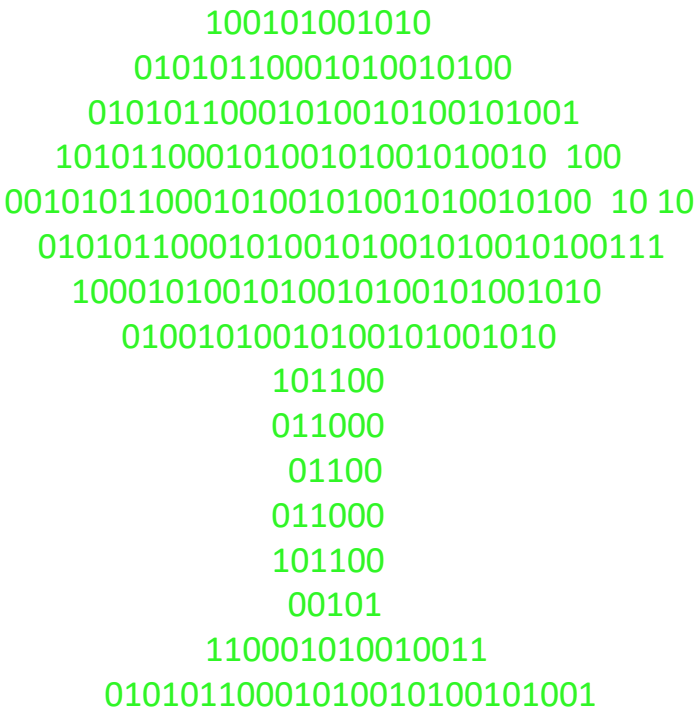
Structuri de date și algoritmi



Curs, IS – An II

Tablouri și liste

“Nothing is more practical than a good theory.”

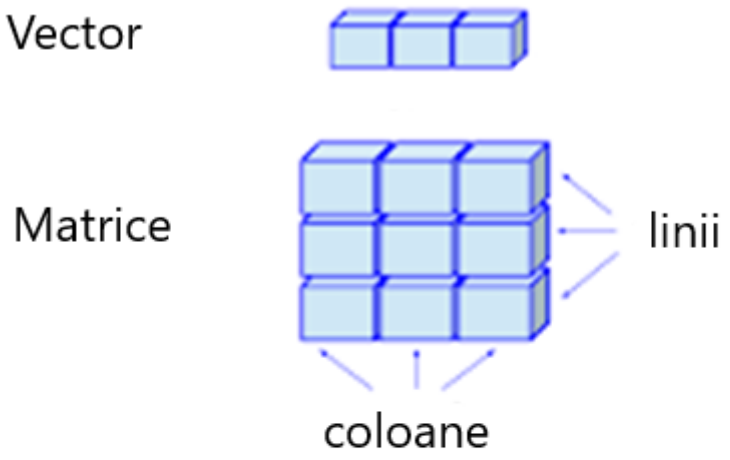
(Ludwig Boltzmann)

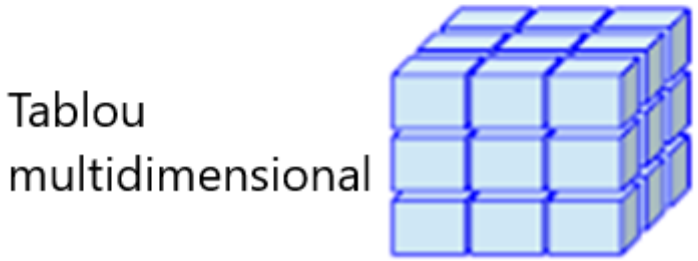
Tablouri

Un tablou este o structura de date abstracta omogenă, statică.

Fiecare element se identifică pe baza unui index (unor indecsi) iar timpul de acces este același pentru fiecare element.

