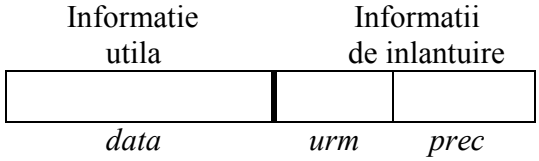
Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 4

Liste liniare dublu inlantuite. Liste circulare

1. Liste liniare dublu inlantuite

O ***lista*** este o colectie de elemente intre care este specificata cel putin o relatie de ordine "<". Daca relatia de ordine "<" este totala se poate construi o secventializare a elementelor listei astfel incit daca *a* si *b* sunt doua elemente consecutive in secventa atunci a<b si nu exista nici un alt element *c* in lista asfel incit *a*<*b*<*c*.
O ***lista liniara dublu inlantuita*** este caracterizata prin faptul ca pe multimea elementelor sunt definite doua relatii de ordine totala inverse una alteia (inainte si inapoi). Rezulta doua secventializari ale listei Ordinea elementelor pentru o astfel de lista este specificata prin doua cimpuri de informatie care sunt parte componenta a fiecarui element si indica elementul urmator si respectiv elementul precedent, conform cu relatiile de ordine definite pe multimea elementelor listei.

Deci fiecare element de lista dublu inlantuita are urmatoarea structura:



Pe baza informatiilor de inlantuire (pastrate in cimpurile ***urm*** si ***prec***) trebuie sa poata fi identificate urmatorul element din lista respectiv elementul precedent.

1.1. Liste liniare dublu inlantuite alocate static

Daca implementarea structurii de lista inlantuita se face prin *tablouri*, aceasta este o lista inlantuita alocata static sau simplu o ***lista inlantuita statica***.

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 4

Consideram urmatoarele declaratii:

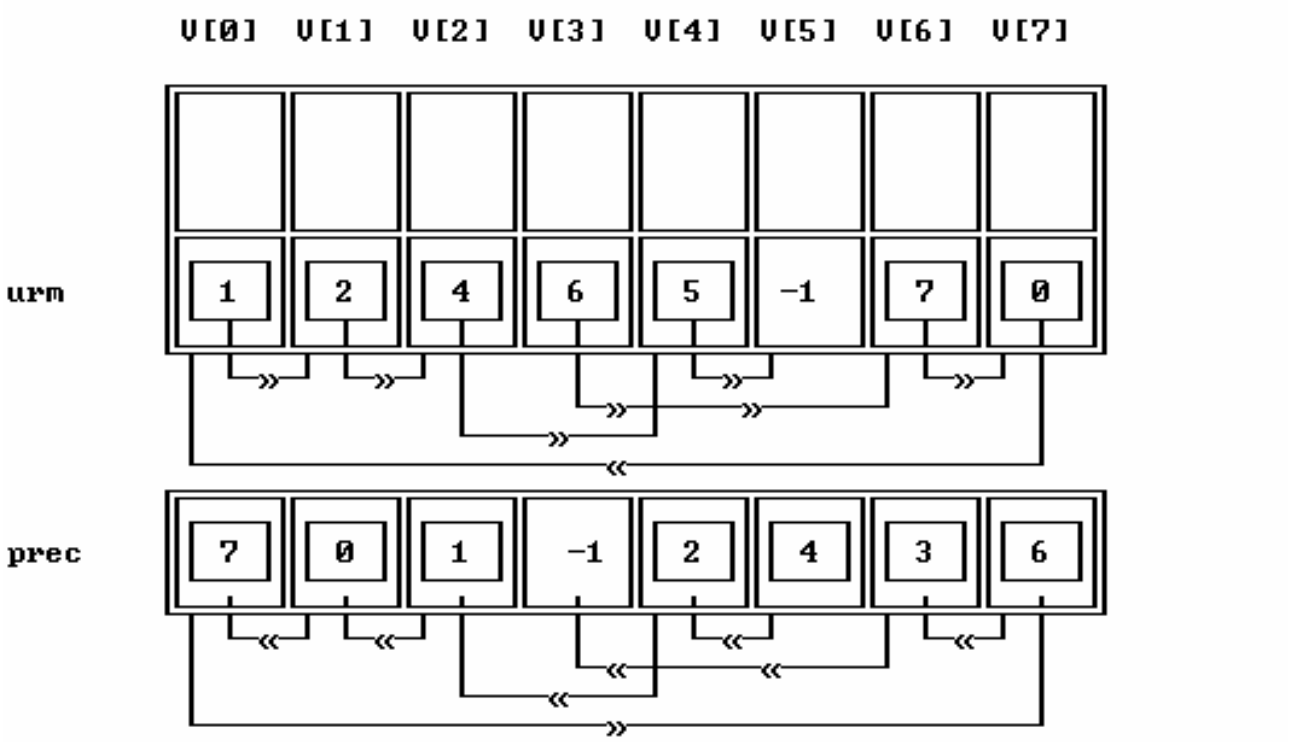
```
struct Element {
           char* data;
           int urm,prec;
           };

Element V[8];
```

