Liste. Stiva. Coada

SD 2014/2015

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 înlănţuite
- Aplicaţie 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: $L = (e_0, \ldots, e_{n-1}),\ n \geq 0$
- $e_i \in$ **`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 = $(e_0,\ldots, e_{n-1})$, k $\in$ **Nat**, e $\in$ **Elt**
  - ieșire
    - L = $(\ldots e_{k-1}, e, e_k,\ldots)$ dacă $0 \leq k \leq n$
    - eroare în caz contrar

insereaza() - exemple

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

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

  (Obs.: Indexul elementelor a,...,g crește cu 1.)

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

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

- insereaza(L, 10, x) => eroare
- insereaza(L, -7, x) => eroare

LLin – operații

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

`elimina()` - exemple

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

- elimina(L, 2) =>
  L = (a, b, d, e, f, g)

  (Obs.: Indexul elementelor d,...,g descreşte cu 1.)

- elimina(L, 10) => eroare

- elimina(L, -7) => eroare

LLin – operații

- alKlea()

  - intrare:
    - L = $(e_0,\ldots, e_{n-1})$, k $\in$ **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)

- alKlea(L, 0) => a
- alKlea(L, 2) => c
- alKlea(L, 6) => g
- alKlea(L, 20) => eroare
- alKlea(L, -2) => eroare

`LLin` – operații

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

`elimTotE()` - exemple

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

- elimTotE(L, a) => (b, c, b, c)
- elimTotE(L, c) => (a, b, a, b, a)
- elimTotE(L, d) => (a, b, c, a, b, c, a)

LLin – operații

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

`parcurge()` - exemple

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

- parcurge(L, oriDoi()) => (2, 4, 6, 2, 4, 6)

- parcurge(L, incrementeaza()) =>
  (2, 3, 4, 2, 3, 4)

`LLin` – operații

- poz()
  - intrare:
    - L = $(e_0,\ldots, e_{n-1})$, e $\in$ **`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)

- poz(L, a) => 0
- poz(L, c) => 2
- poz(L, d) => 6
- poz(L, e) => -1

LLin – operații

- lung()
  - intrare:
    - L = (e_0,…, e_{n-1})
  - ieșire:
    - n – lungimea listei L

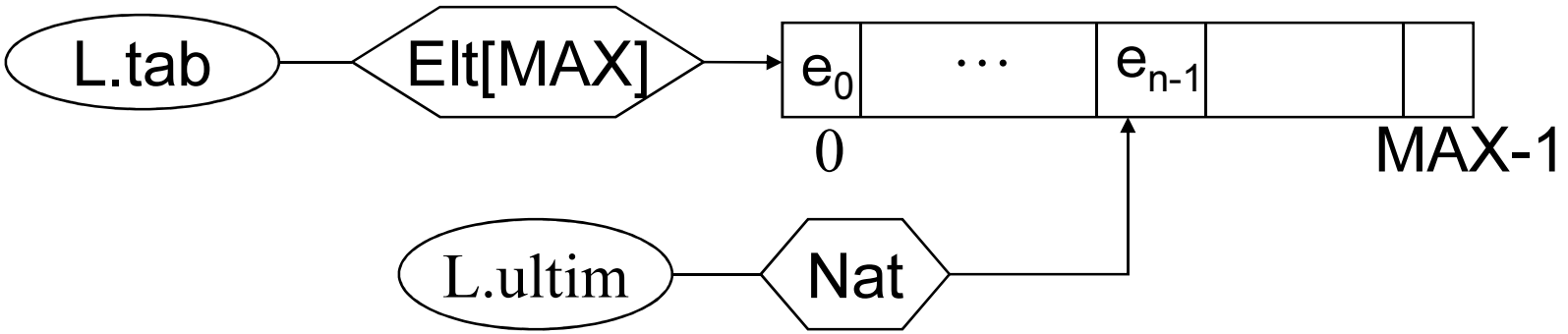
- Exemplu:

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

lung(L) => 7

LLin: implementare cu tablouri

- reprezentarea obiectelor L = (e_0,…, e_{n-1})



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

LLin: implementare cu tablouri

- insereaza()
  - deplasează elementele de pe poziţiile k, k+1, … la dreapta cu o poziţie
  - inserează e pe poziţia k
  - excepţii:
    - k<0, k > L.ultim+1 (n)
    - L.ultim = 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

