Facultatea de Matematica si Informatica
Universitatea din Bucuresti

Structuri de Date si Algoritmi

- suport de curs -

Dobrovat Anca - Madalina

An universitar 2019 – 2020
Semestrul I
Seriile 21 + 25

Curs 4

22/10/2019

Curs 4 - Cuprins

2. Structuri lineare in alocare dinamica (inlantuita)

Operatii pe liste: traversare, cautare, inserare, stergere.

Tipuri particulare de liste (cu nod marcaj, circulare, dublu inlantuite).

Aplicatii ale listelor: reprezentarea numerelor mari, reprezentari de polinoame.

Multiliste. Aplicatii: reprezentarea matricilor rare, reprezentari de grafuri.

Structuri lineare cu restrictii la intrare/iesire: stive si cozi. Aplicatii.

Liste liniare inlantuite

- alocate static si dinamic

Nodul contine informatia si indicele (adresa) urmatorului nod

Avantaj: operatiile de adaugare sau stergere sunt rapide

Dezavantaj:

- Accesul la un nod se face prin parcurgerea nodurilor precedente

- Indicele (adresa) nodului urmator ocupa memorie suplimentara

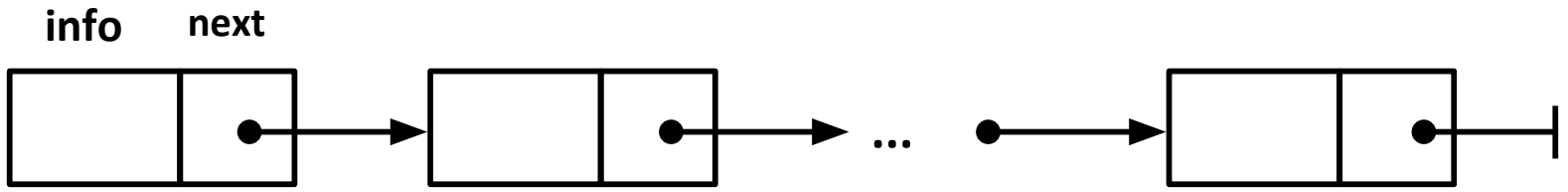
Liste liniare simplu inlantuite alocate dinamic

elementele listei s.n. noduri

fiecare nod conţine:

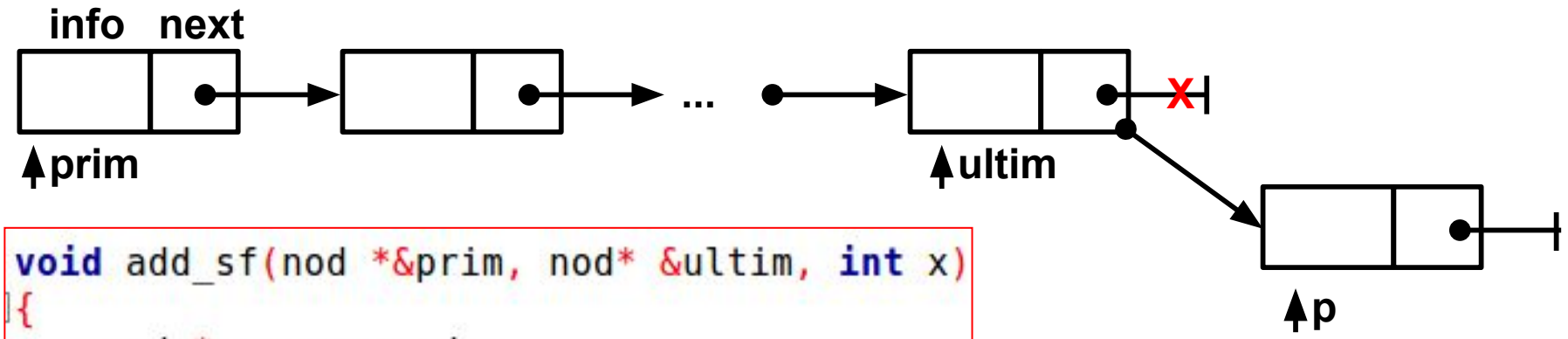
(1) un câmp, pe care se reprezintă un element al mulţimii;
în algoritmii care urmează putem presupune că elementul ocupă un singur câmp, *info*;

(2) un pointer către nodul următor, next.



Liste liniare simplu inlantuite alocate dinamic

Inserare (la sfarsitul listei)

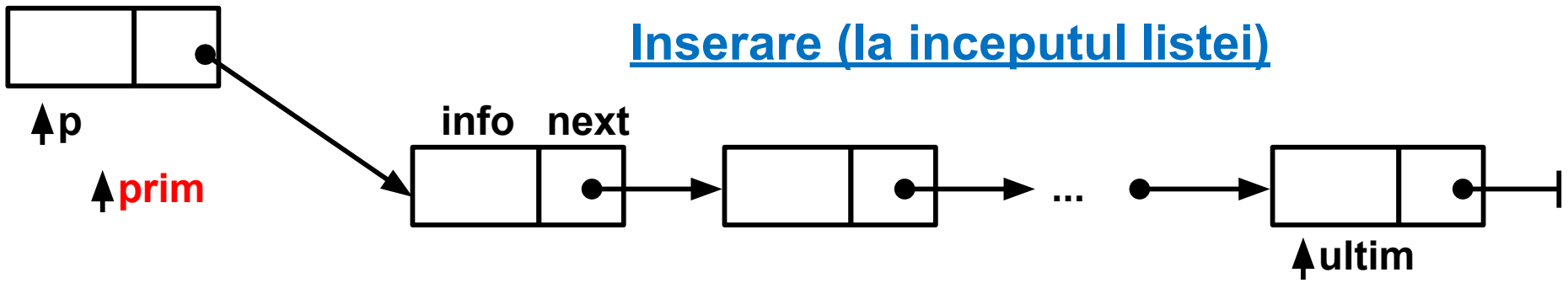



```
void add_sf(nod *&prim, nod* &ultim, int x)
{
    nod *p = new nod;
    p -> info = x;
    p -> next = NULL;
    if (prim==NULL)
    {
        prim = p;
        ultim = prim;
    }
    else
    {
        ultim -> next = p;
        ultim = p;
    }
}
```

```
struct nod
{
    int info;
    nod* next;
};
```

Liste liniare simplu inlantuite alocate dinamic

Inserare (la inceputul listei)




```
void add_inc(nod *&prim, int x)
{
    nod *p = new nod;
    p -> info = x;
    p -> next = prim;
    prim = p;
}
```

```
struct nod
{
    int info;
    nod* next;
};
```

Liste liniare simplu inlantuite alocate dinamic

Inserare (in interiorul listei) – pe o pozitie

```
void add_k(nod *&prim, nod *&ultim, int x, int k)
{
    // pozitionarea pointer-ului r pe pozitia k-1
    nod *r = prim;
    for (int i=0; i<k-1; i++) r=r->next;

    // inserarea pe pozitia k
    nod *p = new nod;
    p -> info = x;
    p -> next = r->next;
    r->next = p;
}
```

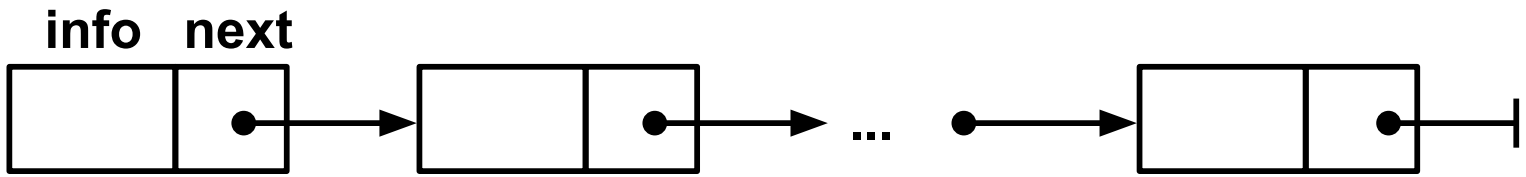
Liste liniare simplu inlantuite alocate dinamic

Inserare (in interiorul listei) – dupa o valoare

```
void add_val (nod *&prim, nod *&ultim, int x, int y)
{
    nod *r = prim;
    while (r && r->info!=y) r=r->next;
    if (r==NULL)
        add_sf(prim, ultim, x);     // valoarea y nu
    else
    {
        // inserarea dupa valoarea y
        nod *p = new nod;
        p -> info = x;
        p -> next = r->next;
        r->next = p;
    }
}
```