Pentru elementele vectorului **V** exista o ordine naturala data de aranjarea in memorie a elemetelor sale: V[0], V[1], ... V[7]. Vom reperezenta memoria ocupata de vectorul **V** astfel incit fiecare element sa fie reprezentat vertical, in felul urmator:

	0	1	2	3	4	5	6	7
data								
urm								
prec								

Completand cimpurile ***urm*** si ***prec*** pentru fiecare element al vectorului se obtine o lista liniara dublu inlantuita. Valoarea cimpului ***urm*** va fi indexul in vector al urmatorului element din lista iar valoarea cimpului ***prec*** va fi indexul in vector al elementului precedent din lista.

Vectorul V:



Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 4

Pe baza inlantuirii stabilita de valorile din figura de mai sus se obtin:

secventa inainte: V[3], V[6], V[7], V[0], V[1], V[2], V[4], V[5] si
secventa inapoi : V[5], V[4], V[2], V[1], V[0], V[7], V[6], V[3].

Obs. Ultimul element din lista are in cimpul de legatura valoarea -1.

Este necesar sa cunoastem care este primul element in cele doua inlantuiri. Pentru aceasta retinem in doua variabile:

```
int cap1,cap2;
```

indexul primului element din fiecare inlantuire:

```
cap1=3.
cap2=5.
```

Parcurgerea in ordinea inainte a elemntelor listei se face in felul urmator:

```
int crt;
.................
crt = cap1;
while (crt!=-1) {
                ┌──────────────────────┐
                │ Prelucreaza V[crt]   │
                └──────────────────────┘
                crt = V[crt].urm;
                }
```

Parcurgerea in ordinea inapoi a elementelor listei se face in felul urmator:

```
int crt;
.................
crt = cap2;
while (crt!=-1) {
                ┌──────────────────────┐
                │ Prelucreaza V[crt]   │
                └──────────────────────┘
                crt = V[crt].prec;
                }
```

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 4

1.2. Liste liniare dublu inlantuite alocate dinamic

Daca implementarea structurii de lista inlantuita se face prin tehnici de alocare dinamica se obtine o lista inlantuita alocata dinamic sau simplu o lista inlantuita dinamica.

Iata cum arata declaratiile tipului "element de lista" in C++:

```
struct Element {
       TipOarecare data;          // informatia utila
       Element* urm,prec;         // legatura
       };
```

In C va trebui sa scriem:

```
typedef struct _Element {
     TipOarecare data;
     struct _Element* urm,prec;
     } Element;
```

Avind declaratiile de mai sus (una din forme), si

```
Element* p;        // un pointer la Element
```

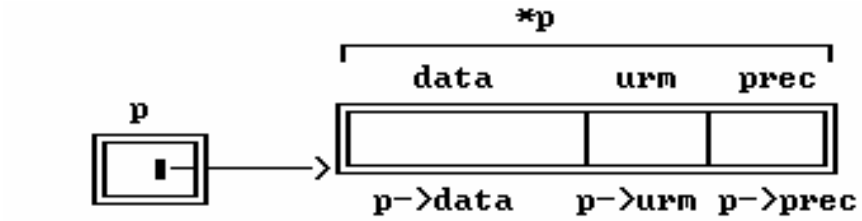
in urma unei operatii:

```
p = (Element*) malloc( sizeof(Element) );     // in C
```

sau, simplu

```
p = new Element;                              // in C++
```

p a fost initializat cu adresa unei variabile de tip ***Element*** alocata in zona de alocare dinamica:



Atunci, aceasta din urma va fi identificata prin expresia ***p** iar cimpurile sale prin expresiile **p->data** si respectiv **p->urm**, **p->prec**.

Constanta **0** (**NULL**) pentru un pointer inseamna o adresa imposibila. Aceasta valoare va fi folosita pentru a sfirsi inlantuirea (ultimul element din lista va avea **p->urm=0** iar primul element din lista va avea **p->prec=0**).

1.3. Operatii in liste liniare dublu inlantuite

O lista liniara dublu inlantuiata poate fi privita ca o lista liniara simplu inlantuiata daca se face abstrcatie de una din inlantuiri. Astfel operatiile cu astfel de liste au implementari similare celor corespunzatoare de la listele liniare simplu inlantuite. Operatiile de insertie sau stergere la sfarsitul listei pot fi simplificate utilizand cealalta inlantuire pentru a accesa ultimul element (ultimul element intr-o inlantuire este primul element in cealalta inlantuire). Operatiile de insertie sau stergere in interiorul listei pot fi optimizate alegand inlantuirea cea mai potrivita situatiei respective.