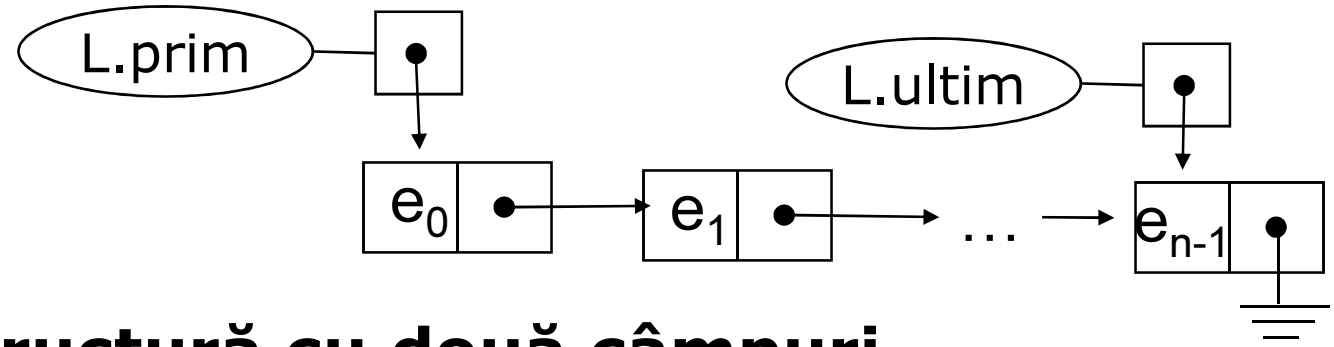
- parcurge()

```
procedure parcurge(L, viziteaza())
begin
   for i ←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)

LLin: implementare cu structuri înlănțuite

- reprezentarea obiectelor L = (e_0,..., e_{n-1})

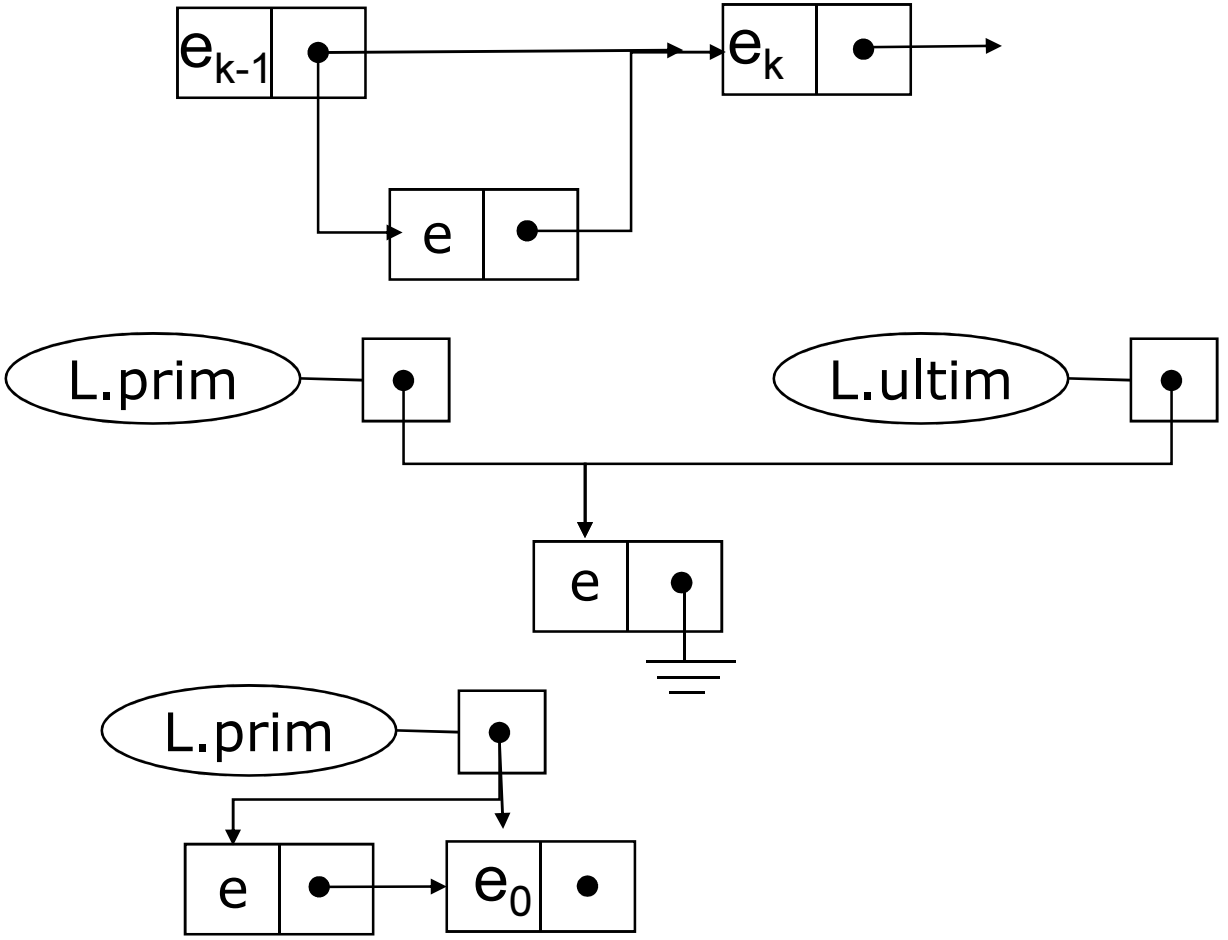


- **L structură cu două câmpuri**
  - pointer la primul element
  - pointer la ultimul element
- **un nod *p** (aflat la adresa din p) **are două câmpuri:**
  - unul pentru memorarea informaţiei: p->elt = e_i
  - unul pentru nodul succesor: p->succ

LLin: implementare cu structuri înlănțuite

- insereaza()
  - parcurge elementele 0, 1,…, k-1
  - inserează un nou element după k-1
    - creează nodul
    - memorează informaţii
    - reface legături
  - excepţii
    - 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 si 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
```

