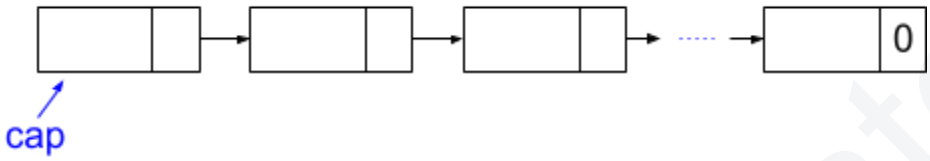
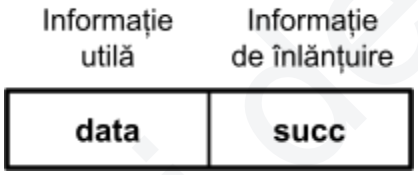
Liste liniare simplu înlănțuite

1. Introducere

O **listă** este o colecție de elemente între care este **specificată cel puțin o relație de ordine**.



O **listă liniară simplu înlănțuită** este caracterizată prin faptul că relația de ordine definită pe mulțimea elementelor este unică și totală. Ordinea elementelor pentru o astfel de listă este specificată explicit printr-un câmp de informație care este parte componentă a fiecărui element și indică elementul următor, conform cu relația de ordine definită pe mulțimea elementelor listei.
Prin urmare, fiecare element de listă simplu înlănțuită are următoarea structură:



Pe baza informației de înlănțuire (păstrată în câmpul **succ**) trebuie să poată fi identificat următorul element din listă.
Dacă **există un ultim element în listă** atunci lista se numește **liniară**. Dacă **nu există un element** care să conțină în câmpul de înlănțuire valoarea **NULL** (sau **0**), atunci lista este **circulară**.

2. Lista liniară simplu înlănțuită alocată static

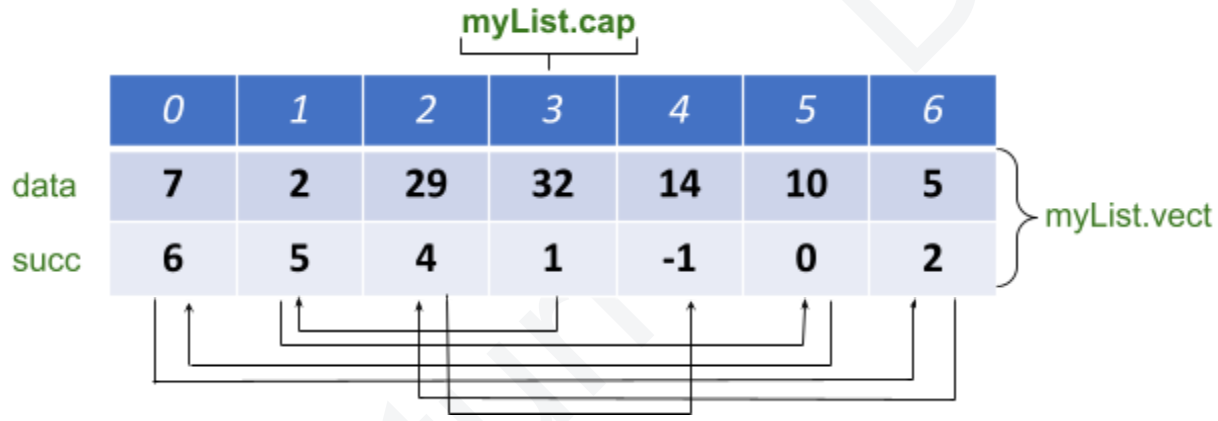
Dacă **implementarea structurii de listă înlănțuită** se face prin **tablouri**, aceasta este o **listă înlănțuită alocată static** sau simplu o **listă înlănțuită statică**.
Considerăm următoarele declarații:

```
C/C++
struct Element{
      char* data;
      int succ;
};

struct Lista{
       Element vect[7];
```

```
        int cap;
}
...
Lista myList;
myList.cap = 3;
myList.vect[myList.cap].data = 32;
myList.vect[myList.cap].succ = 1;
...
```