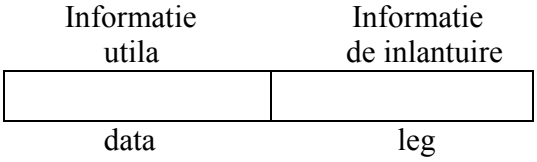
2. Liste circulare

O ***lista circulara simplu inlantuita*** este o lista liniara simplu inlantuita modificata astfel incat ultimul element pointeaza spre primul element din lista.

O ***lista circulara dublu inlantuita*** este o lista liniara dublu inlantuita modificata astfel incat ultimele elemente pointeaza respectiv spre primele elemente din lista.

In continuare ne vor referi la liste circulare simplu inlantuite pe care le vom numi simplu: *liste circulare*.

Deci fiecare element de lista circulara are urmatoarea structura:



Pe baza informatiei de inlantuire (pastrata in cimpul ***leg***) trebuie sa poata fi identificat urmatorul element din lista.

2.1. Lista circulara alocata static

Daca implementarea structurii de lista circulara se face prin tablouri se o lista circulara alocata static sau simplu o lista circulara statica.

Consideram urmatoarele declaratii:

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 4

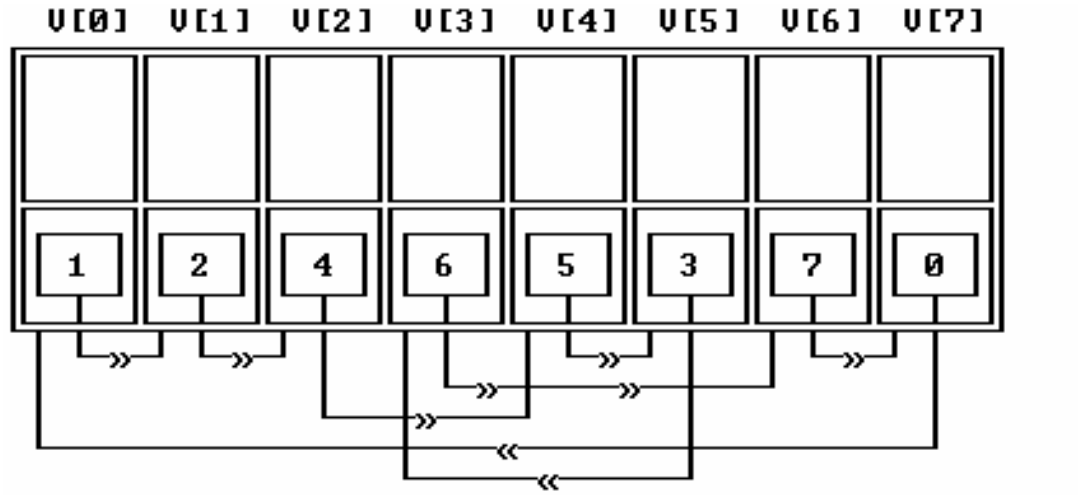
```
struct Element {
          char* data;
          int leg;
          };

Element V[8];
```

Pentru elementele vectorului **V** exista o ordine naturala data de aranjarea in memorie a elemetelor sale: V[0], V[1], ... V[7]. Vom reperezenta memoria ocupata de vectorul **V** astfel incit fiecare element sa fie reprezentat vertical, in felul urmator:

	0	1	2	3	4	5	6	7
data								
leg								

Completind cimpul ***leg*** pentru fiecare element al vectorului putem obtine o lista liniara simplu inlantuita. Valoarea cimpului ***leg*** va fi indexul in vector al urmatorului element din lista.

Vectorul V:



Pe baza inlantuirii stabilita de valorile din figura de mai sus se obtine secventa: V[3], V[6], V[7], V[0], V[1], V[2], V[4], V[5], V[3], V[4], ...

Obs. Nu mai exista un ultim element in lista ca la listele liniare.

Este necesar sa cunoastem care este primul element din inlantuire, pentru aceasta retinem intr-o variabila:

```
int cap;
```

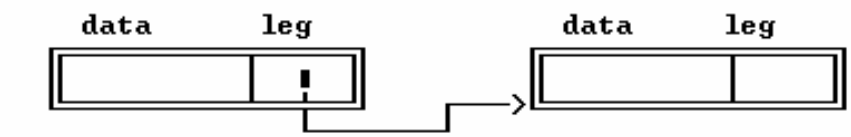
indexul primului element

cap=3.

