

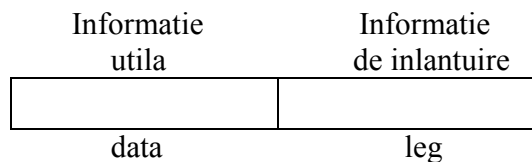
Liste liniare simplu inlantuite

1. Introducere
2. Liste liniare simplu inlantuite alocate static
3. Liste liniare simplu inlantuite alocate dinamic
4. Operatii in liste liniare simplu inlantuite

1. Introducere

O **lista** este o colectie de elemente intre care este specificata cel putin o relatie de ordine. O **lista liniara simplu inlantuita** este caracterizata prin faptul ca relatia de ordine definita pe multimea elementelor este unica si totala. Ordinea elementelor pentru o astfel de lista este specificata explicit printr-un cimp de informatie care este parte componenta a fiecarui element si indica elementul urmator, conform cu relatia de ordine definita pe multimea elementelor listei.

Deci fiecare element de lista simplu inlantuita are urmatoarea structura:



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

Daca exista un ultim element in lista atunci lista se numeste **liniara**. Daca nu exista un element care sa contina in cimpul informatie valoarea *null*

2. Lista liniara simplu inlantuita alocata static

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

Consideram urmatoarele declaratii:

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 1

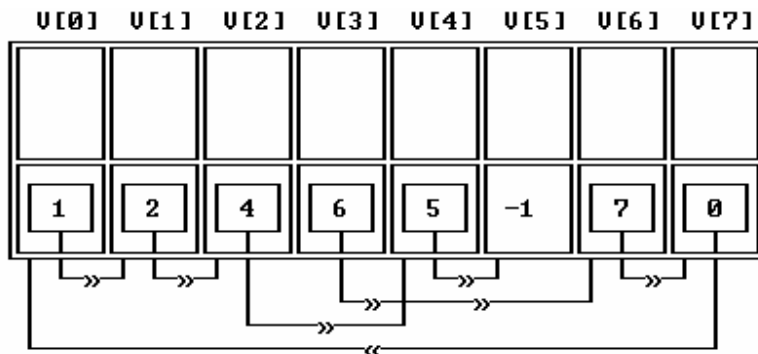
```
struct Element {
    char* data;
    int leg;
};
Element V[8];
```

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 urmatoar:

	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 ordinea:

$V[3]$, $V[6]$, $V[7]$, $V[0]$, $V[1]$, $V[2]$, $V[4]$, $V[5]$.

Obs. Ultimul element din lista are in cimpul **leg** valoarea -1. Este necesar sa cunoastem care este primul element din inlantuire, pentru aceasta retinem intr-o variabila:

```
int cap;
```

indexul primului element.

```
cap=3.
```

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 1

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

```
int crt;
.....
crt = cap;
while (crt!=-1) {
    Prelucreaza V[crt]
    crt = V[crt].leg;
}
```

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 ca valoarea memorata aici indica elementul la care duce sageata.

3. Lista liniara simplu inlantuita alocata 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.

Pentru rolul pe care il joaca informatiile de legatura in structurile inlantuite, cel mai potrivit este tipul *pointer*. Tipul cimpului **leg** va fi "***pointer la element de lista***".

Iata cum arata declaratiile tipului "element de lista liniara simplu inlantuita" 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;
```

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 1

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

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

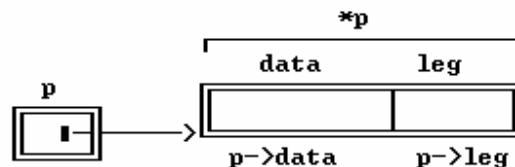
in urma unei operatii:

```
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->leg**.

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->leg = 0**).

Pentru a manevara 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.