Pentru elementele vectorului `myList.vect` există o ordine naturală dată de aranjarea în memorie a elementelor sale: `myList.vect[0]`, `myList.vect[1]`, …, `myList.vect[6]`. Vom reprezenta memoria ocupată de vectorul `myList.vect` astfel încât fiecare element să fie reprezentat vertical, în felul următor:



Completând câmpul **succ** pentru fiecare element al vectorului putem obține o listă liniară simplu înlănțuită. Valoarea câmpului **succ** va fi indexul în vector al următorului element din listă.

Pe baza înlănțuirii stabilită de valorile din figura de mai sus se obține ordinea: `myList.vect[3]`, `myList.vect[1]`, `myList.vect[5]`, `myList.vect[0]`, `myList.vect[6]`, `myList.vect[2]`, `myList.vect[4]`.

Observații

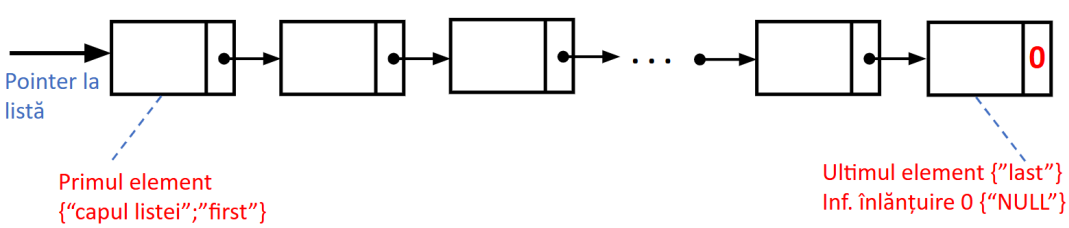
- Ultimul element din listă are în câmpul **succ** valoarea **-1**.
- Este necesar să cunoaștem care este primul element din înlănțuire, pentru aceasta reținem într-o variabilă `myList.cap` indexul primului element.

Parcurgerea în ordine a elementelor listei se face în felul următor:

```
C/C++
int crt;
```

```
...
crt = myList.cap;
while(crt != -1){
     prelucreaza(myList.vect[crt]);
     crt = myList.vect[crt].succ;
}
```

3. Lista liniară simplu înlănțuită alocată dinamic



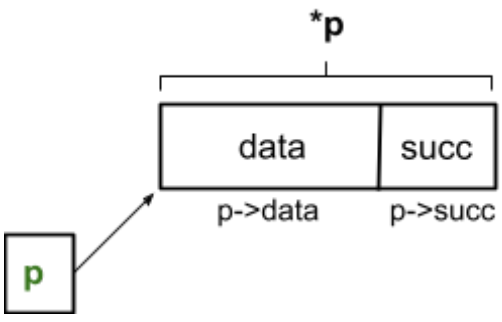
Dacă **implementarea structurii de listă înlănțuită** se face prin tehnici de **alocare dinamică** se obține o **listă înlănțuită alocată dinamic** sau simplu o **listă înlănțuită dinamică**.

Pentru rolul pe care îl joacă informațiile de legătură în structurile înlănțuite, cel mai potrivit este tipul *pointer*. Tipul câmpului **succ** va fi **"pointer la element de listă"**.

Iată cum arată declarațiile tipului "element de listă liniară simplu înlănțuită" în C++:

```
C/C++
typedef int Atom; //orice tip predefinit
struct Element{
      Atom data;
      Element *succ;
};
```

Având declarația de mai sus și `Element *p;` în urma unei operații de tipul `p=new Element;`, `p` a fost inițializat cu adresa unei variabile de tip `Element` alocată în zona de alocare dinamică (heap).