Parcurgerea in ordine a elemntelor listei se face in felul urmator:

```
int crt;
.................
if (cap!=null){
               crt = cap;
               ┌──────────────────────┐
               │ Prelucreaza V[crt]   │
               └──────────────────────┘
               while (V[crt].leg!=cap) {
                      crt = V[crt].leg;
                      ┌──────────────────────┐
                      │ Prelucreaza V[crt]   │
                      └──────────────────────┘
                      }
               }
```

Indiferent de modul cum se materializeaza informatiile de legatura pentru a reprezenta o lista inlantuita vom folosi urmatoarea reprezentare:



Sageata care porneste din cimpul ***leg*** arata faptul valoarea memorata aici indica elementul la care duce sageata.

2.2. Liste circulare alocate dinamic

Daca implementarea structurii de lista circulara se face prin tehnici de alocare dinamica se obtine o lista circulara alocata dinamic sau simplu o ***lista circulara dinamica***.

Declaratiile tipului "*element de lista circulara*" in C++:

```
struct Element {
       TipOarecare data;          // informatia utila
       Element* leg;              // legatura
       };
```

In C va trebui sa scriem:

```
typedef struct _Element {
     TipOarecare data;
     struct _Element* leg;
     } Element;
```

Avind declaratiile de mai sus (una din forme), si

```
Element* p;          // un pointer la Element
```

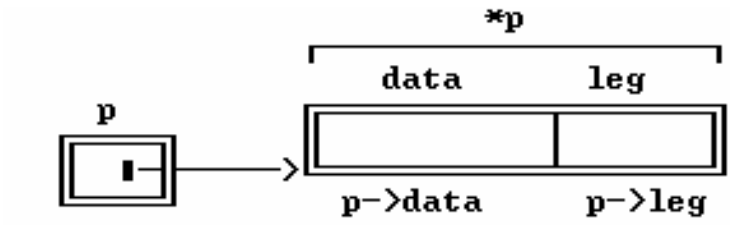
in urma unei operatii:

```
p = (Element*) malloc( sizeof(Element) );  // in C
```

sau, simplu

```
p = new Element;                                    // in C++
```

p a fost initializat cu adresa unei variabile de tip ***Element*** alocata in zona de alocare dinamica:



Atunci, acesta din urma va fi identificata prin expresia ***p** iar cimpurile sale prin expresiile **p->data** si respectiv **p->leg**.

Pentru a manevra o lista avem nevoie doar de un pointer la primul element al listei. Pentru a indica o lista vida acest pointer va primi valoarea 0 (NULL).

2.3. Operatii in liste circulare

Fara a restringe generalitatea, vom detalia doar implementarea prin pointeri.

Consideram declaratiile de tipuri de mai sus si variabilele:

```
Element* cap;     // pointer la primul element al unei liste
Element* p;
Element* q;
```

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 4

Operatiile primitive pentru acces la o lista inlantuita sint:

2.3.1. Parcurgerea listei

Consideram: **cap** - contine adresa primului element din lista.

O parcurgere inseamna prelucrarea pe rind a tuturor elementelor listei, in ordinea in care acestea apar in lista. Vom avea o variabila pointer **p** care va indica pe rind fiecare element al listei:

```
if (cap!=0){
           p = cap;
           ┌──────────────────────┐
           │ Prelucreaza p->data  │
           └──────────────────────┘
           while (p->leg->!=cap){
                                p = p->leg;
                                ┌──────────────────────┐
                                │ Prelucreaza p->data  │
                                └──────────────────────┘
                                }
           }
```

Un caz special apare atunci cind dorim sa facem o parcurgere care sa se opreasca in fata unui element care sa indeplineasca o conditie (ca in cazul cind inseram un element intr-o pozitie data printr-o conditie, sau stergem un element care indeplineste o conditie).

Presupunem ca lista are cel putin un element.

```
p = cap;
while (p->leg!=cap && !conditie(p->leg))
                    p = p->leg;
```