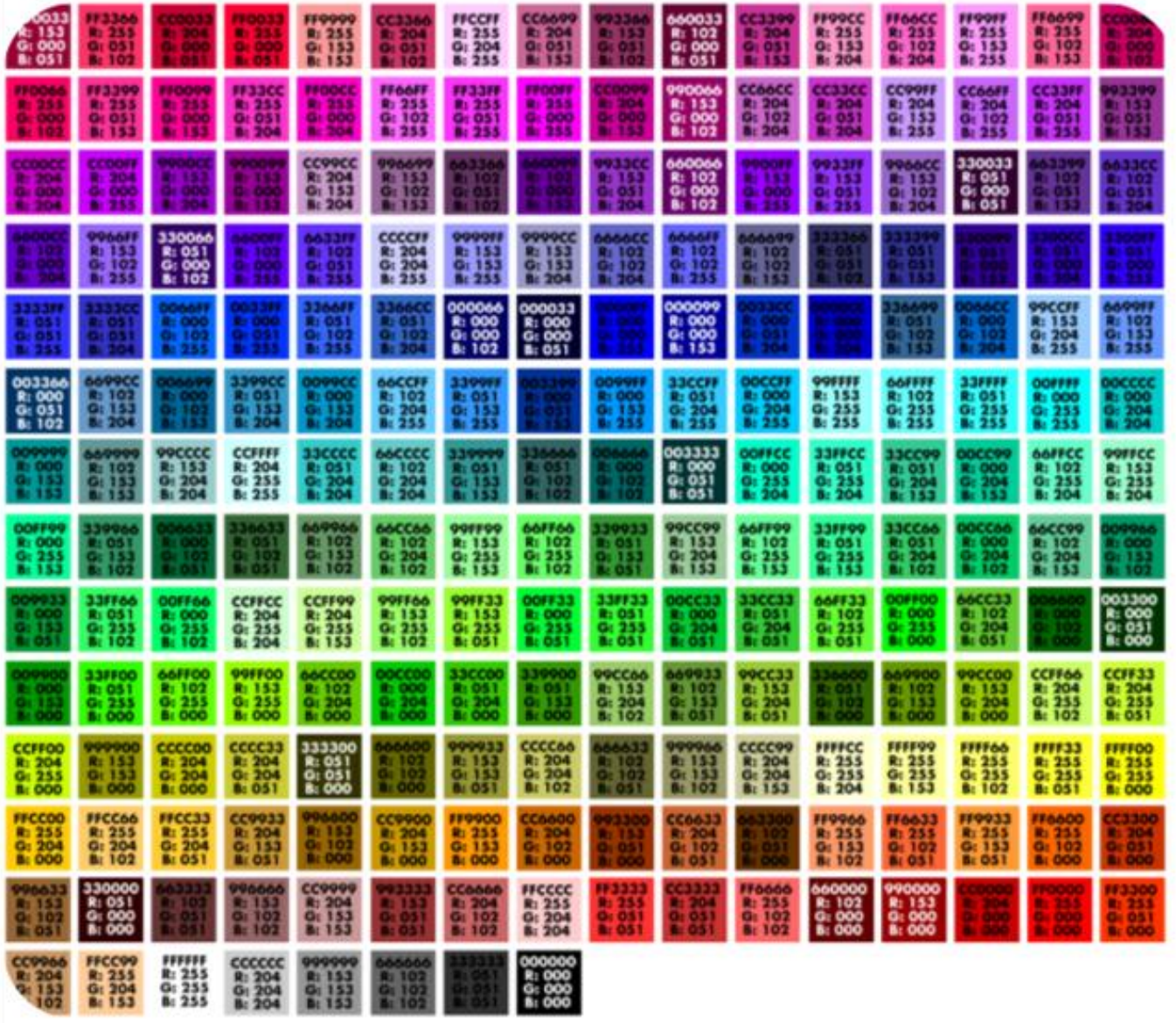
- Indecsii aparțin unei mulțimi peste care este definită o relație de ordine totală;
- Dimensiune fixă specificată la alocare;
- Elementele ocupă locații succesive în memorie;
- Numele tabloului se asociază cu locația de memorie a primului element;
- Adăugarea unui element – operatie laborioasă
    1. Crearea unui nou vector de dimensiune mai mare
    2. Copierea elementelor în noul câmp
- Modificarea pozițiilor elementelor (de ex. într-un vector sortat) operație foarte laborioasă.





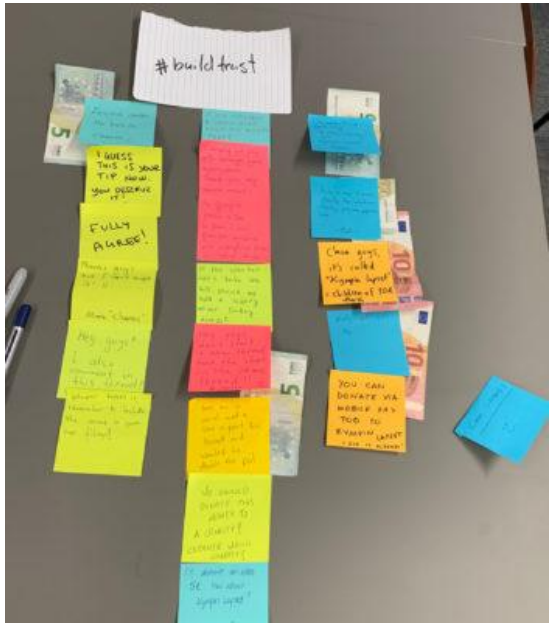
Tablouri ***k*** dimensionale interpretate ca vectori de tablouri ***k-1*** dimensionale