Liste liniare simplu inlantuite alocate dinamic

Traversare

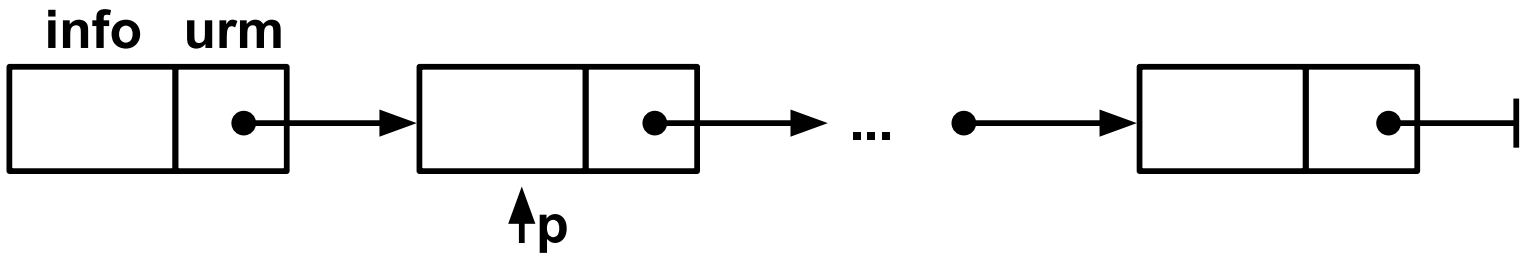



```
void afisare(nod * prim)
{
    nod * p = prim;
    while (p!=NULL)
    {
        cout<<p->info<<" ";
        p = p -> next;
    }
    cout<<endl;
}
```

```
adaugare la inceput
100
100 1 2 3

adaugare pe pozitia k - numerotare de la 0
200 2
100 1 200 2 3

adaugare dupa valoarea y
300 2
100 1 200 2 300 3
```

Liste liniare simplu inlantuite alocate dinamic

Cautare

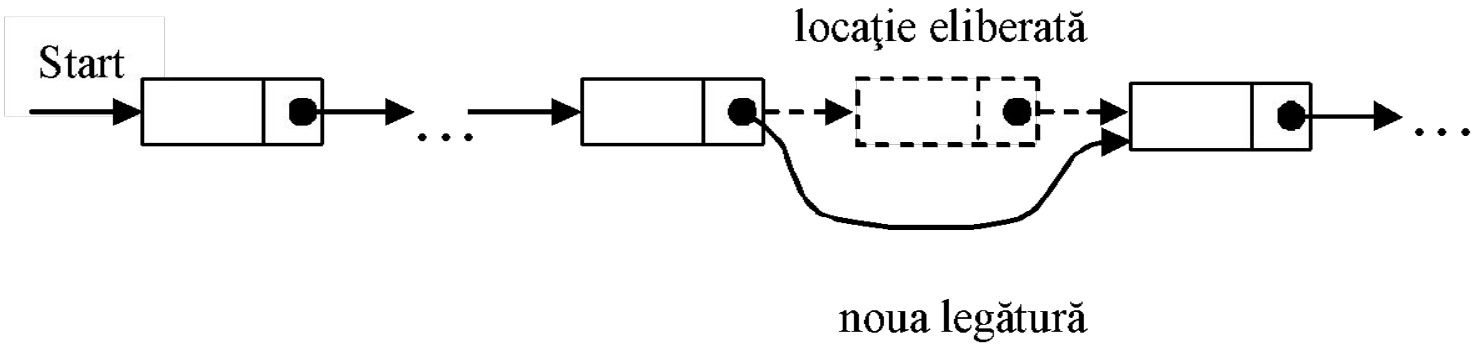


// prelucrare p → info;
p = p → urm;

```
nod *r = prim;
while (r && r->info!=y) r=r->next;
if (r==NULL)
```

Liste liniare simplu inlantuite alocate dinamic

Stergere

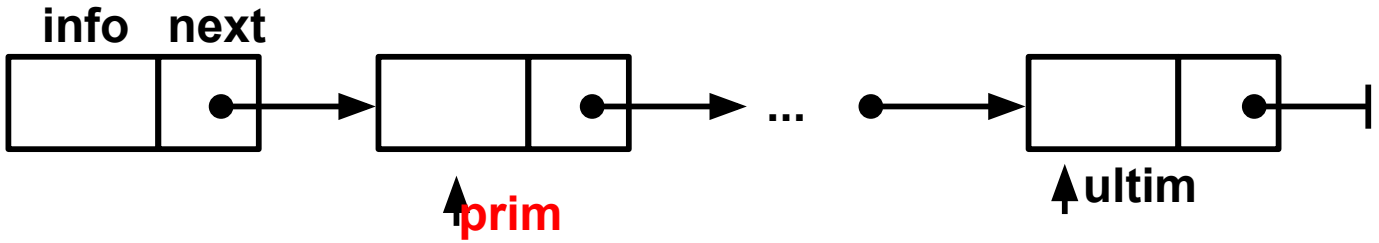


Refacerea structurii de lista simplu inlantuita pe nodurile ramase

Eventual dealocare de spatiu pt. Nodul extras (sau alte operatii cu el)

Liste liniare simplu inlantuite alocate dinamic

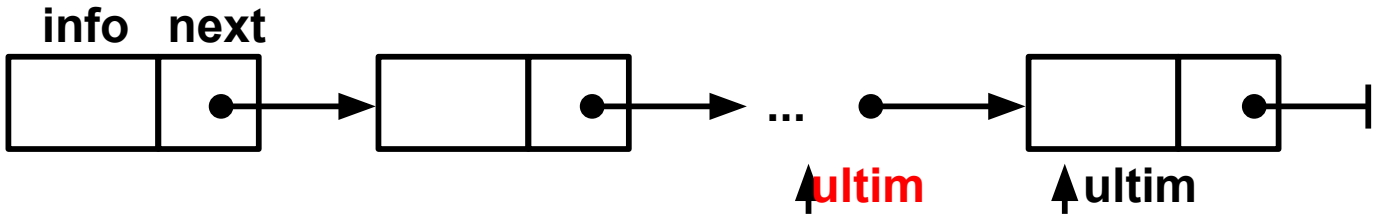
Stergerea primului element




```
void del_inc(nod*&prim)
{
    nod * t = prim;
    prim = prim -> next;
    delete t;
}
```

Liste liniare simplu inlantuite alocate dinamic

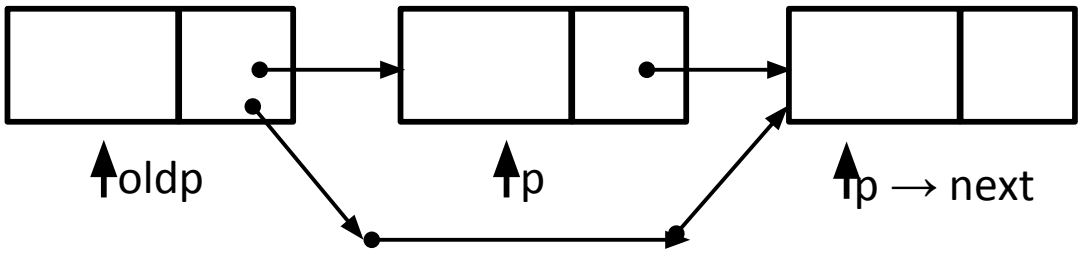
Stergerea ultimului element




```
void del_ultim(nod *&prim, nod *&ultim)
{
     nod *r;
     // pozitionarea pe penultimul element
     for (r = prim; r->next->next; r = r->next);
     // stergerea si reactualizarea listei
     nod * t = r -> next;
     r -> next = NULL;
     ultim = r;
     delete t;
}
```

Liste liniare simplu inlantuite alocate dinamic

Stergerea elementului de pe pozitia k




```
void del_k(nod *&prim, nod *&ultim, int k)
{
    // pozitionarea pointer-ului r pe pozitia k-1
    nod *r = prim;
    for (int i=0; i<k-1; i++) r=r->next;

    // stergere pe pozitia k
    nod *t = r-> next;
    r->next = r->next->next;
    delete t;
}
```