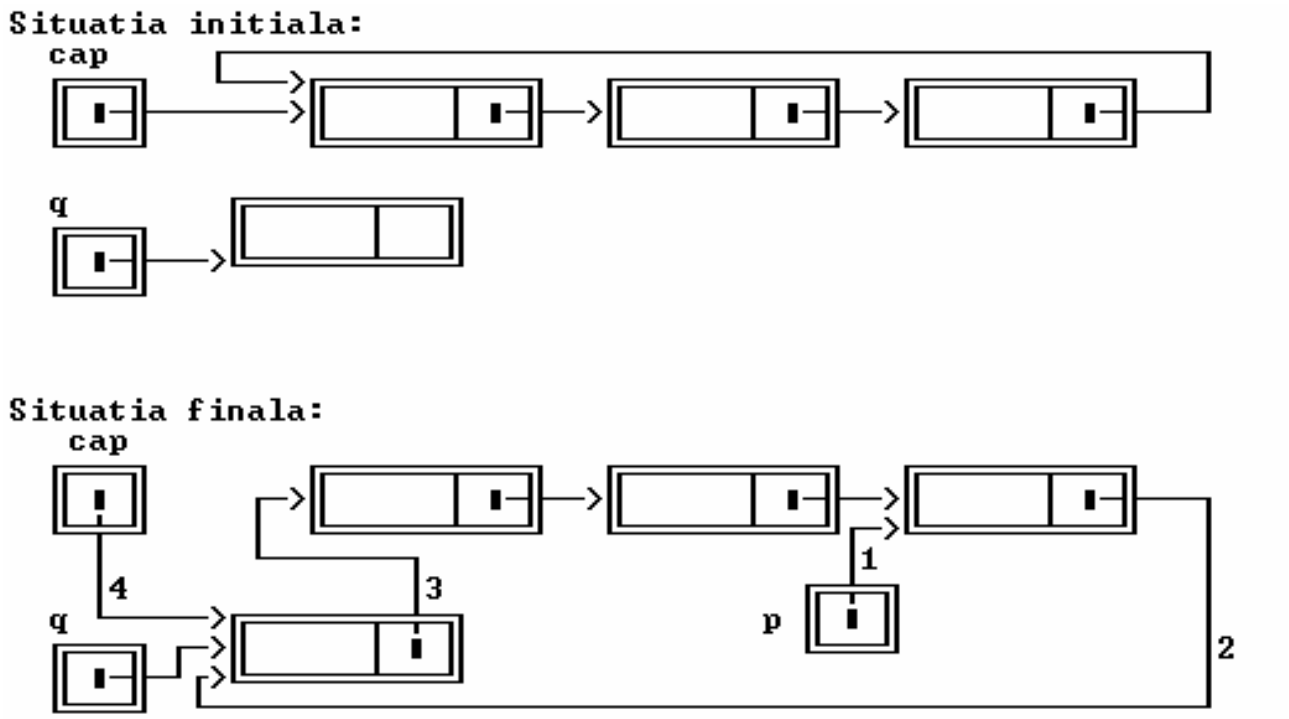
Bucla while se poate opri pe condifia "***p->leg==cap***", ceea ce insemna ca nici un element din lista nu indeplineste conditia iar pointerul ***p*** indica ultimul element din lista, sau pe conditia "***conditie(p->leg)***", ceea ce insemna ca pointerul ***p*** va contine adresa elementului din fata primului element care indeplineste conditia.

2.3.2. Inserarea unui element in lista

Consideram: **cap** - contine adresa primului element din lista;
q - contine adresa unui element izolat care dorim sa fie inserat in lista.

Laboratorul de Structuri de Date si Algoritmi – Lucrarea nr. 4

[A] Inserarea in fata



Fiecare sageata nou creata insemna o atribuire: se atribuie varibilei in care sageata nou creata isi are originea, valoarea luata dintr-o variabila in care se afla originea unei sageti cu aceeasi destinatie.

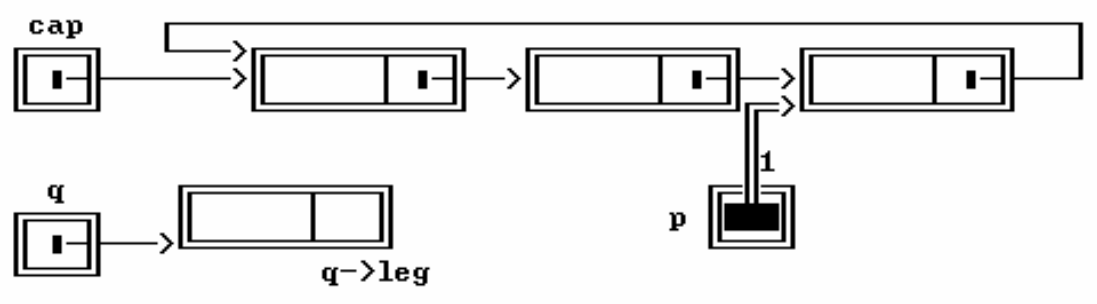
In cazul nostru avem atribuirile (fiecare atribuire corespunde sagetii cu acelasi numar din figura):

(1) parcurge lista (**p** = adresa elemenului ce contine in cimpul **leg**
adresa continuta de pointerul **cap**)
(2) p->leg = q;
(3) q->leg = cap;
(4) cap=q

Sa detaliem:

Prima operatie:

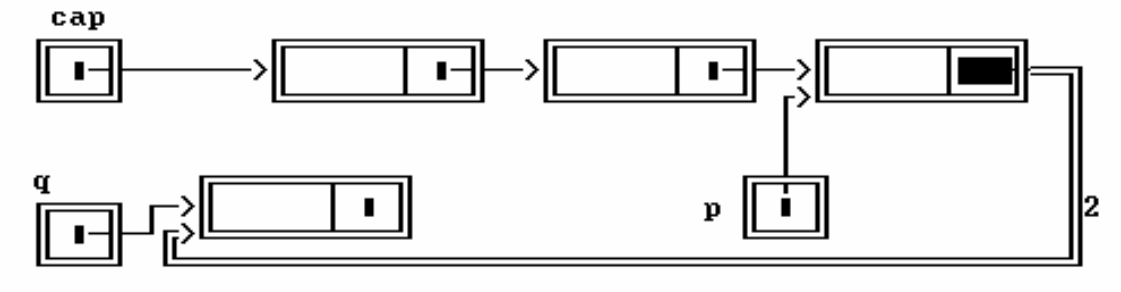
Parcugerea listei are rolul de a depista elementul care pointeaza capul listei (contine in cimpul leg adresa continuta de pointerul ***cap***). Adresa acestui element va fi continuta de pointerul ***p***.



A doua operatie:

```
p->leg=q
```

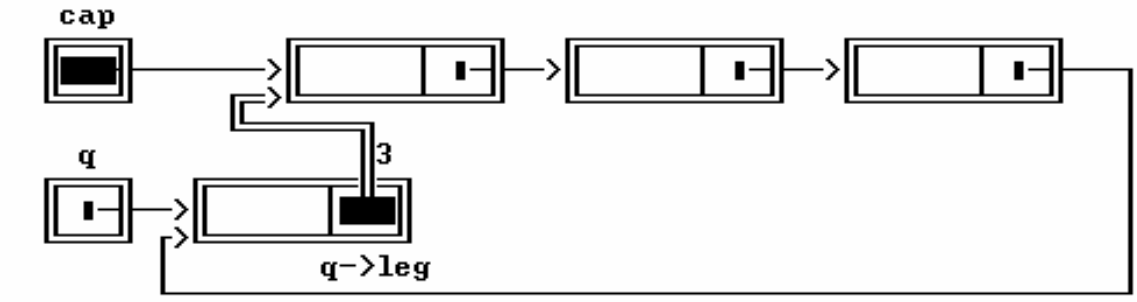
Modifica elementul care contine in cimpul ***leg*** adresa din ***cap*** a primului element din lista pentru a contine adresa continuta de ***q*** a elementului inserat element care va fi de altfel si capul listei



A treia operatie:

```
q->leg = cap;
```

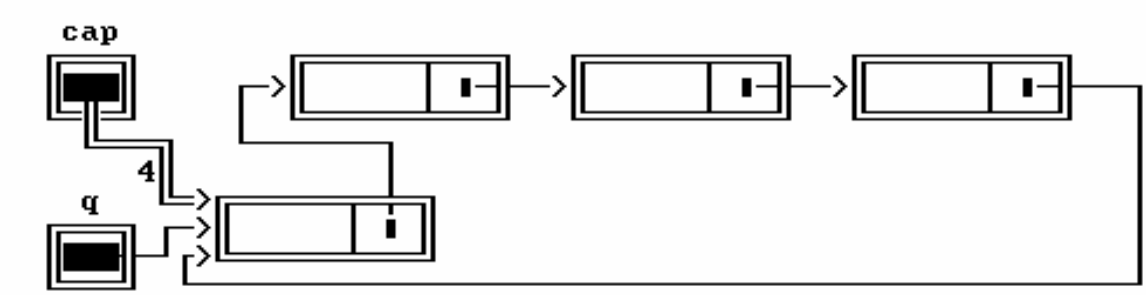
leaga elementul de inserat de restul listei. In urma acestei atribuiri, ***cap*** si ***q->leg*** vor indica ambii inceputul listei initiale (vezi figura de mai jos).



A patra operatie:

```
cap = q;
```

pune in pointerul ***cap*** adresa elementului inserat in fata listei.