  - atenţie: timpul de execuţie 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
  - $L = (e_0,\ldots, e_{n-1})$, $n \geq 0$, $e_i \in$ **`Elt`**, $e_0 <\ldots< e_{n-1}$
- operaţii
  - listaVida()
    - intrare: nimic
    - iesire
      - () (lista cu zero elemente)
  - insereaza()
    - intrare:
      - $L = (e_0,\ldots, e_{n-1})$, $e \in$ **`Elt`**
    - ieșire
      - $L = (\ldots e_{k-1}, e, e_k,\ldots)$ dacă $e_{k-1} \leq e \leq e_k$
        $(e_{-1} = -\infty , e_n = +\infty)$

Liste liniare ordonate: `LLinOrd`

- elimina()
  - intrare:
    - L = (e_0,…, e_{n-1}), e ∈ **`Elt`**
  - ieșire
    - L = (…e_{k-1}, e_{k+1}…) dacă e = e_k
    - eroare în caz contrar
- alKlea()
- parcurge()
- poz()

LLinOrd: implementare cu tablouri

- **căutare binară**

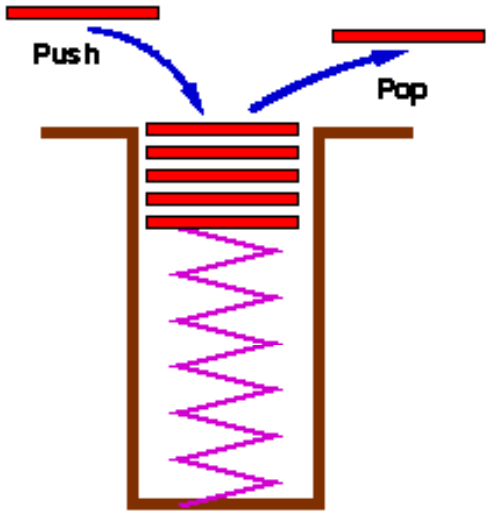
```
function Poz(L, e)
begin
    p ← 0; q ← L.ultim
    m ← [(p+q)/2]
    while(L.tab[m] != e && p < q) do
          if (e < L.tab[m])
          then q ← m - 1
          else p ← m + 1
          m ← [(p+q)/2]
    if (L.tab[m] =  e)
    then return m
    else return -1;
end
```

