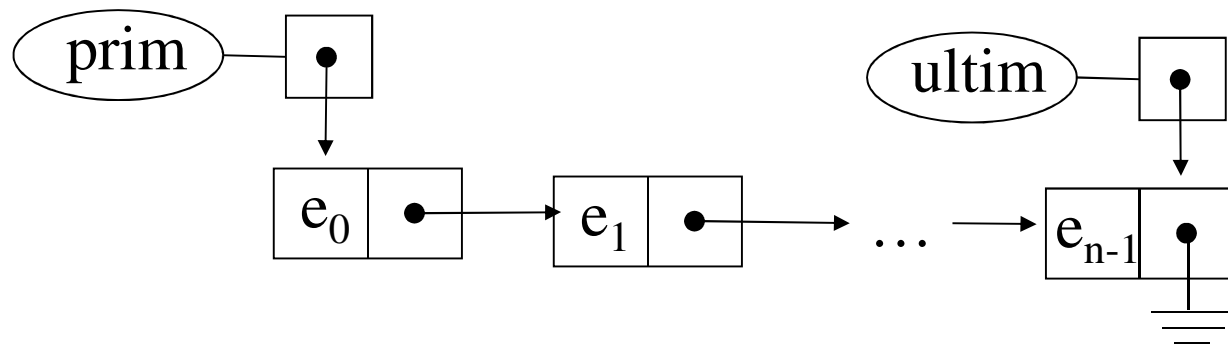
Coadă: implementare cu tablouri

- implementarea operațiilor
 - `insereaza()`

```
procedure insereaza(C, e)
begin
    if ((ultim + 1) % nMax = prim)
    then throw "error"
    ultim ← (ultim + 1) % nMax
    C[ultim] ← e
end
```

Coada: implementare cu structuri înlănțuite

- reprezentarea obiectelor



Stiva: implementare cu structuri înlănțuite

- implementarea operațiilor
– insereaza()

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

Aplicație: notația postfixată a expresiilor

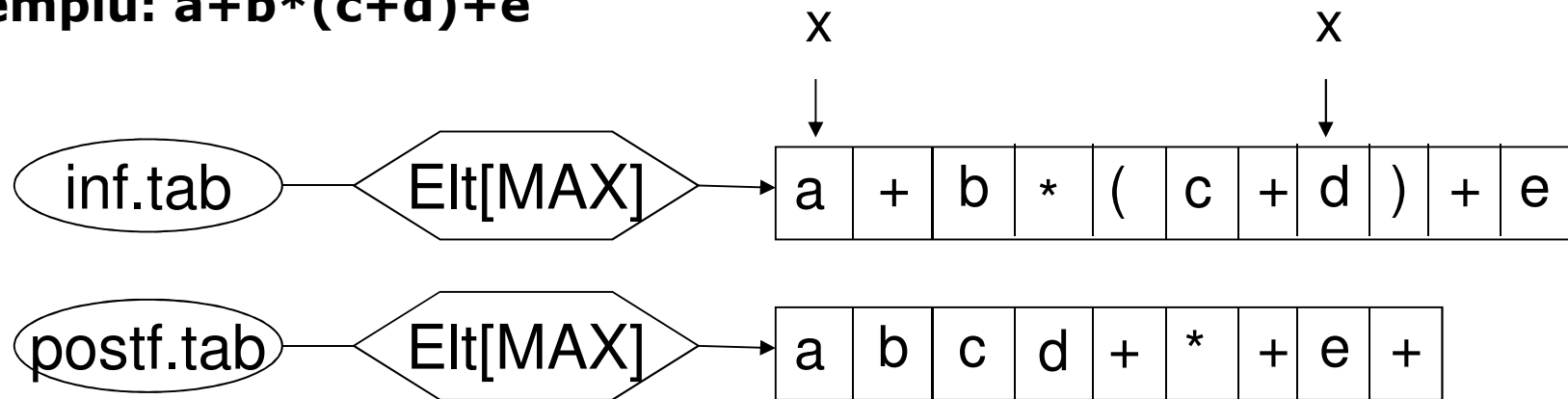
- notația infixată
 $a + b$
 $a + (b * 2)$
- notația postfixată
 $a \ b \ +$
 $a \ b \ 2 \ * \ +$
- reguli de precedență
 $a + b * 2$
- reguli de asociere
 $7 / 3 * 2$
 - la stânga $(7 / 3) * 2$
 - la dreapta $7 / (3 * 2)$

Conversia infixat -> postfixat

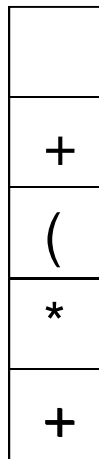
```
procedure convInfix2Postfix(infix, postfix)
/* infix si postfix sunt cozi */
begin
  S ← stivaVida()
  while (not esteVida(infix)) do
    citeste(infix, x); elimina(infix);
    if (x este operand) then insereaza(postfix, x)
    else if (x = '(') then
      while (top(s) ≠ '(') do
        insereaza(postfix, 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
```

Conversia infixat -> postfixat

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



S (stiva)

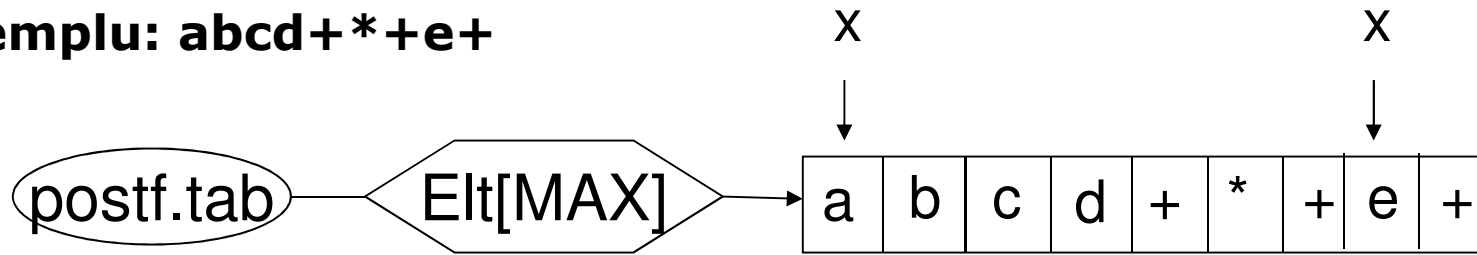


Evaluarea expresiilor postfixate

```
function valPostfix(postfix)
begin
    S ← stivaVida()
    while (!esteVida(postfix)) do
        citește(postfix, x); elimina(postfix)
        if (x este operand)
        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
```

Conversia infixat -> postfixat

Exemplu: $abcd+*+e+$



S (stiva)

d
ced
b*(c+d)
a*b*(c+d)e