Liste liniare simplu inlantuite alocate dinamic

Stergere

```
cout<<endl<<"stergere prim element"<<endl;
del_inc(prim);n--;
afisare(prim);


cout<<endl<<"stergere ultim element"<<endl;
del_ultim(prim,ultim);n--;
afisare(prim);


cout<<endl<<"stergere de pe pozitia k - numerotare de la 0"<<endl;
cin>>k;
del_k(prim,ultim,k);n--;
afisare(prim);
```

```
stergere prim element
1 200 2 300 3

stergere ultim element
1 200 2 300

stergere de pe pozitia k - numerotare de la 0
1
1 2 300
```

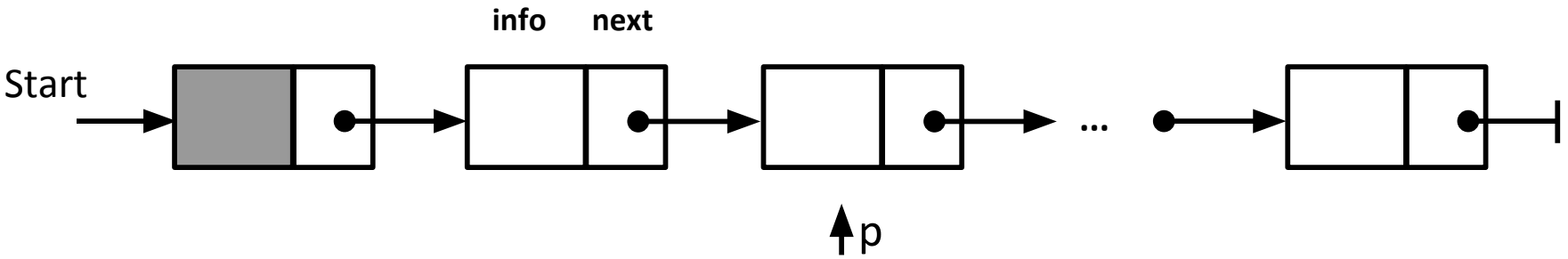
Liste liniare inlantuite alocate dinamic

Alte tipuri de liste

- cu nod marcaj
- circulare
- dublu inlantuite
- alte inlantuiri
  - liste de liste
  - masive

Liste cu nod marcaj

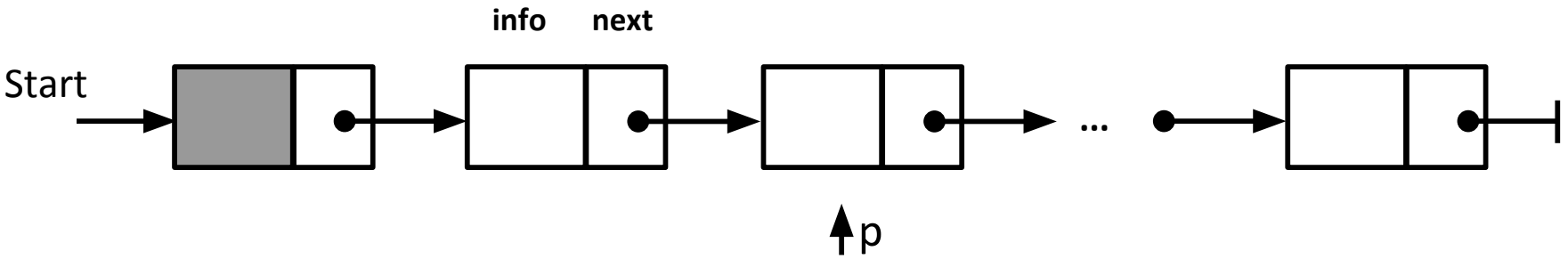
Traversare




```
nod *p;
 p = Start → next;
 while(p != NULL)
                {
                 // prelucrare p → info
                    p = p → next;
                 }
```

Liste cu nod marcaj

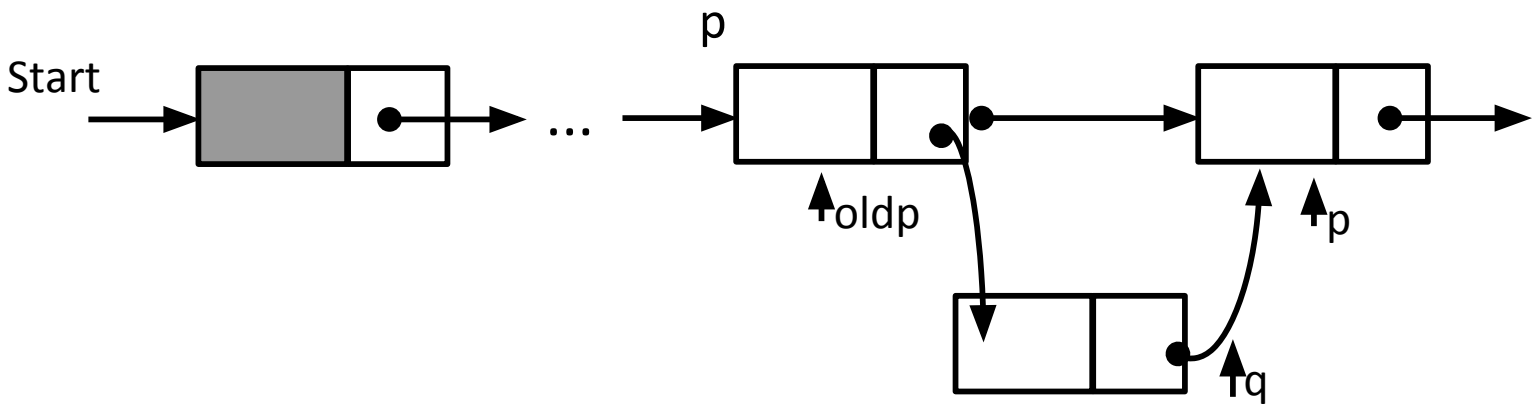
Cautare




```
nod *p;
p = Start → next;
while (p != NULL  &&  Val != p → info)
          p = p → next;
if (p == NULL) // cautare fara succes;
     else // gasit in p  Val == p → info;
```

Liste cu nod marcaj

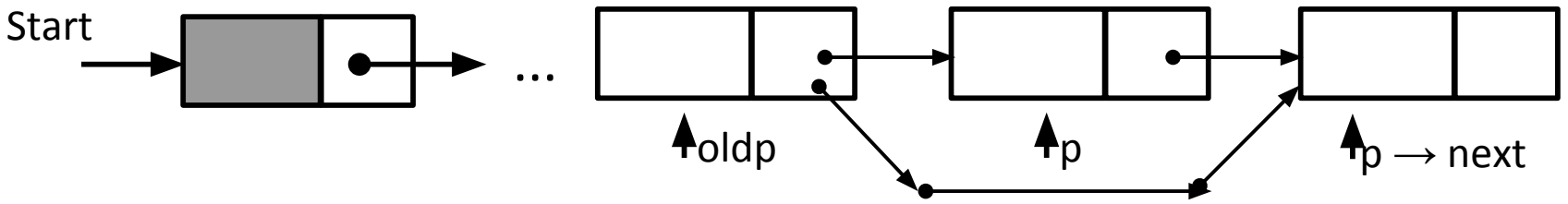
Inserare




```
nod *q = new nod ;
// prelucrare q → info;
      q → next = p;
      oldp → next = q;
```

Liste cu nod marcaj

Stergere




```
if (Start → next == NULL)
      // lista vida
else
{
      nod *temp = p;
      oldp → next = p → next;
      // prelucrare temp sau temp → info
      delete temp;
}
```