Imagine RGB: Vector de 3 matrice (una pentru fiecare culoare)

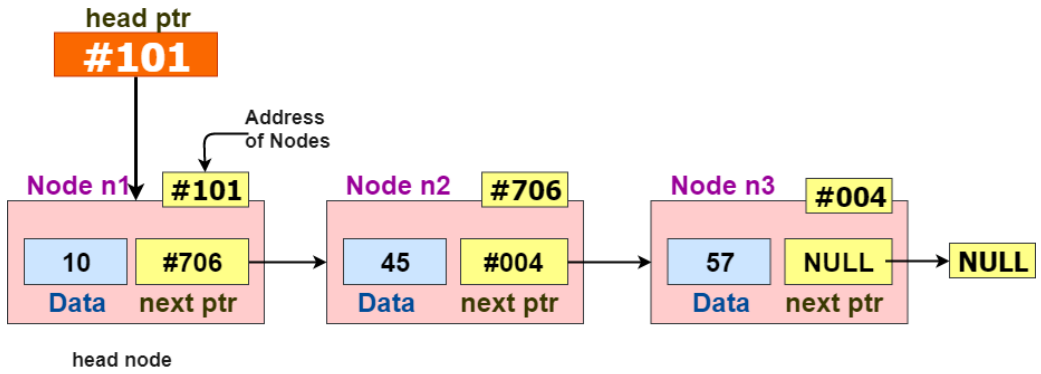


Reprezentare imagine: matrice bidimensională de triplete sau vector de 3 matrice bidimensionale

Tipul abstract de dată (TAD) Listă



Liste de activități



O listă stocată în memorie

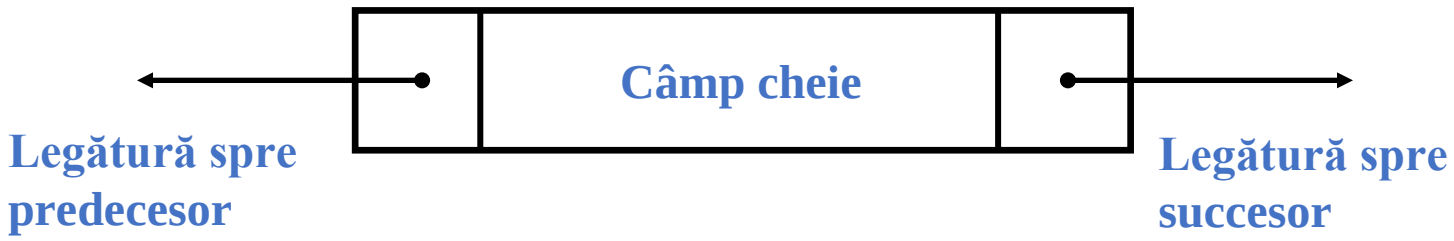
- De ce organizăm astfel informațiile?
- Ce operații vom executa?

TAD Listă

Definiție:

O listă este o structură de date omogenă, secvențială formată din elemente (noduri) între care există cel puțin o relație de ordine.

Un element al unei liste are forma generală:



Un element al listei se identifică prin succesorul sau predecesorul său.

Operații cu liste

1. Operații de caracterizare

- ✓ Determinarea numărului de elemente
- ✓ Localizarea unui element cu anumite proprietăți (cautare)
- ✓ Parcurgerea (afișarea) listei

2. Operații care modifică conținutul listei

- ✓ Initializarea
- ✓ Inserarea unui element în oricare loc din listă
- ✓ Ștergerea unui element din listă
- ✓ Modificarea conținutului unui element
- ✓ Operații complexe (ex. Sortarea listei după câmpul de date)