Câmpurile elementului **p** se vor accesa prin expresiile **p->data** și **p->succ**. Constanta **NULL** (sau **0**) pentru un pointer înseamnă o adresă imposibilă. Această valoare va fi utilizată pentru a termina înlănțuirea (ultimul element din listă va avea **p->succ = NULL**).

Pentru a manevra o listă este nevoie doar de un pointer la primul element al listei. Pentru a indica o **listă vidă**, acest pointer va primi **valoarea NULL**.Parcurgerea se face într-un singur sens, pornind de la primul element al listei.

4. Operații în liste liniare simplu înlănțuite

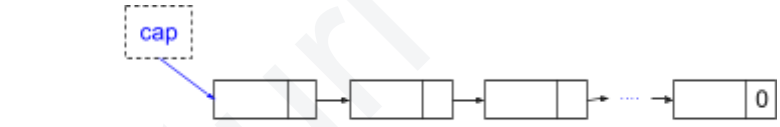
Fără a restrânge generalitatea, vom detalia doar implementarea prin pointeri.
Considerăm declarațiile de tipuri de mai sus și variabilele:

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

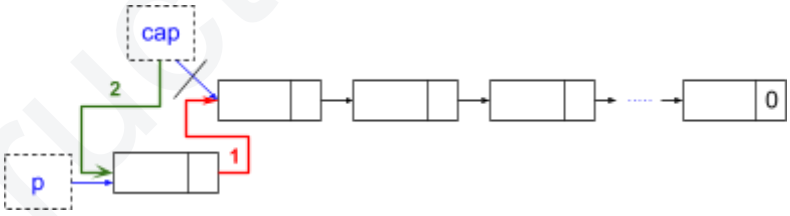
4.1 Inserarea unui element în listă

4.1.1. Inserarea în fața primului element

Situația inițială:



Situația finală:



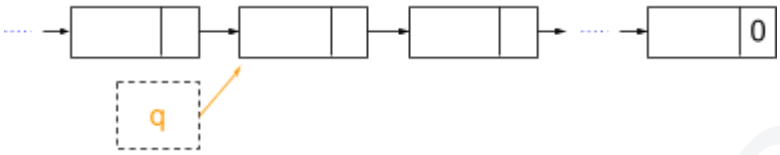

```
p <- get_sp(); //alocare spațiu pentru noul element
data(p) <- info; //stocarea valorii/informației în câmpul data
succ(p) <- cap; //crearea legăturii de la element la listă (1)
cap <- p; //actualizarea adresei primului element din listă cu adresa
          //elementului nou inserat (2)
```

Observație
Dacă pointerul **cap** este inițial **NULL**, (ceea ce înseamnă inserarea într-o listă vidă) atribuirile de mai sus funcționează corect rezultând o listă cu un singur element.
`succ(p) <- cap; //de fapt succ(p) <- 0;`

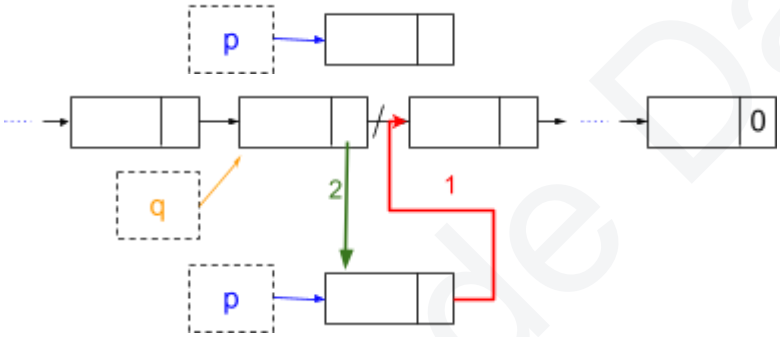
4.1.2. Inserarea unui element în interiorul listei

Se parcurge lista până la elementul de pointer *q*, după care se face inserarea!