Liste circulare

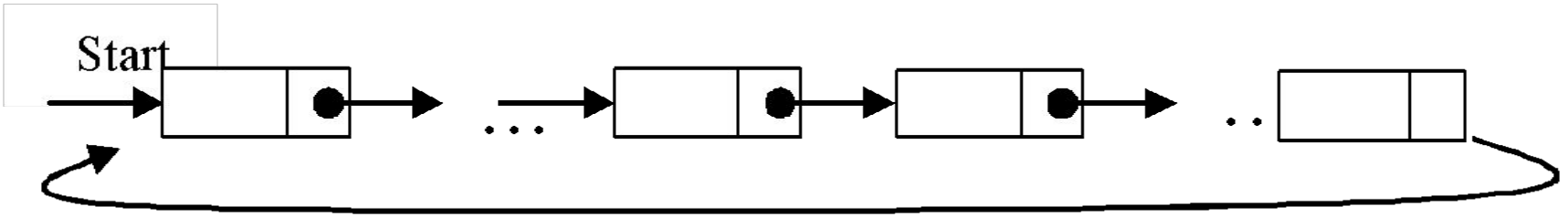


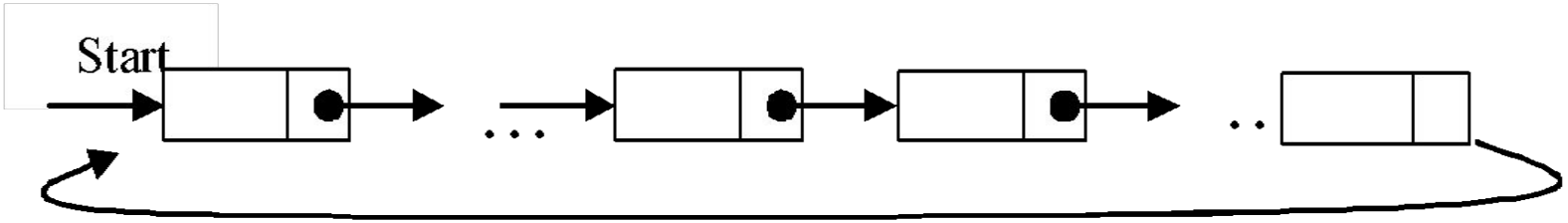
Fig.2.1.3. Listă circulară.

- utilă pentru aplicaţiile în care este nevoie să facem parcurgeri repetate ale listei

- testul de nedepăşire al structurii nu va mai fi de tipul $p \neq nil$

Liste circulare

Traversare

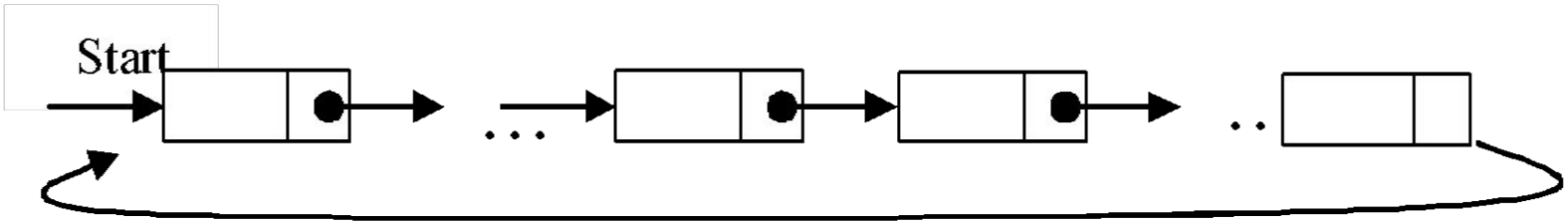


Exemplu – stergere duplicatelor dintr-o lista

```
nod *p;
 p = Start;
 // prelucrare p → info    primul element al listei
 p = p → next;
 while(p != Start)
                { // prelucrare p → info
                  p = p → next;
                 }
```

Liste circulare

Cautare




```
p = Start;
if (p → info == Val) // cautare cu succes
     else { p = p → next;
               while (p != Start  &&  Val != p → info)
                    p = p → next;
               if (p == Start) // cautare fara succes;
                    else // gasit in p       Val == p → info;
}
```

Liste circulare cu nod marcaj

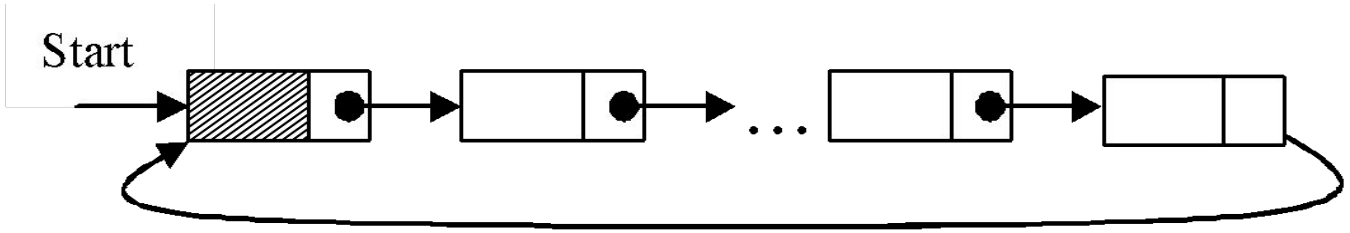


Fig.2.1.4. Listă circulară cu nod marcaj.

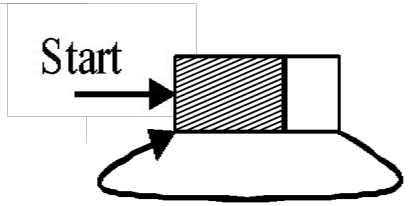


Fig.2.1.5. Listă circulară vidă cu nod marcaj.

- cautare: Se introduce valoarea căutată *Val* pe câmpul *info* al nodului marcaj cu *Start->info := Val*. Se începe căutarea în lista *Start->next*.

***Loc*= pointerul returnat de operaţia de căutare. Dacă *Loc* ≠ *Start* căutarea este cu succes, iar dacă *Loc* = *Start* căutarea este fără succes.**

Liste circulare cu nod marcaj

Traversare

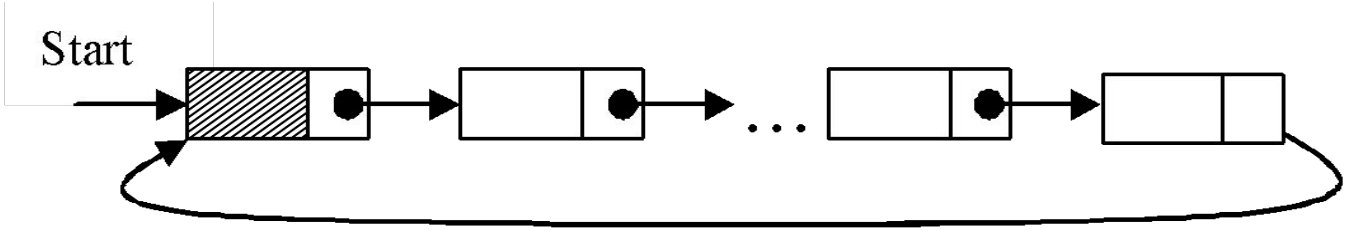


Fig.2.1.4. Listă circulară cu nod marcaj.

```
nod *p;
 p = Start → next;
 while(p != Start)
                {
                 // prelucrare p → info
                    p = p → next;
                 }
```

Liste circulare cu nod marcaj

Cautare

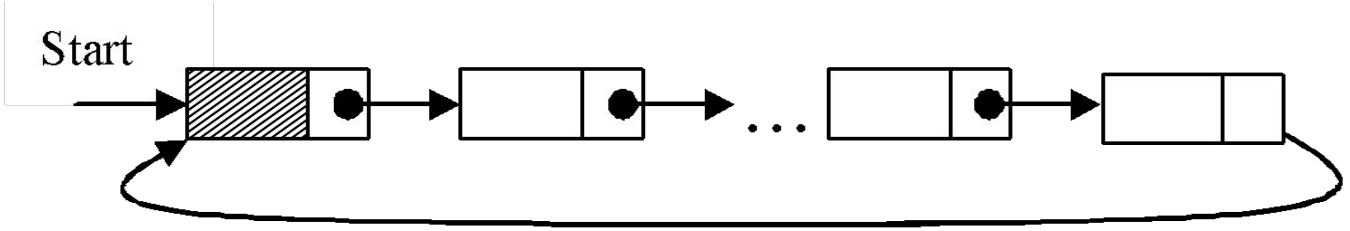


Fig.2.1.4. Listă circulară cu nod marcaj.

```
Loc = Start → next;
while (Loc → info != Val)
        Loc = Loc → next;
if (Loc == Start)   // cautare fara succes
    else // gasit in Loc
```