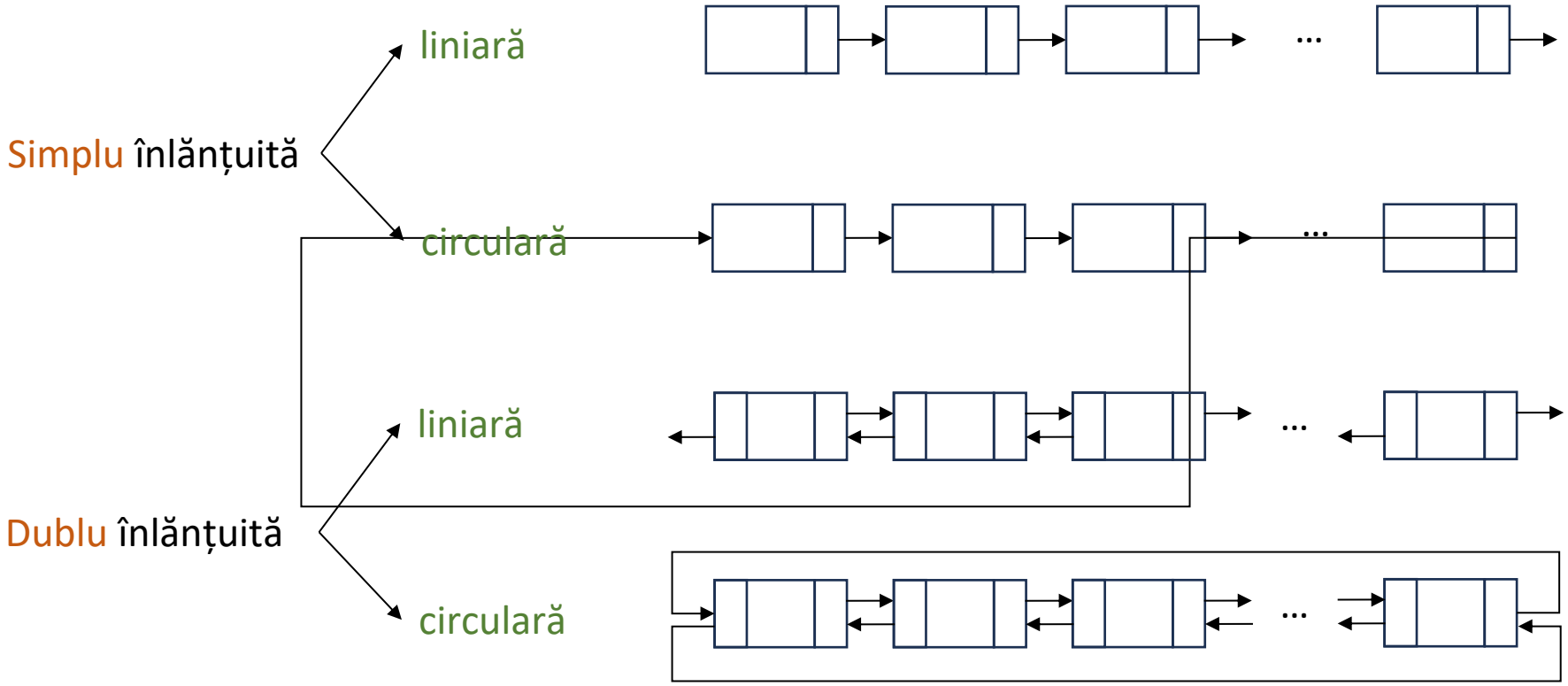
3. Alte operații complexe

- ✓ Selecția elementelor după un criteriu și crearea unei noi liste
- ✓ Separea unei liste în mai multe liste după un anumit criteriu
- ✓ Combinarea unor liste prin concatenare sau interclasare

TAD Listă

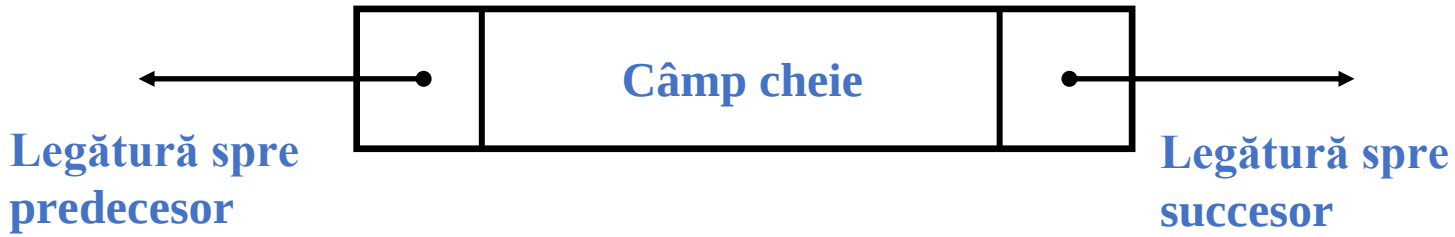
Un element al unei liste are forma generală:

Tipuri de liste:



TAD Listă

Un element al unei liste are forma generală:



Implementare:

- Dinamică
- Statică

TAD Listă

Un element al unei liste are forma generală:

Lista liniară simplu înlanțuită:

Implementare statică

cap

	0	1	2	3	4	5	6
data	**7**	**2**	**29**	**32**	**14**	**10**	**5**
succesor	**6**	**5**	**4**	**1**	**-1**	**0**	**2**

} vec

```
struct Element{
        Atom data;
        uint succesor;
};
```

```
struct Lista{
        Element vec[7];
        uint cap;
};
```

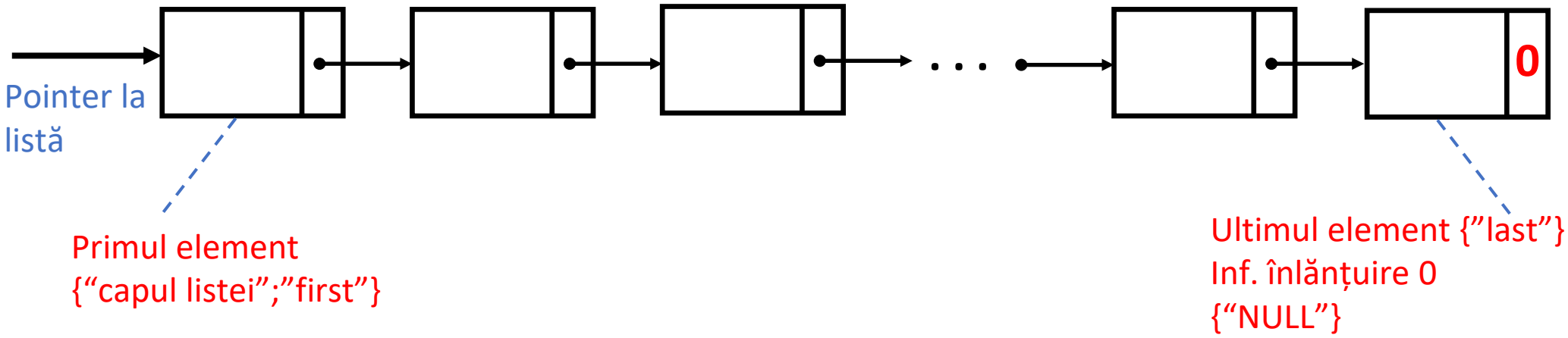
```
Lista myList;
myList.cap = 3;
myList.vec[myList.cap].data = 32;
myList.vec[myList.cap].succesor = 1;
....
```

TAD Listă

Un element al unei liste are forma generală:

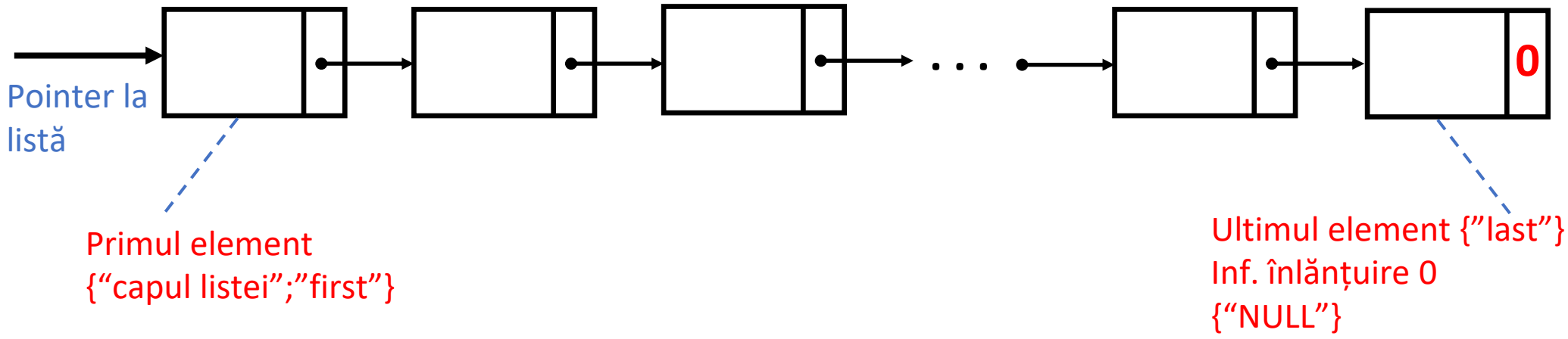
Lista liniară simplu înlanțuită:

Implementare dinamică



Listă lineară simplu înlănțuită – Implementare dinamică

- Un element (nod) din listă este de un tip structură și are (cel puțin) două câmpuri: un câmp de date (sau mai multe) și un câmp de legătură care este pointer la succesor.
- Sunt două elemente particulare:
  - ✓ primul element (“capul listei”) - singurul prin care se face accesul la listă;
  - ✓ ultimul element – al cărui câmp de legatură este NULL.
- Parcurgerea se face într-un singur sens, pornind de la primul element
- Dimensiunea listei (numărul de elemente) este teoretic nelimitată dar practic este limitată de memoria de lucru disponibilă



Lista liniară simplu înlănțuită – Implementare dinamică

Declarații

```
typedef  int Atom;//orice tip predefinit

struct Element
{
   Atom data;
   Element *succ;
};


... main ....

Element *cap;//pointer la lista ("cap")
```

Notații pseudocod

p – adresa unui element din listă {Element *p}

data(p) – informație valoare (Atom) {p→data}

succ(p) – informație înlănțuire (adresa următorului element) {p→succ}

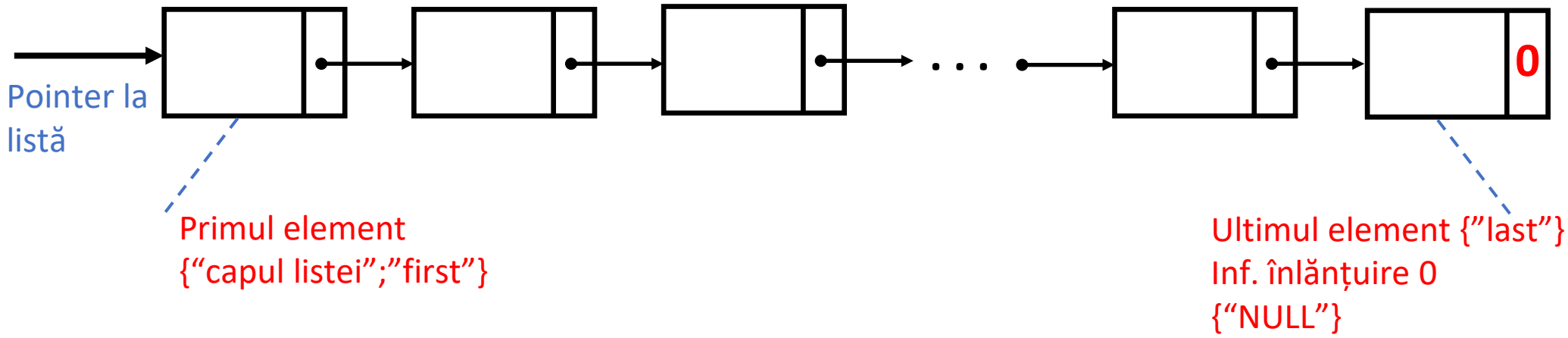
p ← get_sp() – alocarea unei zone de memorie pentru un element {p=new Element}

free_sp(p) – eliberarea zonei de memorie ocupate de elemental de pointer p {delete p}

Operații cu liste liniare simplu înlănțuite

Parcurgerea listei

```
P← cap; //pointer la primul element
While p≠0 Do
    prelucrare (data(p));
    p ← succ(p); // pointer la următorul element
EndWhile
```