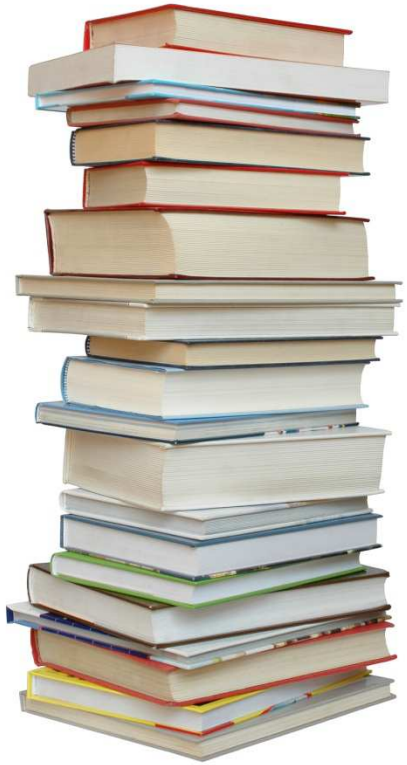
`LLinOrd`: complexitatea căutării

- implementare cu tablouri
  - timpul de execuţie O(log n)

- implementarea cu liste înlănţuite
  - timpul de execuţie 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 cunoaşte vechimea elementelor introduse (liste LIFO)
- operaţii
  - stivaVida()
    - intrare: nimic
    - iesire
      - S=(), lista vida
  - push()
    - intrare
      - S $\in$ **`Stiva`**, e $\in$ **`Elt`**
    - ieșire
      - S la care s-a adaugat e ca ultimul element introdus (cel cu vechimea cea mai mică)

Tipul abstract **`Stiva`**

- pop()
  - intrare
    - S ∈ **`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 ∈ **`Stiva`**
  - ieșire
    - **`true`** dacă S este vidă
    - **`false`** dacă S nu este vidă

Tipul abstract **`Stiva`**

- top()
  - intrare
    - S $\in$ **`Stiva`**
  - ieșire
    - ultimul element introdus (cel cu vechimea cea mai mică)
    - eroare dacă S este vidă

Stiva: implementare cu liste

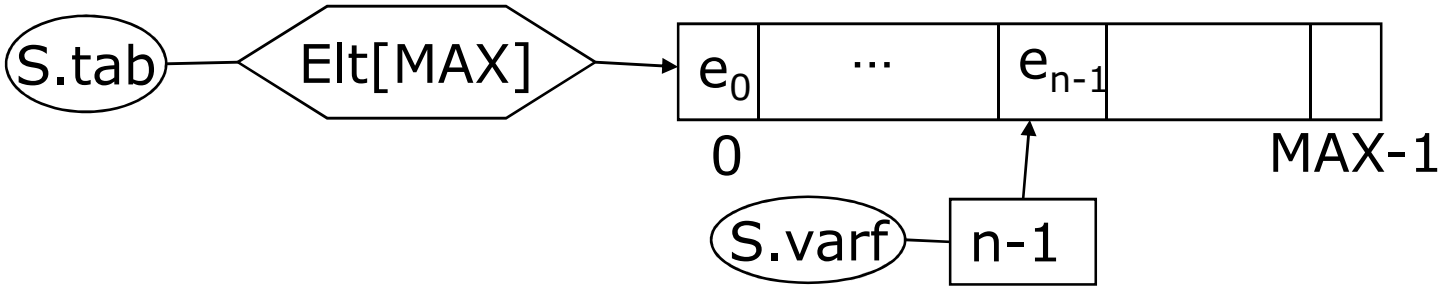
push(S, e) = insereaza(S, 0, e)
pop(S) = elimina(S, 0)
top(S) = alKlea(S, 0)

sau

push(S, e) = insereaza(S, lung(S), e)
pop(S) = elimina(S, lung(S)-1)
top(S) = alKlea(S, lung(S)-1)