Liste circulare cu nod marcaj

Inserare

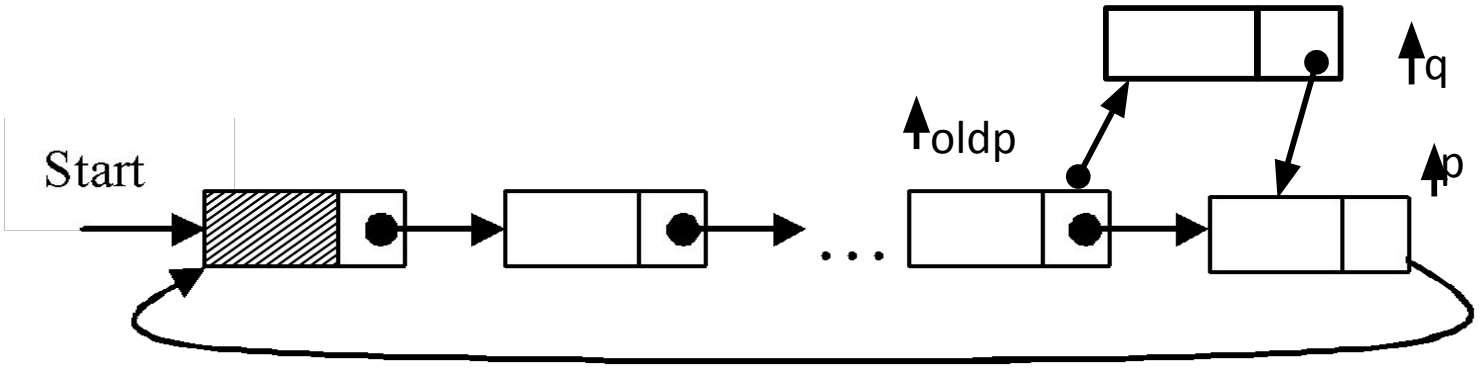
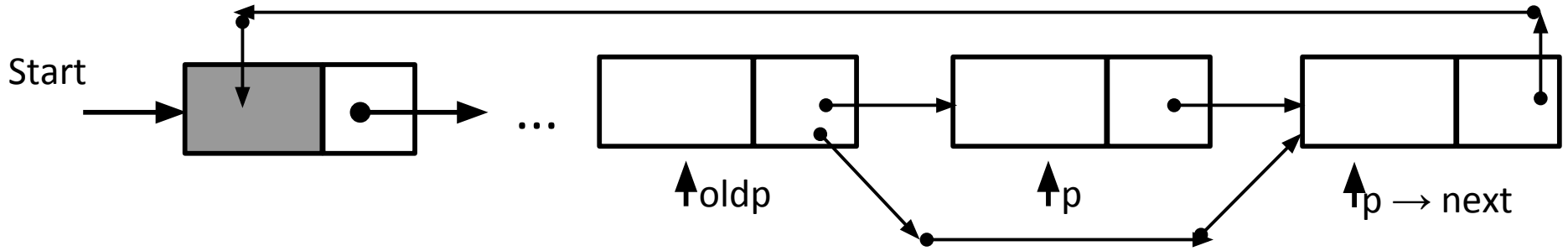


Fig.2.1.4. Listă circulară cu nod marcaj.

```
nod *q = new nod;
// prelucrare q → info;
     q → next = p;
     oldp → next = q;
```

Liste circulare cu nod marcaj

Stergere




```
if (Start → next == Start)
    // lista vida
else
{
    nod *temp = p;
    oldp → next = p → next;
    // prelucrare temp sau temp → info
    delete temp;
}
```

Liste liniare dublu inlantuite alocate dinamic

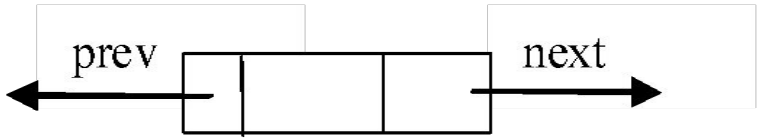


Fig.2.1.6. Nod într-o listă dublu înlănţuită.

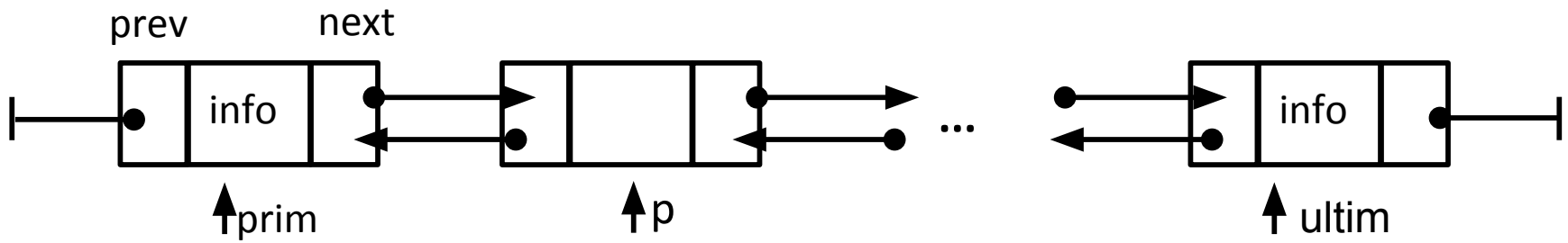
-- inserari/stergeri: parcurgerea cu cautarea locului se poate face cu un singur pointer

-- parcurgeri in ambele sensuri

-- ***cost***: locatii in plus !

Liste liniare dublu inlantuite alocate dinamic

Inserare (la sfarsitul listei)

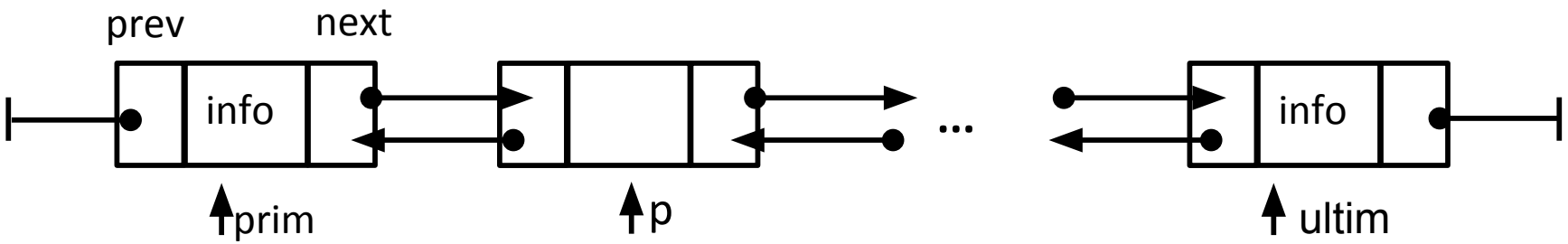



```
void add_sf(nod *&prim, nod* &ultim, int x)
{
    nod *p = new nod;
    p -> info = x;
    p -> next = NULL;
    p -> prev = NULL;
    if (prim==NULL)
    {
        prim = p;
        ultim = prim;
    }
    else
    {
        ultim -> next = p;
        p -> prev = ultim;
        ultim = p;
    }
}
```

```
struct nod
{
    int info;
    nod* next, *prev;
};
```

Liste liniare dublu inlantuite alocate dinamic

Inserare (la inceputul listei)




```
void add_inc(nod *&prim, int x)
{
    nod *p = new nod;
    p -> info = x;
    p -> next = prim;
    p -> prev = NULL;
    prim -> prev = p;
    prim = p;
}
```

```
struct nod
{
    int info;
    nod* next, *prev;
};
```

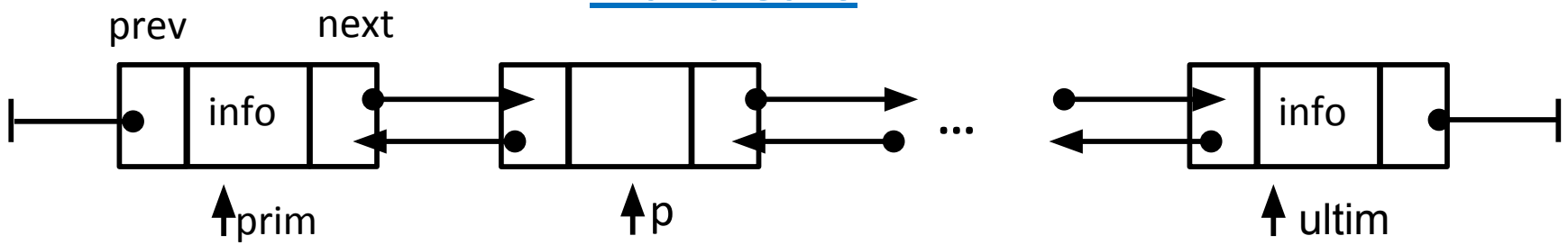
Liste liniare dublu inlantuite alocate dinamic