4. Operatii in liste liniare simplu inlantuite

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;
```

Operatiile primitive pentru acces la o lista inlantuita sint:

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 1

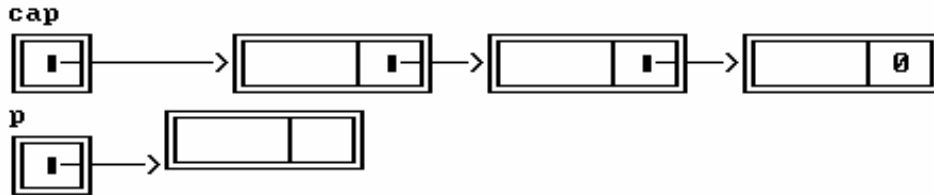
4.1. Inserarea unui element in lista

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

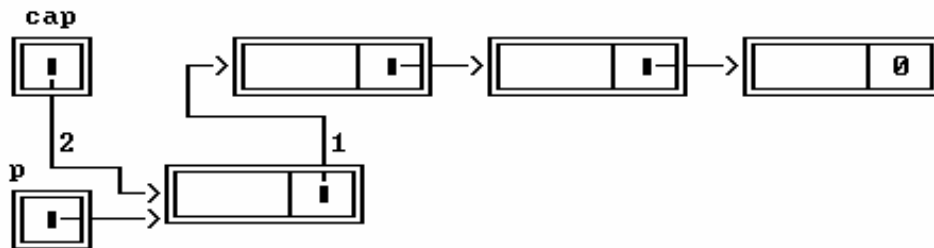
p - contine adresa unui element izolat care dorim sa fie inserat in lista.

4.1.1. Inserarea in fata

Situatia initiala:



Situatia finala:



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

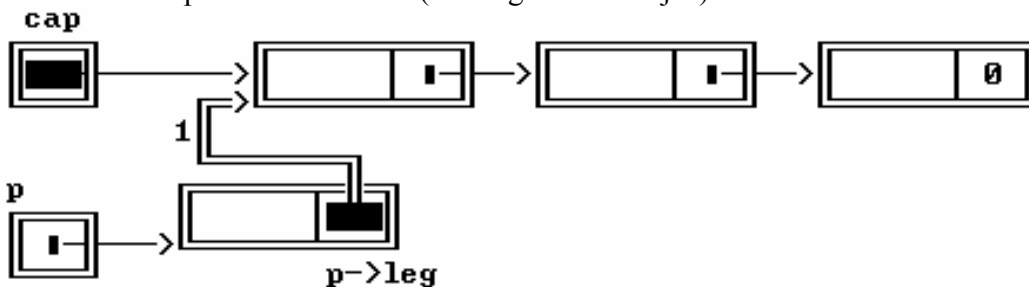
In cazul nostru avem atribuirile (fiecare atribuire corespunde sagetii cu acelasi numar din figura):

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

Sa detaliem: Prima atribuire

`p->leg = cap;`

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

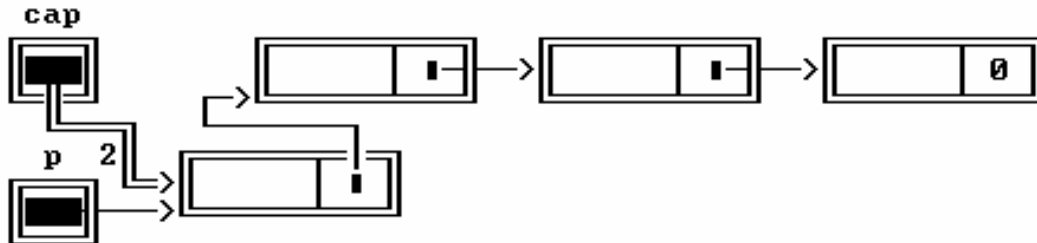


Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 1

A doua atribuire:

```
cap = p;
```

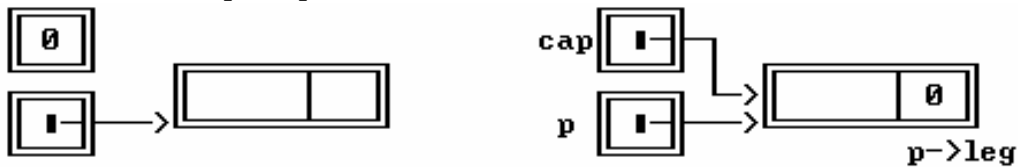
pune in pointerul cap adresa elementului inserat in fata listei.



Observatie:

Daca pointerul **cap** este initial nul, (ceea ce inseamna inserarea intr-o lista vida) atribuirile de mai sus functioneaza corect rezultind o lista cu un singur element.

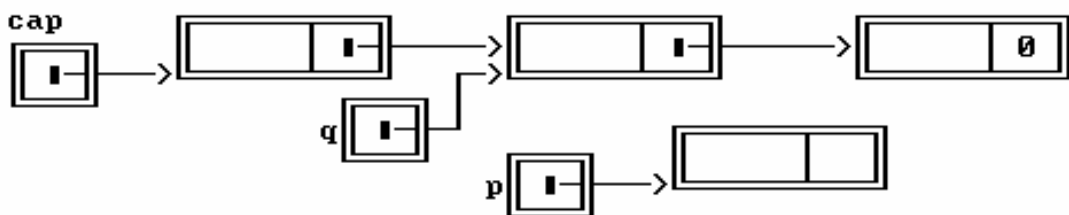
```
p->leg = cap;    // de fapt    p->leg = 0;
cap = p;
```



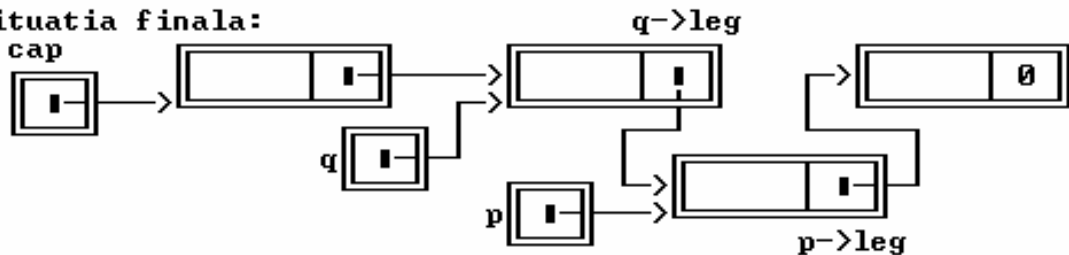
4.1.2. Inserarea in interior sau la sfirsit

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

Situatia initiala:



Situatia finala:



Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 1

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

Observatii:

Atunci cind **q** indica ultimul element dintr-o lista, atribuirile de mai sus functioneaza corect si adauga elementul indicat de **p** la sfirsitul listei.

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

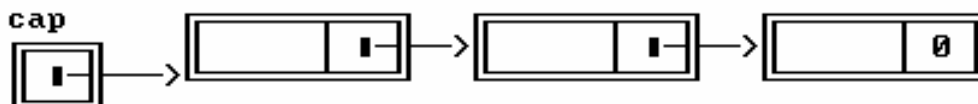
4.2. Stergerea unui element din lista

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

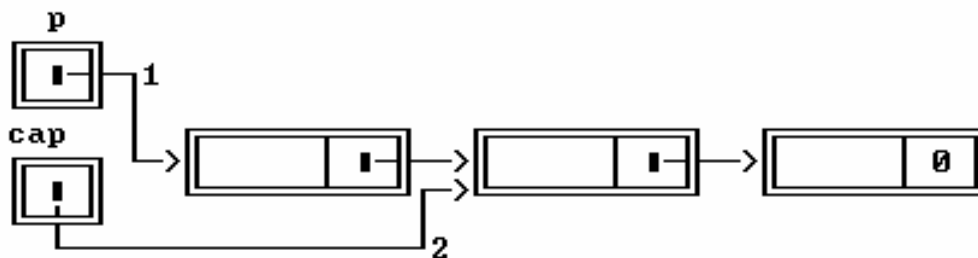
4.2.1. Stergerea la inceputul listei

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:



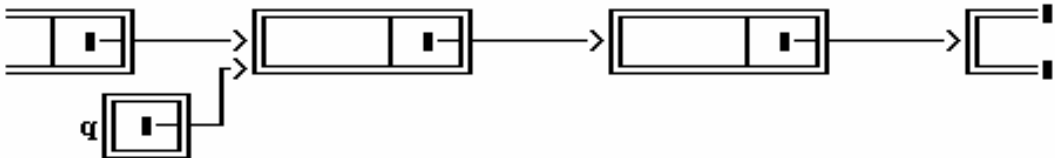
```
(1)      p = cap;  
(2)      cap = cap->leg;  
          delete p;           // Elibereaza zona de memorie
```

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 1

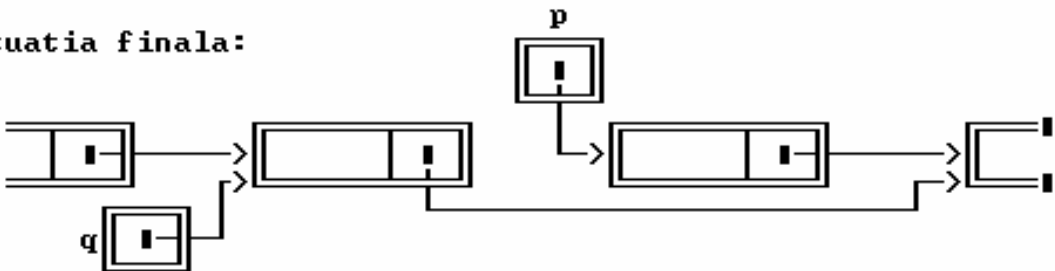
4.2.2. Stergerea in interior sau la sfirsit

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

Situatia initiala:



Situatia finala:



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

Observatii:

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

Nu se poate face stergerea elementului indicat de **q** fara parcurgerea listei de la capat.

4.3. 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:

```
p = cap;  
while (p!=0){
```

```
    Prelucraza p->data
```

```
    p = p->leg;  
}
```

```
for(p=cap; p!=0; p=p->leg){
```

```
    Prelucraza p->data
```

```
}
```


Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 1

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!=0 && !conditie(p->leg))
    p = p->leg;
```

Buclo **while** se poate opri pe conditia "**p->leg==0**", 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.

TEMA

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

- Sa se plaseze numerele citite intr-o lista inlantuita, prin inserari repetate in fata listei.
- Sa se afiseze lista creata.
- Se citeste un numar si sa se determine daca acesta se afla printre elementele listei create.
- Sa se insereze un numar citit de la intrare intr-o pozitie citita de la intrare.
- Se se stearga un element din lista dintr-o pozitie citita de la intrare.
- Se se afiseze elementul situat pe pozitia k numarata de la sfirsitul la inceputul listei, fara a parcurge lista mai mult de o data.

2. Sa se construiasca modul (fisierle .H si .C (.CPP)) care sa contina tipurile de date si operatiile care implementeaza sub forma unei liste simplu inlantuite o agenda de numere de telefon. Elementele listei vor contine ca informatie utila doua cimpuri:

- nume - numele persoanei;
- tel - numarul de telefon;

Elementele listei vor fi pastrate in ordine alfabetica dupa numele persoanei.

Sa se definesca procedurile care:

- insereaza un element in lista;
- sterge din lista o persoana data;
- cauta in lista numarul de telefon al unei persoane date;
- afiseaza lista in intregime.

Sa se construiasca un program C (C++) care cu ajutorului unui meniu simplu sa permita selectarea operatiilor definite mai sus.

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 1

3. Fie $X=(x[1],x[2],...,x[n])$ si $Y=(y[1],y[2],...,y[m])$ doua liste liniasre simplu inlantuite. Scrieti un program C (C++) care:

- sa uneasca cele doua liste in una singura:

$Z=(x[1],x[2],...,x[n],y[1],y[2],...,y[m])$

- sa interclasese cele doua liste astfel:

$Z=(x[1],y[1],x[2],y[2],...,x[m],y[m],x[m+1],...,x[n])$ daca $m \leq n$ sau

$Z=(x[1],y[1],x[2],y[2],...,x[n],y[n],y[n+1],...,y[m])$ daca $n \leq m$

4. Sa se construiasca un modul (fisierle .H si .C (.CPP)) care sa contina tipurile de date si functiile care implementeaza urmatoarele operatii:

- parcurge o lista simplu inlantuita in ambele sensuri (dus-intors) utilizind $O(1)$ memorie suplimentara.

- testeaza daca o lista simplu inlantuita are bucle.

- determina mijlocul unei liste simplu inlantuite.

Sa se construiasca un program C (C++) care cu ajutorului unui meniu simplu sa permita selectarea operatiilor definite mai sus.