Situația inițială:



Situația finală:




```
//parcurgere până la elementul de pointer q după care se face
inserarea
…
//inserarea elementului de pointer p după cel de pointer q
p <- get_sp(); //alocare spațiu pentru noul element
data(p) <- info; //stocarea valorii/informației în câmpul data
succ(p) <- succ(q); //crearea legăturii de la element la listă (1)
succ(q) <- p; //crearea legăturii de la listă la element (2)
```

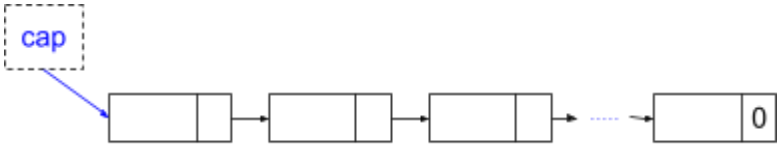
Observații

- Atunci când **q** indică ultimul element dintr-o listă, atribuirile de mai sus funcționează corect și adaugă elementul indicat de **p** la sfârșitul listei.
- Nu se poate face inserarea în fața unui element dat (prin **q**) fără a parcurge lista de la capăt.

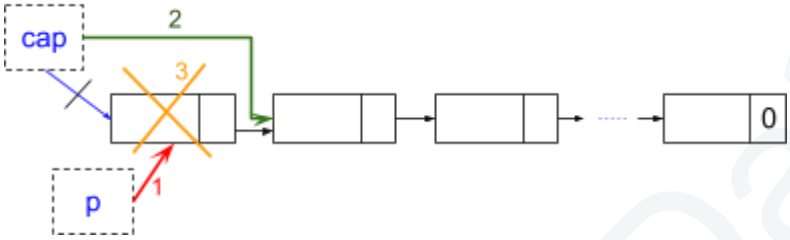
4.2 Ștergerea unui element din listă

4.2.1 Ștergerea primului element

Situația inițială:



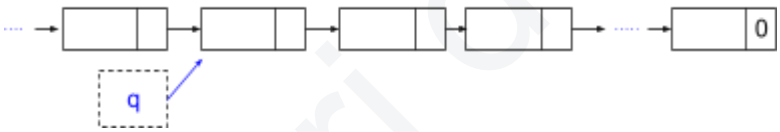
Situația finală:



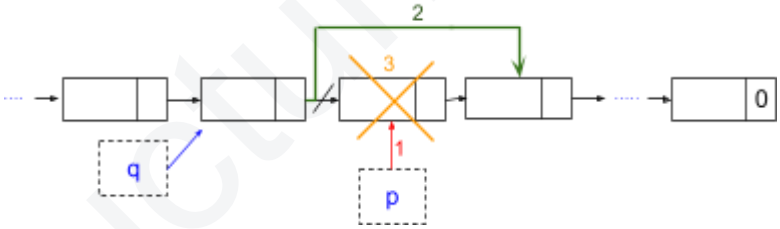

```
p <- cap;//în p se va salva adresa elementul pe care îl vom șterge (1)
cap <- succ(cap); //se actualizează adresa primului element (2)
free_sp(p); //eliberarea zonei de memorie (3)
```

4.2.1 Ștergerea unui element din interiorul listei

Situația inițială:



Situația finală:




```
//parcurgere până la elementul de pointer q, este necesar să ne oprim
cu un element în fața celui pe care dorim să îl ștergem
…
//ștergere elementului de pointer p
p <- succ(q); //salvăm adresa elementului pe care îl vom șterge (1)
succ(q) <- succ(p); //refacem legăturile din listă (2)
free_sp(p); //eliberăm zona de memorie (3)
```

Observații

- Atunci când **q** indică penultimul element dintr-o listă, atribuirile de mai sus funcționează corect și șterg ultimul element din listă.
- Nu se poate face ștergerea elementului indicat de **q** fără parcurgerea listei de la capăt.