Inserare (in interiorul listei) – pe o pozitie

```
void add_k(nod *&prim, nod *&ultim, int x, int k)
{
    // pozitionarea pointer-ului r pe pozitia k-1
    nod *r = prim;
    for (int i=0; i<k-1; i++) r=r->next;

    // inserarea pe pozitia k
    nod *p = new nod;
    p -> info = x;
    p -> next = r->next;
    r -> next -> prev = p;
    p -> prev = r;
    r->next = p;
}
```

Liste liniare dublu inlantuite alocate dinamic

Traversare

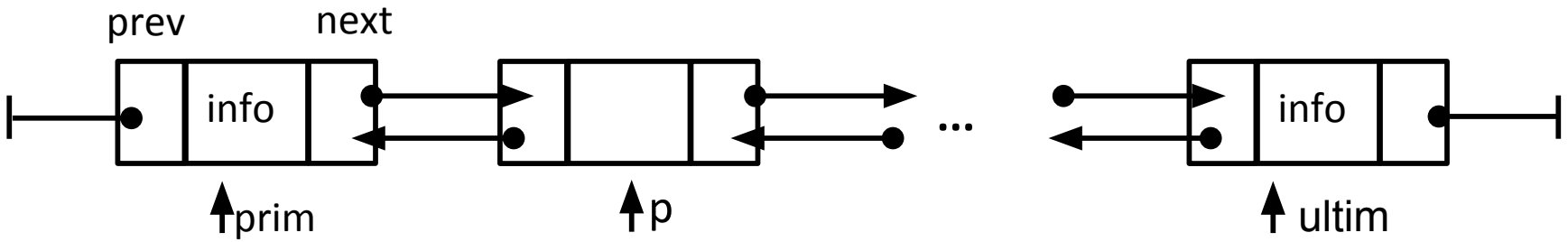



```
void afisare_st_dr(nod * prim)
{
    nod * p = prim;
    while (p!=NULL)
    {
        cout<<p->info<<" ";
        p = p -> next;
    }
    cout<<endl;
}
```

```
void afisare_dr_st(nod * ultim)
{
    nod * p = ultim;
    while (p!=NULL)
    {
        cout<<p->info<<" ";
        p = p -> prev;
    }
    cout<<endl;
}
```

Liste liniare dublu inlantuite alocate dinamic

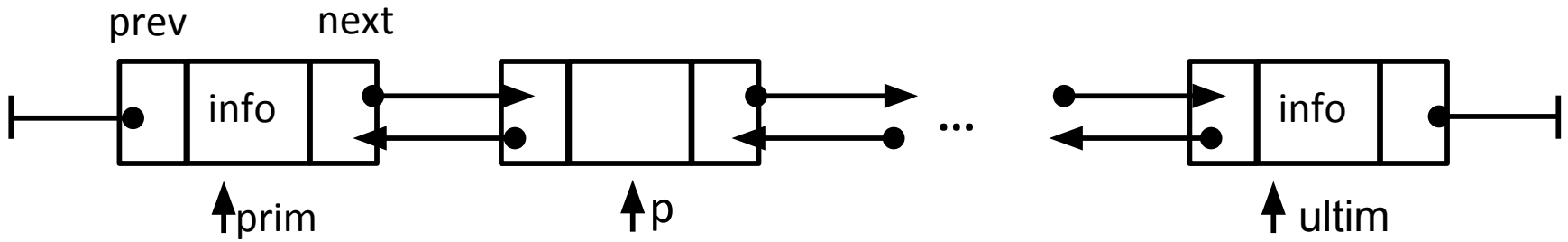
Stergerea primului element




```
void del_inc(nod*&prim)
{
    nod * t = prim;
    prim = prim -> next;
    prim -> prev = NULL;
    delete t;
}
```

Liste liniare dublu inlantuite alocate dinamic

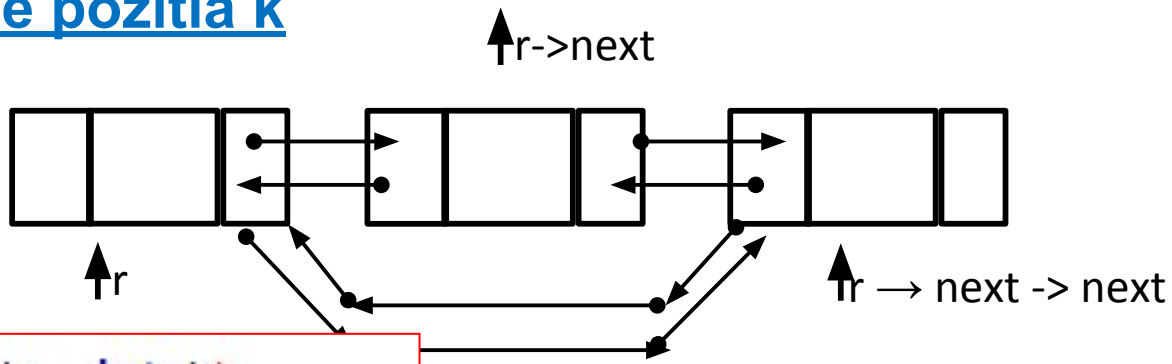
Stergerea ultimului element




```
void del_ultim(nod *&prim, nod *&ultim)
{
    // pozitionarea pe penultimul element
    ultim = ultim -> prev;

    // stergerea si reactualizarea listei
    nod * t = ultim -> next;
    ultim -> next = NULL;
    delete t;
}
```

Liste liniare dublu inlantuite alocate dinamic

Stergerea elementului de pe pozitia k




```
void del_k(nod *&prim, nod *&ultim, int k)
{
    // pozitionarea pointer-ului r pe pozitia k-1
    nod *r = prim;
    for (int i=0; i<k-1; i++) r=r->next;

    // stergere pe pozitia k
    nod *t = r-> next;
    r -> next -> next -> prev = r;
    r->next = r->next->next;

    delete t;
}
```

Liste liniare dublu inlantuite alocate dinamic

```
cout<<endl<<"adaugare dupa valoarea y"<<endl;
cin>>x>>y;
add_val(prim,ultim,x,y
afisare_st_dr(prim);
afisare_dr_st(ultim);


cout<<endl<<"stergere
del_inc(prim);
n--;
afisare_st_dr(prim);
afisare_dr_st(ultim);


cout<<endl<<"stergere
del_ultim(prim,ultim);
n--;
afisare_st_dr(prim);
afisare_dr_st(ultim);


cout<<endl<<"stergere
cin>>k;
del_k(prim,ultim,k);
n--;
afisare_st_dr(prim);
afisare_dr_st(ultim);
```

listeDuble

```
creare lista cu n elemente intregi
3
1
2
3
1 2 3
3 2 1

adaugare la inceput
100
100 1 2 3
3 2 1 100

adaugare pe pozitia k - numerotare de la 0
200 2
100 1 200 2 3
3 2 200 1 100

adaugare dupa valoarea y
300 2
100 1 200 2 300 3
3 300 2 200 1 100

stergere prim element
1 200 2 300 3
3 300 2 200 1

stergere ultim element
1 200 2 300
300 2 200 1

stergere de pe pozitia k - numerotare de la 0
1
1 2 300
300 2 1
```

Aplicatii

Reprezentarea vectorilor rari

- are cel putin 80% dintre elemente egale cu0.

- reprezentare eficienta → liste simplu inlantuite alocate dinamic

- fiecare nod din lista retine:

  - **valoarea**

  - **indicele din vector**

Cerinte: adunarea, respectiv, produsul scalar a doi vectori rari.