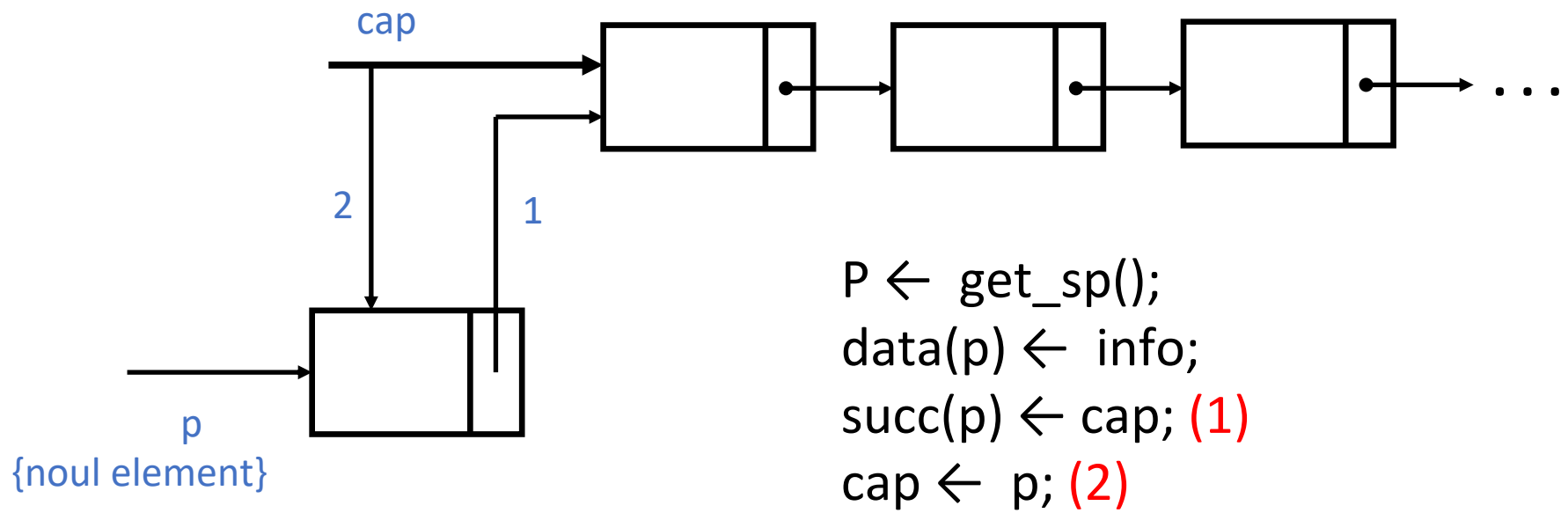


Operații cu liste liniare simplu înlănțuite

Inserarea unui element

Indiferent de locul inserării este necesară alocarea memoriei pentru noul element!

Inserarea în fața primului element



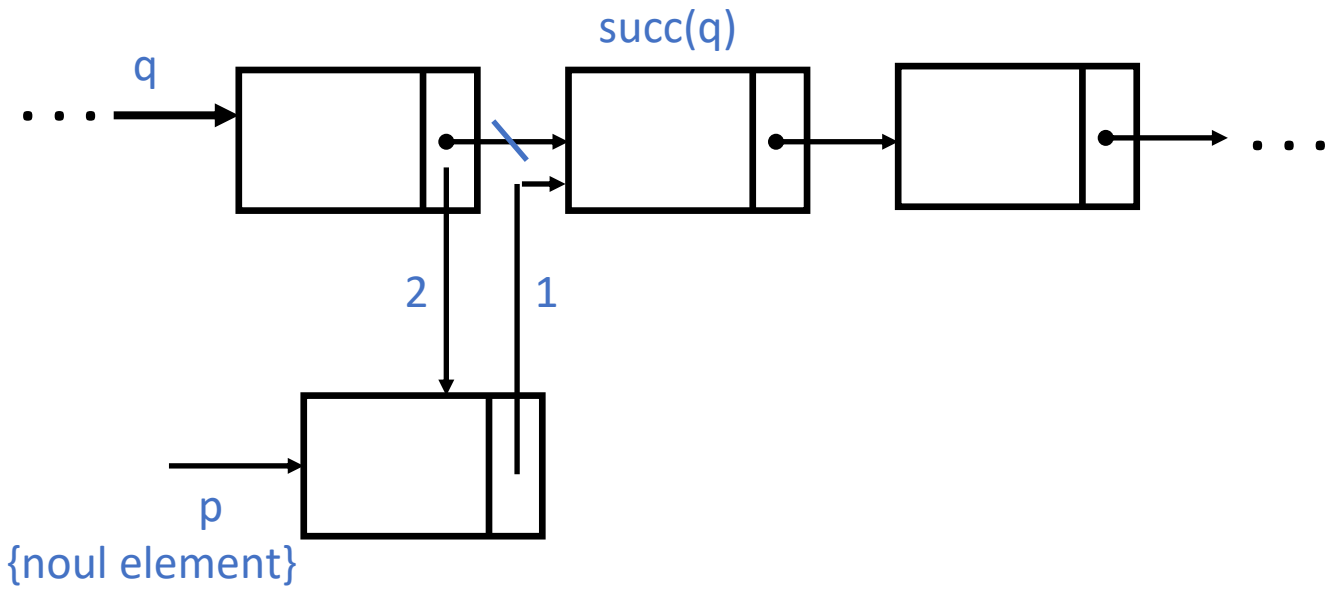
P ← get_sp();
data(p) ← info;
succ(p) ← cap; (1)
cap ← p; (2)

Complexitate O(1)

Operații cu liste liniare simplu înlănțuite

Inserarea unui element în interiorul listei

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



P ← get_sp();
data(p) ← info;
succ(p) ← succ (q); (1)
succ(q) ← p; (2)

Complexitate O(n)

Secvența este valabilă și pentru inserarea după ultimul element.