Stiva: implementare cu tablouri

- reprezentarea obiectelor

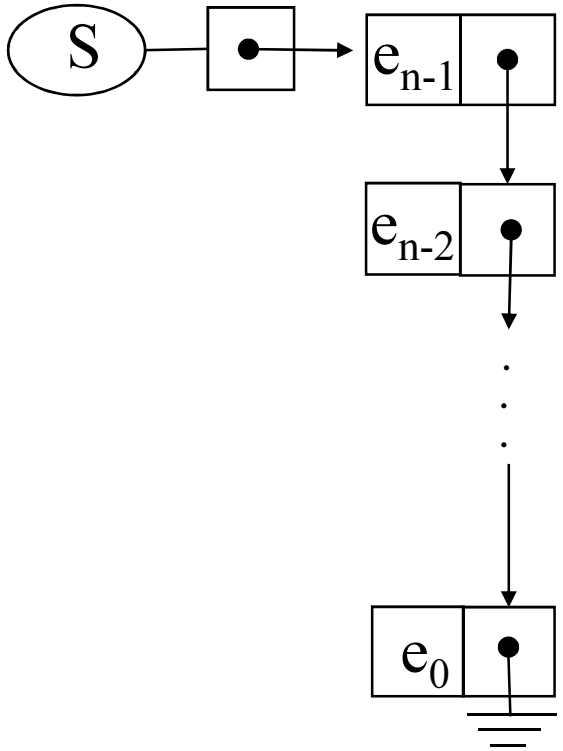


- implementarea operaţiilor
  - push()