Aplicatii

Reprezentarea vectorilor rari - Implementare

```
void adauga(nod *&prim, nod *&ultim, int a, int b)
{   nod *q = new nod;
    q->val=a;    q->poz=b; q->next=NULL;

     if(prim==NULL)
     {    prim = q;
          ultim = prim;}
     else
     {    ultim -> next = q;
          ultim = q; }
}
```

```
struct nod
{
    int poz, val;
      nod*next;
};
```

```
void creare_vector(int &n, nod *&p, nod *&u)
{ int i,a,b;
cin>>n;
for(i=1;i<=n;i++)
    {cin>>a>>b;
    adauga(p, u, a, b);
    }
}
```

Aplicatii

Reprezentarea vectorilor rari - Implementare

```
void suma (nod *prim1, nod *prim2, nod *&prim3, nod *&ultim3)
{
    nod *p1, *p2;
    for (p1 = prim1; p1!= NULL; p1 = p1 -> next)
        adauga(prim3, ultim3, p1 -> val, p1 -> poz);

    for (p2 = prim2; p2!= NULL; p2 = p2 -> next)
    {   int ok = 0;
        for (p1 = prim3; p1!= NULL; p1 = p1 -> next)
            if (p2 -> poz == p1 -> poz) {p1 -> val += p2 -> val; ok = 1;}
        if (ok == 0) adauga(prim3, ultim3, p2 -> val, p2 -> poz);
        }
}
```

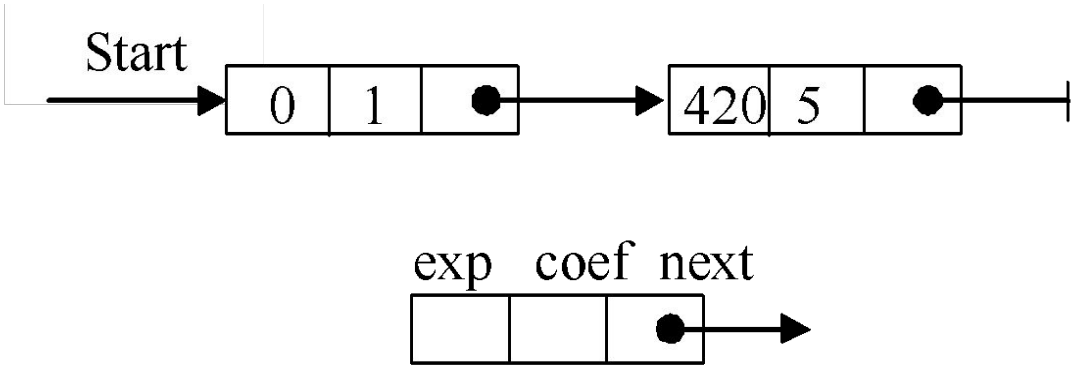
Aplicatii

Reprezentarea vectorilor rari - Implementare

```
int prod_scalar(nod *prim1, nod *prim2)
{   int prod = 0; nod *p1, *p2;

    for (p2 = prim2; p2!= NULL; p2 = p2 -> next)
       for (p1 = prim1; p1!= NULL; p1 = p1 -> next)
             if (p2 -> poz == p1 -> poz) prod += p1 -> val * p2 -> val;
return prod;
}
```

Aplicatii

Reprezentarea polinoamelor rare




```
struct nod
{
    int exp, coef;
     nod*next;
};
```

Cerinte: - evaluarea intr-un punct

- suma si produsul a doua polinoame.

Crearea polinoamelor si suma lor – analog vectori rari!

Aplicatii

Reprezentarea polinoamelor rare - Implementare

```
void produs(nod *prim1, nod *prim2, nod *&prim3, nod *&ultim3)
{
    nod *p1, *p2;
    for (p1 = prim1; p1!= NULL; p1 = p1 -> next)
        for (p2 = prim2; p2!= NULL; p2 = p2 -> next)
            { int a = p1 -> coef * p2 -> coef;
              int b = p1 -> exp + p2 -> exp;
              int ok = 0;
              for (nod* p3 = prim3; p3!= NULL; p3 = p3 -> next)
                if (p3 -> exp == b) {p3 -> coef += a; ok =1;}
              if (ok == 0)
                adauga(prim3, ultim3, a,b);
            }
}
```

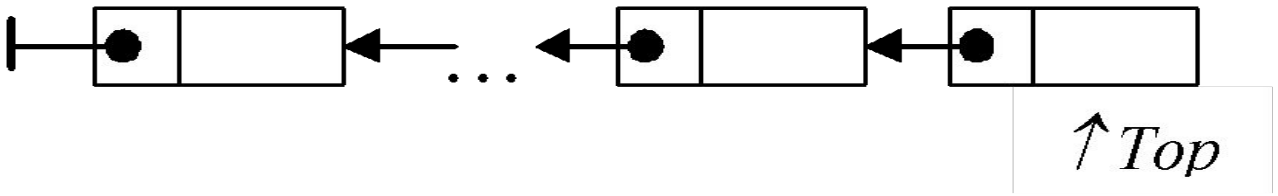
Aplicatii

Reprezentarea polinoamelor rare - Implementare

```
int eval(nod *prim, int x)
{
      // evaluarea unui polinom intr-un punct
    int prod = 0;
      nod *p;

    for (nod *p = prim; p!= NULL; p = p -> next)
        prod += p2 -> coef * pow(x,p -> exp);
return prod;
}
```

Stiva in alocare dinamica

C / C++

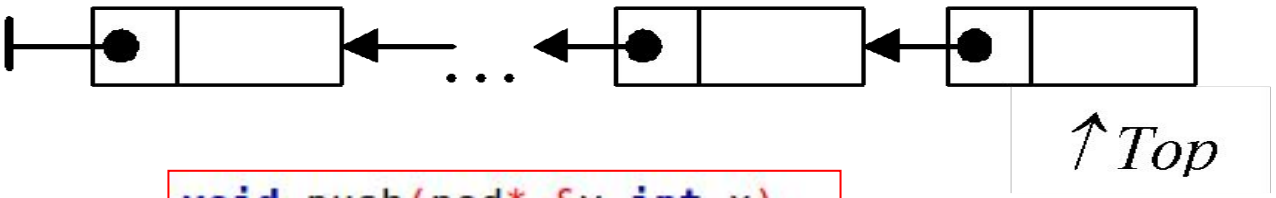



```
struct nod
{
    int info;
    nod *back;
};
```

Se refac operatiile de adaugare si stergere de la liste simplu inlantuite, respectand restrictiile!

Stiva in alocare dinamica

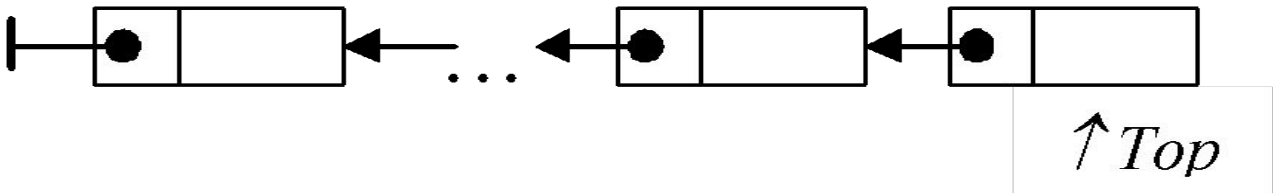
Inserare (Push)




```
void push(nod* &v,int x)
{
     if(!v)
     {
          v = new nod;
          v -> info = x;
          v -> back = NULL;
     }
     else
     {
          nod *t = new nod;
          t -> back = v;
          t -> info = x;
          v = t;
     }
}
```

Stiva in alocare dinamica

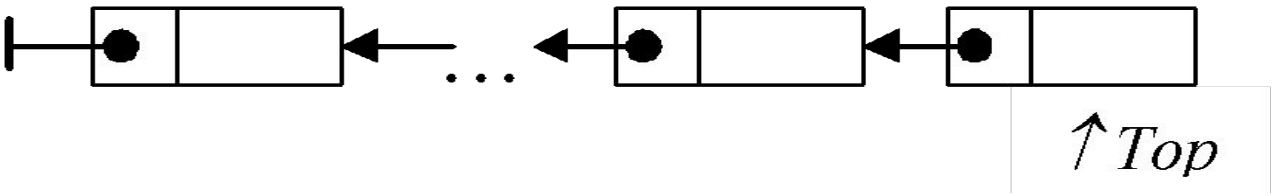
Afisare




```
void afisare(nod *v)
{
     nod *t = v;
     while(t)
     {
          cout<<t->info<<" ";
          t = t -> back;
     }
     cout<<endl;
}
```