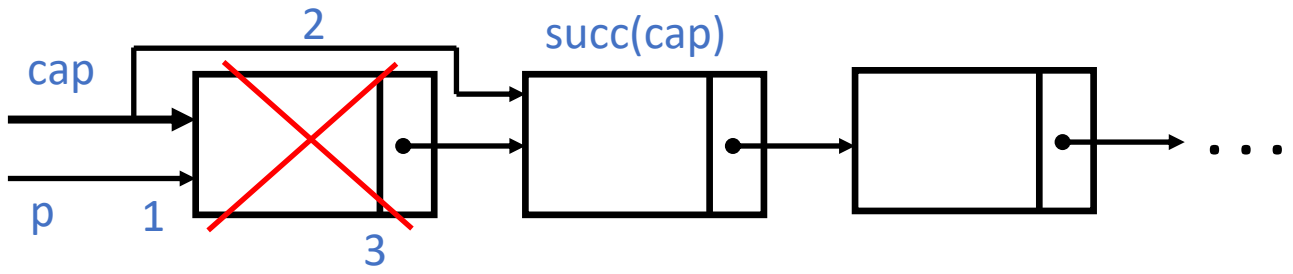
Se poate insera un nou nod în fața nodului de pointer q?

Operații cu liste liniare simplu înlănțuite

Stergerea unui element

Operația de ștergere implică refacerea legăturilor după detașarea nodului din listă și eliberarea zonei de memorie.

Ștergerea primului element



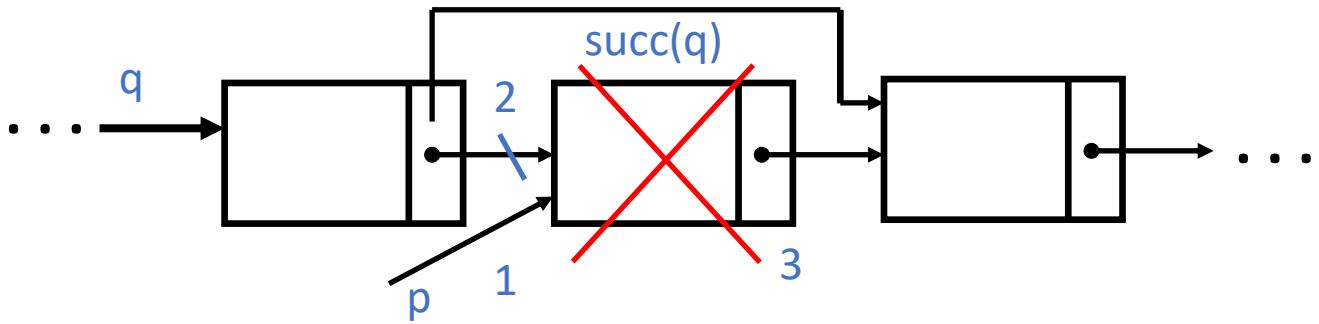
P ← cap; (1)
cap← succ(cap); (2)
free_sp(p); (3)

Complexitate O(1)

Operații cu liste liniare simplu înlănțuite

Ștergerea unui element din interiorul listei

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



p ← succ(q); (1)
succ(q) ← succ (p); (2)
free_sp(p); (3)

Complexitate O(n)

Secvența este valabilă și pentru ștergerea ultimului element.

Se poate șterge un nod de pointer q?

Exemplu: crearea unei liste liniare simplu inlantuite prin inserari in fata

```
struct Nod{
    int data;
    Nod * succ;
};
```

Inserare in fata primului element

```
void Insert(Nod *&cap, int val)
{//transfer prin referinta
    Nod *p;
    p=new Nod;
    p->data=val;
    p->succ=cap;
    cap=p;
}
```

Creare lista

```
void CreateList(Nod*& cap)
{//transfer prin referinta
    int n;
    cout << "introduceti valori (0- termina)!"
    << endl;
    cin >> n;
    while (n)
    {
        Insert(cap, n);
        cin >> n;
    }
}
```

Apel in main()

```
Nod *p=0;//initializarea!!
CreateList(p);
```

Exemplu: functie care afiseaza valoarea elementului din mijlocul listei fara sa numere elementele acesteia

Se considera doi pointeri pentru parcurgerea listei, dintre care unul se “deplaseaza” de doua ori mai repede. Initial ambii indica primul element. Cand pointerul “mai rapid” ajunge la sfarsitul listei, cel mai “lent” va fi pe nodul din mijloc.

```
Middle(cap)
    p←cap;
    q←cap;
    While(q≠0 and succ(q)≠0) Do
        p←succ(p);
        q←succ(succ(q));
    EndWhile
    Print data(p);
End
```

Discutie: ce se afiseaza daca lista are numar par de noduri?