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

Stiva: implementare cu structuri înlănțuite

- reprezentarea obiectelor



Stiva: implementare cu structuri înlănțuite

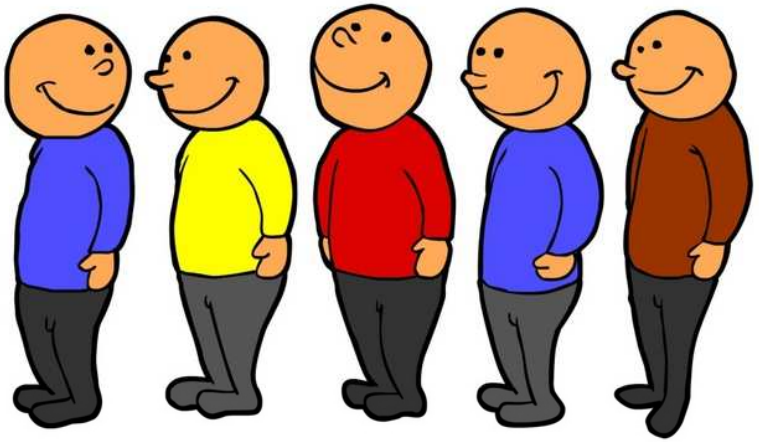
- implementarea operaţiilor
  - push()

```
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: imprimate).
- Aplicații indrecte
  - structură de date auxiliară în anumiți algoritmi.

Tipul abstract **Coada**

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

Tipul abstract **Coada**

- elimina()
  - intrare
    - C ∈ **Coada**
  - ieșire
    - C din care s-a eliminat primul element introdus (cel cu vechimea cea mai mare)
    - eroare dacă C este vidă
- esteVida()
  - intrare
    - C ∈ **Coada**
  - ieșire
    - **true** dacă C este vidă
    - **false** dacă C nu este vidă

Tipul abstract **Coada**

- citeste()
  - intrare
    - C $\in$ **Coada**
  - ieșire
    - primul element introdus (cel cu vechimea cea mai mare)
    - eroare dacă C este vidă

Coada: implementare cu liste

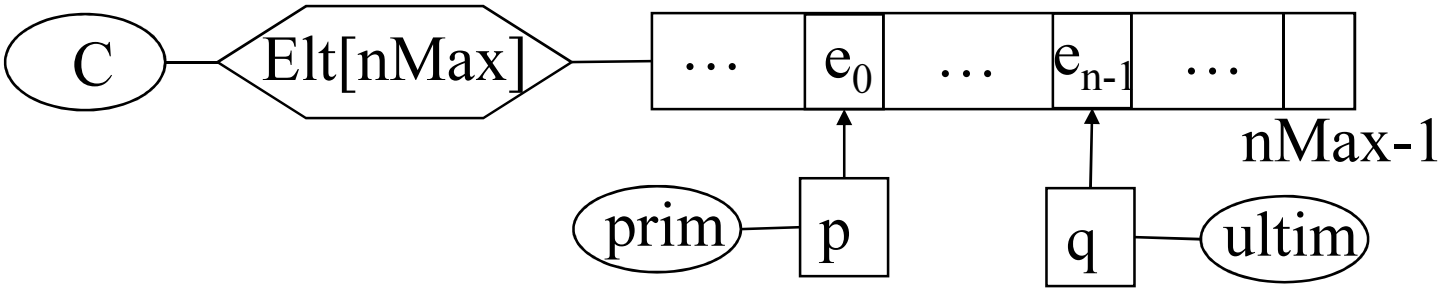
insereaza(C, e) = insereaza(C, lung(C), e)

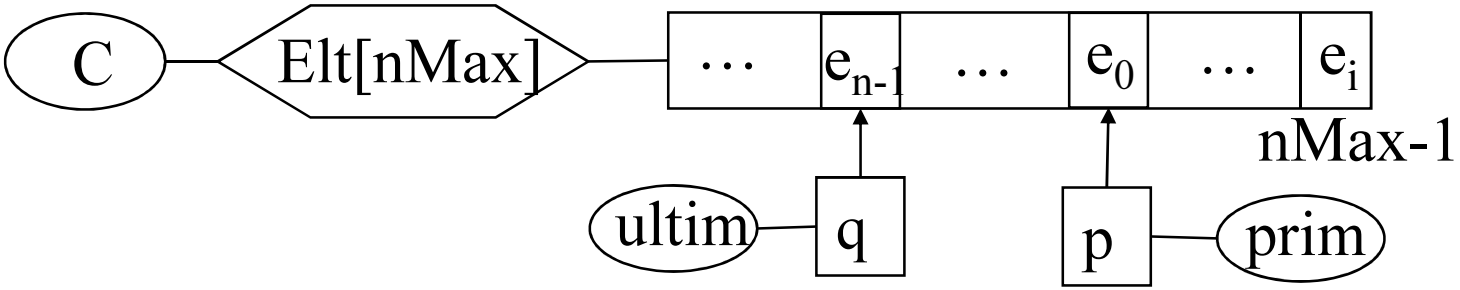
elimina(C) = elimina(C, 0)

citeste(C) = alKlea(C, 0)

Coada: implementare cu tablouri

- reprezentarea obiectelor





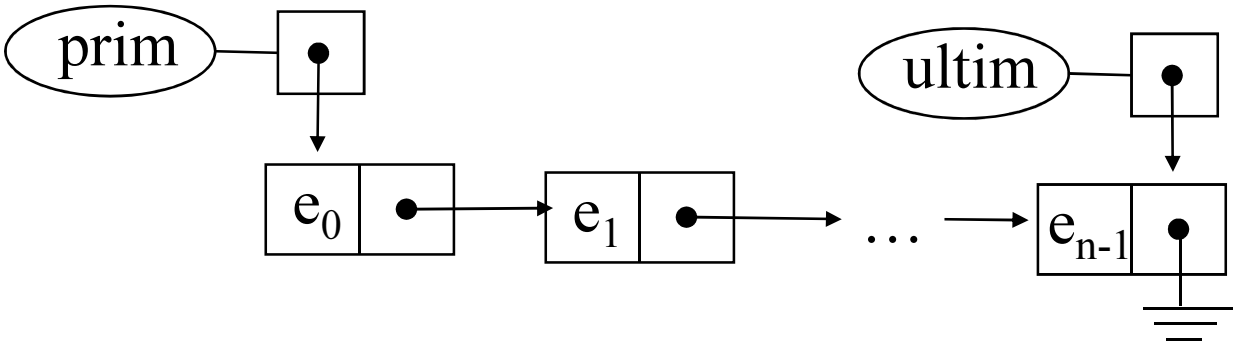
Coada: implementare cu tablouri

- implementarea operaţiilor
  - 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



Coada: implementare cu structuri înlănțuite

- implementarea operaţiilor
  - 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

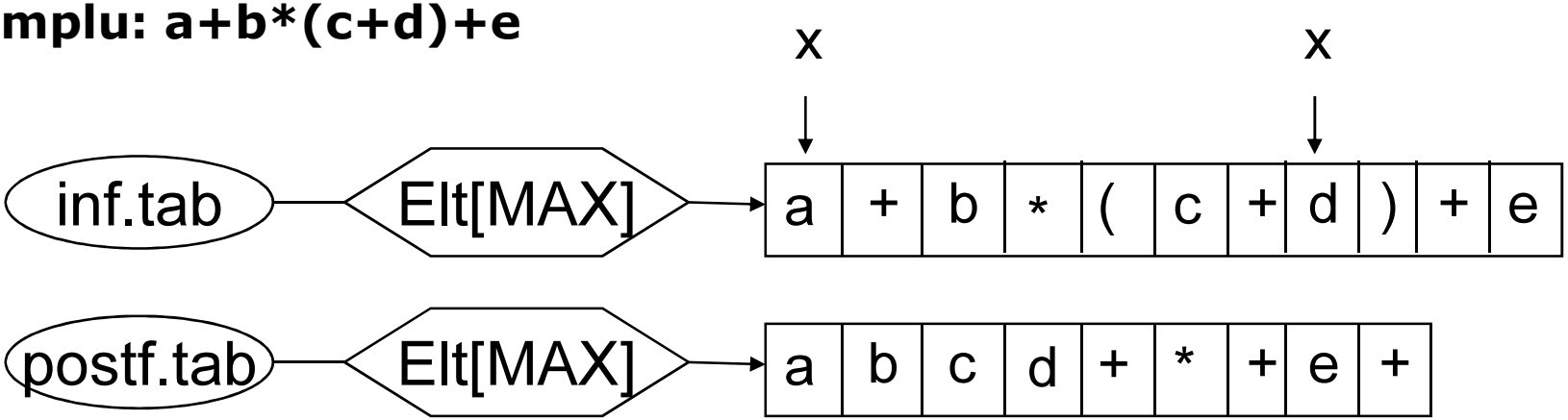
- notația infixată
  a + b
  a + (b * 2)
- notația postfixată
  a b +
  a b 2 * +
- reguli de precedență
  a + b * 2
- reguli de asociere
  7 / 3 * 2
  - la stânga (7 / 3) * 2
  - la dreapta 7 / (3 * 2)

Conversia infixat -> postfixat

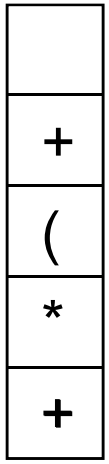
```
procedure convInfix2Postfix(infix, postfix)
/* infix si postfix sunt cozi */
begin
   S ← stivaVida()
   while (not esteVida(infix)) do
        citeste(infix, x); elimina(infix);
        if (x este operand) then insereaza(postfix,x)
        else if (x = ')')then
                while (top(s) ≠ '(')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