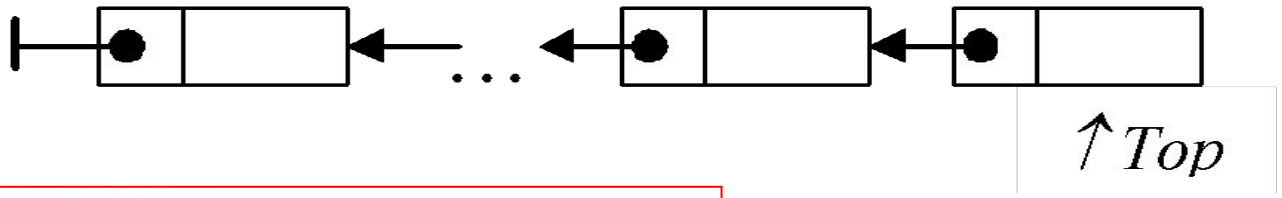
Stiva in alocare dinamica

Stergere (Pop)

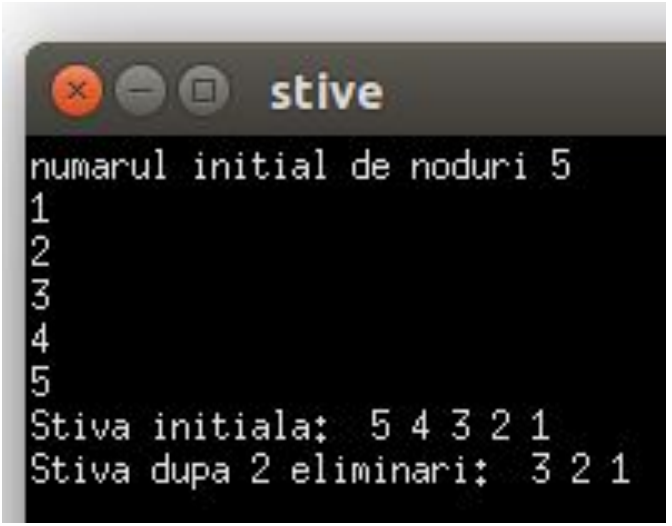



```
int pop(nod* &v)
{
    if(!v)
        cout<<"Stiva vida";
    else
    {
        nod *t = v;
        v = v -> back;
        delete t;
    }
}
```

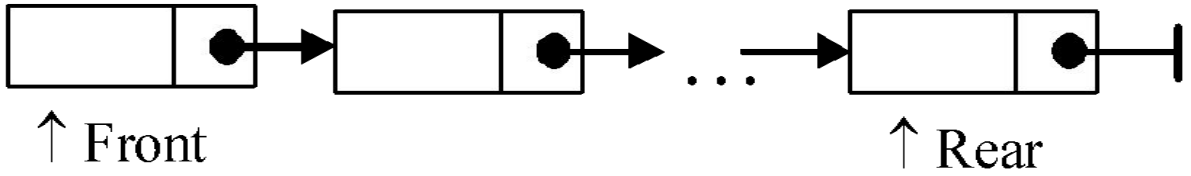
Stiva in alocare dinamica




```
nod *varf=NULL;
int n,x;
cout<<"numarul initial de noduri ";
cin>>n;
for(int i = 0; i < n; i++)
{
    cin>>x;
    push(varf,x);
}
cout<<"Stiva initiala:  ";
afisare(varf);

cout<<"Stiva dupa 2 eliminari:  ";
pop(varf);
pop(varf);
afisare(varf);
```



Coada in alocare dinamica



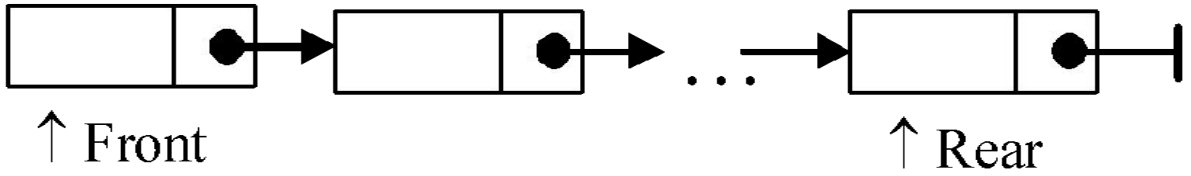
Inserari – *Rear*

Stergeri - *Front*

Coada vidă: *Front = Rear = NULL*.

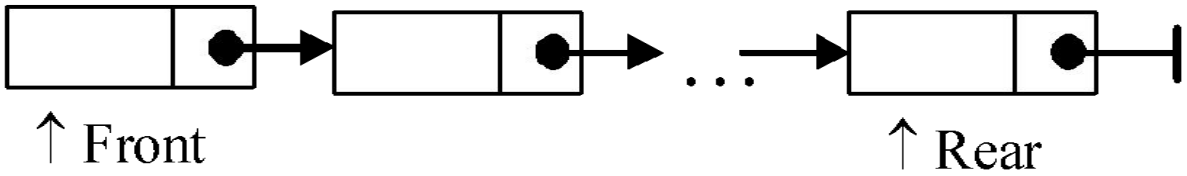
Coada cu un singur element: *Rear = Front != NULL*.

Coada in alocare dinamica



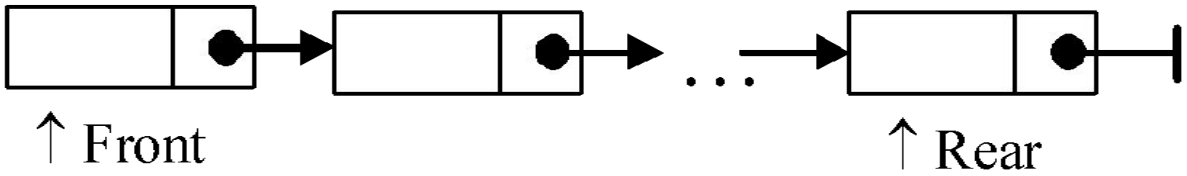

```
struct nod{
    int info;
    nod *next;
    };

nod * Front = NULL;
nod * Rear = NULL;
```

Coada in alocare dinamica




```
void Insert (nod *Front, nod *Rear, int Val)
{
      nod *p = new nod;
      if (p == NULL)      // Overflow
      else
      {
      p → info = Val;
      p → next = NULL;
      if (Rear == NULL)       // coada era vidă
                  Front = p;
      else
            Rear → next = p;
      Rear = p;
      }
}
```

Coada in alocare dinamica




```
void Delete (nod *Front, nod *Rear, int X)
{
      if (Front == NULL)      // Underflow
      else
{
//extragerea valorii, cu eliberarea spaţiului care a fost ocupat de nodul Front
      X = Front → info;
      p = Front;
      Front = Front → next;
      free(p);
      if (Front == NULL)   // coada avea un singur element, iar acum e vidă
            Rear = NULL;
      }
}
```

Alte tipuri de cozi

Coada cu priorităţi - Priority Queues

Elementele au, pe lângă cheie şi o prioritate:
- cea mai înaltă prioritate este 1, urmată de 2, etc.

Ordinea liniară este dată de regulile:
- elementele cu aceeaşi prioritate sunt extrase (şi procesate) în ordinea intrării;
- toate elementele cu prioritate *i* se află înaintea celor cu prioritate *i+1* (şi deci vor fi extrase înaintea lor).

Extragerile se fac dintr-un singur capăt.

Ca să se poată aplica regulile de mai sus la extragere, inserarea unui nou element cu prioritate *i* se va face la sfârşitul listei ce conţine toate elementele cu prioritate *i*.

Alte tipuri de cozi

DEQUE - Double Ended Queue

- structură liniară în care inserările şi ştergerile se pot face la oricare din cele două capete, dar în nici un alt loc din coadă.

În anumite tipuri de aplicaţii sau în modelarea anumitor probleme pot apare structuri de cozi cu restricţii de tipul:

- inserările se pot face la un singur capăt şi extragerile la amândouă.

Facultatea de Matematica si Informatica
Universitatea din Bucuresti

Curs 5

3. Structuri arborescente

Arbori oarecari. Definitii, terminologie, reprezentari, parcurgeri.

Arbori binari. Reprezentari, parcurgeri.

Arbori binari stricti. Proprietati matematice. Aplicatii.

Arbori binari de cautare. Operatii: cautare, inserare, stergere.

Algoritmul de cautare binara si performanta lui.

Arbori binari echilibrati AVL. Performanta cautarii in arbori binari de cautare echilibrati AVL.