4.2 Parcugerea listei

O parcurgere înseamnă prelucrarea pe rând a tuturor elementelor listei, în ordinea în care acestea apar în listă. Vom avea o variabilă pointer **p** care va indica pe rând fiecare element al listei:

```
C/C++
p = cap;
while(p!=0){
   prelucreaza(p->data);
   p = p->succ;
}

sau

for(p=cap; p!=0; p=p->succ)
    prelucreaza(p->data);
```

Un caz special apare atunci când dorim să facem o parcurgere care să se oprească în fața unui element care să îndeplinească o condiție (ca atunci când inserăm un element într-o poziție dată printr-o condiție, sau ștergem un element care îndeplinește o condiție).

Presupunem că lista are cel puțin un element.

```
C/C++
p = cap;
while(p->succ!=0 && !conditie(p->succ))
      p = p->succ;
```

Bucla **while** se poate opri pe condiția **"p->succ==0"**, ceea ce înseamnă că niciun element din listă nu îndeplinește condiția iar pointerul **p** indică ultimul element din listă, sau pe condiția **"conditie(p->succ)"**, ceea ce înseamnă că pointerul **p** va conține adresa elementului din fața primului element care îndeplinește condiția.

3. Aplicații

1. Se citește de la intrare un șir de numere întregi.
    a. Să se creeze o listă înlănțuită, prin inserări repetate în fața listei (vezi curs 2).
    b. Să se afișeze lista creată.
    c. Se citește un număr **x**, să se determine dacă acesta se află printre elementele listei create.
    d. Să se insereze un număr citit de la tastatură într-o poziție citită de la tastatură.
    e. Să se șteargă un element din listă dintr-o poziție citită de la tastatură.

f. Să se afișeze elementul situat pe poziția **k** numărată de la sfârșitul la începutul listei, fără a număra elementele listei.
g. Să se scrie o funcție pentru a parcurge lista simplu înlănțuită în ambele sensuri (dus-întors) fără a utiliza alte structuri adiționale definite de utilizator. Ce puteți spune despre complexitatea timp/spațiu?
h. Să se scrie o funcție pentru a testa dacă lista simplu înlănțuită are bucle.
i. Să se scrie o funcție care determină mijlocul listei simplu înlănțuite fără a număra elementele acesteia.
j. Să se scrie o funcție care inversează legăturile în lista simplu înlănțuită fără resurse suplimentare de memorie (primul element devine ultimul, al doilea element va fi penultimul, etc.).

2. Să se realizeze un proiect care să implementeze sub forma unei liste simplu înlănțuite o agendă de numere de telefon. Elementele listei vor conține ca informație utilă două câmpuri:
   - **nume** - numele persoanei;
   - **tel** - numărul de telefon;

   Elementele listei vor fi păstrate în ordine alfabetică după numele persoanei.

   2.1 Să se implementeze funcții care:
   a. inserează un element în listă;
   b. șterge din listă o persoană dată;
   c. caută în listă numărul de telefon al unei persoane date;
   d. afișează lista în întregime.

   2.2 Să se implementeze un meniu simplu care să permită selectarea operațiilor definite mai sus.

3. Fie X=(x[1],x[2],...,x[n]) si Y=(y[1],y[2],...,y[m]) două liste liniare simplu înlănțuite. Scrieti un program C (C++) care:
   3.1 Să unească (concateneze) cele două liste în una singură:
   Z=(x[1],x[2],..,x[n],y[1],y[2],...,y[m])
   3.2 Să interclaseze cele două liste astfel:
   Z=(x[1],y[1],x[2],y[2],...,x[m],y[m],x[m+1],...,x[n]) dacă m<=n sau
   Z=(x[1],y[1],x[2],y[2],...,x[n],y[n],y[n+1],...,y[m]) dacă n<=m

Aplicațiile neterminate în timpul orelor de laborator rămân ca teme pentru studiu individual!