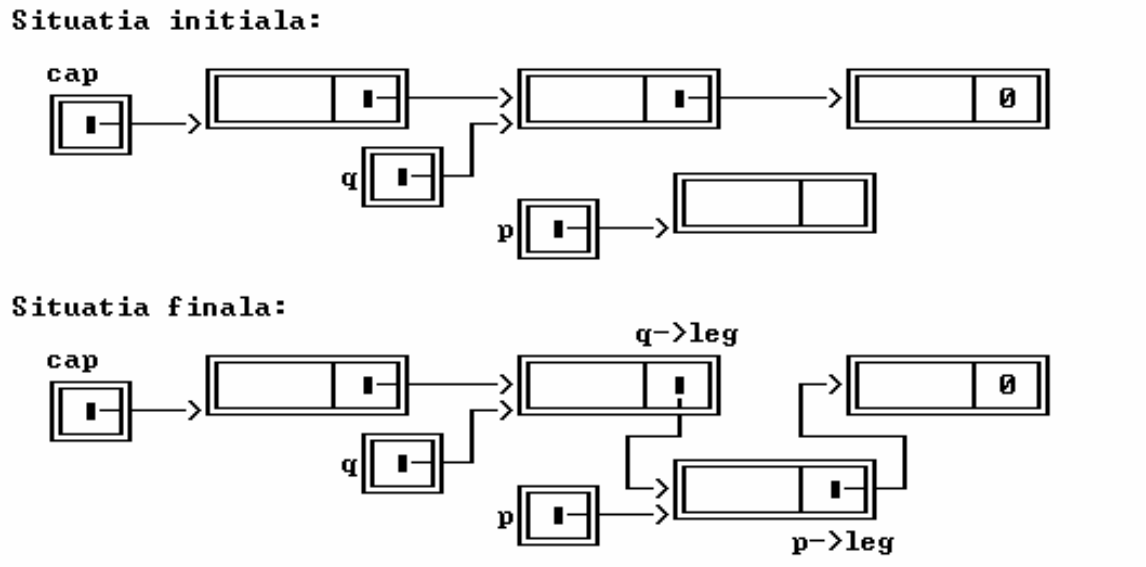
Observatie:

Daca lista in care se face insertia este vida atunci trebuie efectuate citeva modificari in secventa 1- 4 pentru ca rezultatul sa fie corect.

Exercitiu*: Care sunt aceste modificari?*

[B] Inserarea in interior

Varibila ***q*** va indica elementul dupa care se face inserarea.




```
(1)     p->leg = q->leg;
(2)     q->leg = p;
```

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 4

Observatii:

1. Nu se poate face inserarea in fata unui element dat (prin ***q***) fara a parcurge lista de la capat.

2. Nu exista nici o diferenta intre inserarea in interior in cazul listelor liniare simplu inlantuite si cel al listelor circulare.

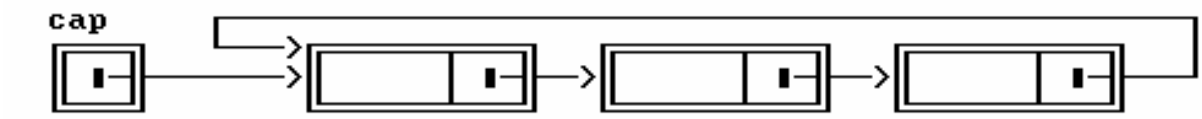
2.3.2. Stergerea unui element din lista

Consideram: ***cap*** - contine adresa primului element din lista.

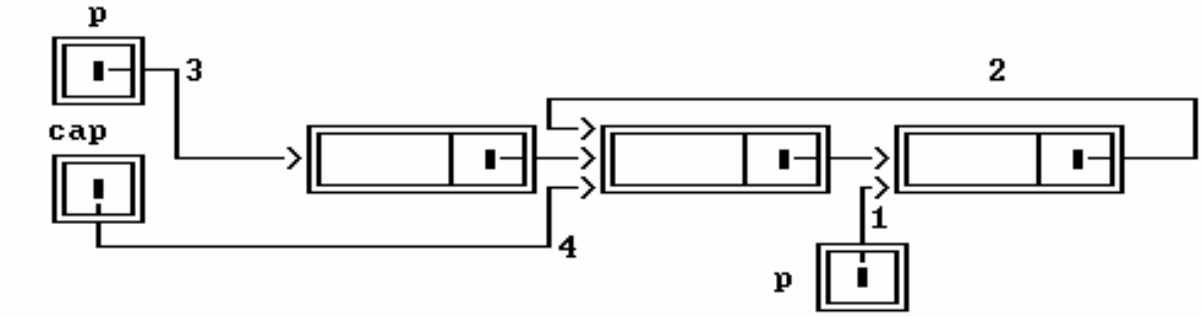
[A] Stergerea la inceputul listei

Prin operatia de stergere se intelege scoaterea unui element din inlantuire. Elementul care a fost izolat de lista trebuie sa fie procesat in continuare, cel putin pentru a fi eliberata zona de memorie pe care o ocupa, de aceea adresa lui trebuie salvata (sa zicem in variabila pointer ***p***).