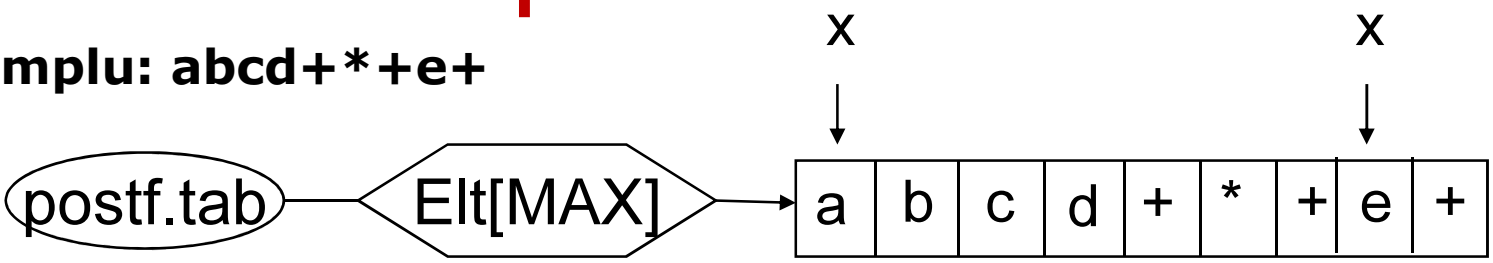


Evaluarea expresiilor postfixate

```
function valPostfix(postfix)
begin
   S ← stivaVida()
   while (!esteVida(postfix)) do
      citeste(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
```

Evaluarea expresiilor postfixate

Exemplu: abcd+*+e+



S (stiva)