Situatia initiala:



Situatia finala:

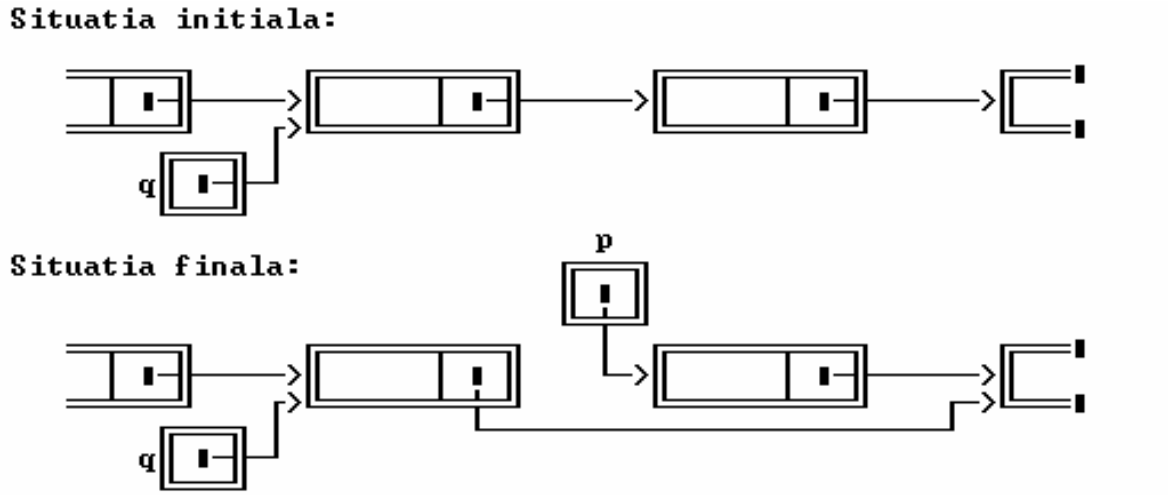



```
(1)     parcurge lista (pointerul p va contine adresa
elementului care pointeaza prin leg la elementul adresat de
cap)
(2)     p->leg=cap->leg
(3)     p = cap;
(4)     cap = cap->leg;
(5)     delete p;          // Elibereaza zona de memorie
```

Laboratorul de Structuri de Date si Algoritmi – Lucrarea nr. 4

[B] Stergerea in interior

Varibila ***q*** va indica elementul din fata celui care va fi sters.




```
(1)     p = q->leg;
(2)     q->leg = p->leg;  // sau q->leg = q->leg->leg;
        delete p;
```

Observatii:

1. Atunci cind ***q*** indica penultimul element dintr-o lista, atribuirile de mai sus functioneaza corect si sterg ultimul element din lista.

2. Nu se poate face stergerea in fata unui element dat (prin ***q***) fara a parcurge lista de la capat.

3. Nu exista nici o diferenta intre stergera in interior in cazul listelor liniare simplu inlantuite si cel al listelor circulare.

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 4

TEMA

1. Se citeste de la intrare un sir de numere intregi.

a) Sa se plaseze numerele citite intr-o lista dublu inlantuita, prin inserari repetate in fata listei.
b) Sa se afiseze lista creata in ordinea inversa citirii numerelor.
c) Sa se insereze un numar citit de la intrare in fata unei pozitii citite de la intrare.
d) Se se stearga un element aflat in fata unei pozitii citita de la intrare.

2. Sa se scrie un program C (C++) care inverseaza legaturile intr-o lista circulara simplu inlantuita.

3. Fie X=(x[1],x[2],...,x[n]) si Y=(y[1],y[2],...,y[m]) doua liste circulare simplu inlantuite.
Scrieti un program C (C++) care sa uneasca cele doua liste in una singura:
Z=(x[1],x[2],..,x[n],y[1],y[2],...,y[m])
Scrieti un program C (C++) care sa interclasese cele doua liste asfel:
Z=(x[1],y[1],x[2],y[2],...,x[m],y[m],x[m+1],...,x[n]) daca m<=n sau
Z=(x[1],y[1],x[2],y[2],...,x[n],y[n],y[n+1],...,y[m]) daca n<=m

4. (*Problema lui Joseph*) Se considera n persoane aranjate in cerc. Incepand cu o pozitie particulara p_0 se efectueaza numararea persoanelor si, in mod brutal se executa fiecare cea de-a m-a persoana; cand este executata o persoana, cercul se inchide dupa eliminarea aceateia. Executia se termina cand ramane o singura persoana; aceasta va ramane in viata.

- Sa scrie o procedura (un modul) care afiseaza persoanele in ordinea in care au fost executate ;
- Presupunand ca se doreste salvarea unei anume persoane, sa se construiasca un modul care sa determine pozitia p_0 astefel incat, daca numaratoarea incepe cu i_0, atunci persoana salvata sa fie cea dorita.
- Sa se scrie un program C (C++) care sa testeze cele doua module.