# Structuri de date alocate dinamic

**LISTE SIMPLU ÎNLĂNŢUITE**

**LISTE DUBLU ÎNLĂNŢUITE**

**LISTE CIRCULARE**

# Sumar

## Competenţe generale

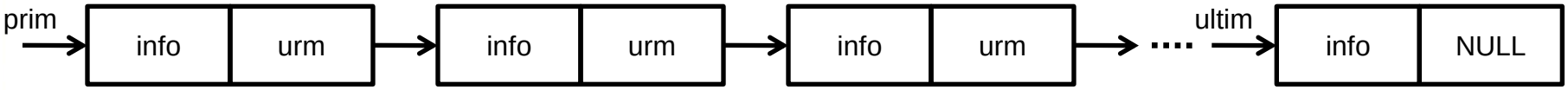
- *identificarea datelor care intervin într-o problemă şi aplicarea algoritmilor fundamentali de prelucrare a acestora*

## Competenţe specifice

- descrierea operaţiilor specifice listelor simplu înlănţuite şi elaborarea unor subprograme care să implementeze aceste operaţii
- analizarea în mod comparativ a avantajelor utilizării diferitelor metode de structurare a datelor necesare pentru rezolvarea unei probleme
- aplicarea în mod creativ a algoritmilor fundamentali în rezolvarea unor probleme concrete

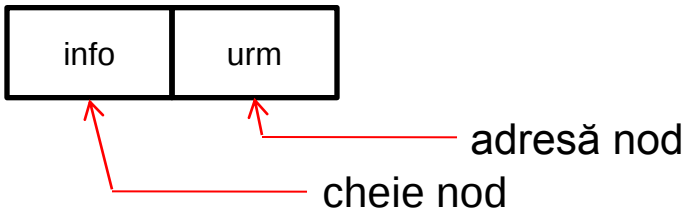
O listă simplu înlănţuită reprezintă o colecţie de noduri aflate într-o relaţie de ordine.

O listă siplu înlănţuită este o structură de forma:



Fiecare nod al listei conţine două informaţii:
- **`info`** reprezintă informaţia proproiu-zisă (cheia nodului);
- **`urm`** reprezintă informaţia de legătură (adresa nodului următor din listă).

Pentru a parcurge nodurile listei trebuie cunoscută adresa primului element al listei (**`prim`**) şi adresa ultimului element din listă (**`ultim`**).
Ultimul element din listă nu are succesor, ca urmare în câmpul de adresă se memorează constanta **`NULL`**.



**Definirea structurii unui nod**

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

Declarare variabile utile:

```
nod *prim, *ultim, *nou;
int x;
```

unde:

- **prim** - adresa primului element al listei;
- **ultim** - adresa ultimului element al listei;
- **nou** - adresa nodului care va fi creat;
- **x** - informaţia care va fi memorată în nod.

## Alocare şi eliberare de memorie

### *Alocarea memoriei*

Alocarea dinamică a spaţiului de memorie necesar pentru nodurile listei se face prin apelul operatorului `new`:

```
prim=new nod;          //pentru primul nod
```

sau:

```
nou=new nod;           //pentru celelalte ndoduri
```

## *Eliberarea memoriei*

Ştergerea listei din memorie, prin eliberarea spaţiului alocat nodurilor, se fece prin apelul operatorului **`delete`**:

```
delete prim;
delete ultim;
delete nou;
```

**Crearea listei**

- se creează primul nod al listei;
- celelalte noduri se adaugă la ultimul nod al listei.

***a). crearea primului nod***

- se alocă memorie în **HEAP** pentru primul nod:

```
prim=new nod;
```

- se completează câmpul de informaţie utilă:

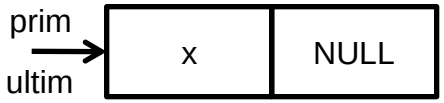
```
prim->info=x;
```

- se completează adresa către nodul următor:

```
prim->urm=NULL;
```

- se memorează în variabila pointer **ultim** adresa ultimului nod al liste:

```
ultim=prim;
```



***b). crearea celorlalte noduri***

- se alocă memorie în **HEAP** pentru noul nod:

```
nou=new nod;
```

- se completează câmpul de informaţie utilă:

```
nou->info=x;
```

- se completează adresa către nodul următor:

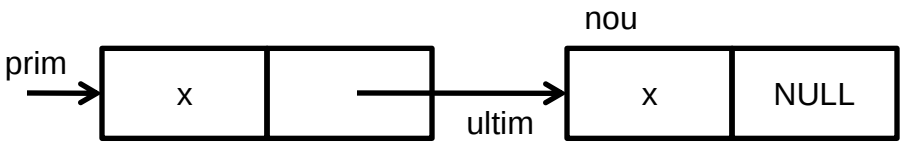
```
nou->urm=NULL;
```

- se stabileşte legătura între cele două noduri:

```
ultim->urm=nou;
```

- se memorează în variabila pointer **ultim** adresa ultimului nod al liste:

```
ultim=nou;
```



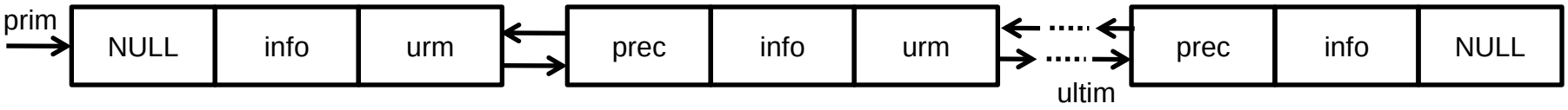
**Parcurgerea listei**

- se utilizează un pointer care adresează primul nod al listei.

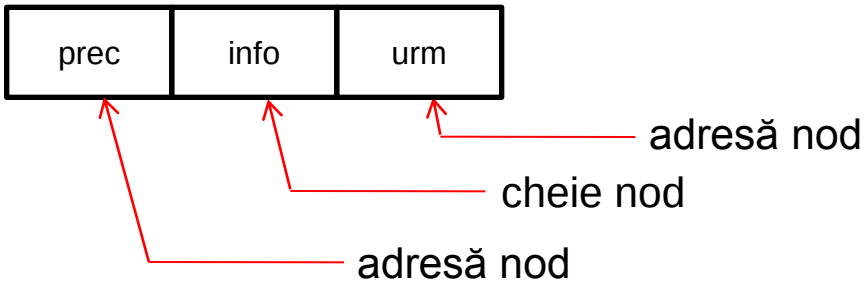
```
p=prim;
while(p!=NULL)      //sau  while(p)
{
      <prelucrează element>
      p=p->urm;
}
```

O listă dublu înlănţuită reprezintă o colecţie de noduri aflate într-o relaţie de ordine.

O listă dublu înlănţuită este o structură de forma:



Fiecare nod al listei conţine trei informaţii:
- **`info`** reprezintă informaţia proproiu-zisă (cheia nodului);
- **`prec`** reprezintă informaţia de legătură (adresa nodului precedent din listă);
- **`urm`** reprezintă informaţia de legătură (adresa nodului următor din listă).



**Definirea structurii unui nod**

```
struct nod
{
int info;
nod *prec,*urm;
};
```

Declarare variabile utile:

```
nod *prim, *ultim, *nou;
int x;
```

unde:

**prim** - adresa primului element al listei;
**ultim** - adresa ultimului element al listei;
**nou** - adresa nodului care va fi creat;
**x** - informaţia care va fi memorată în nod.

**Crearea listei**

- se creează primul nod al listei;
- celelalte noduri se adaugă la ultimul nod al listei.

***a). crearea primului nod***

- se alocă memorie în **HEAP** pentru primul nod:

```
prim=new nod;
```

- se completează câmpul de informaţie utilă:

```
prim->info=x;
```

- se completează adresa către nodul precedent:

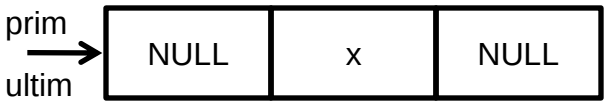
```
prim->prec=NULL;
```

- se completează adresa către nodul următor:

```
prim->urm=NULL;
```

- se memorează în variabila pointer **ultim** adresa ultimului nod al liste:

```
ultim=prim;
```



***b). crearea celorlalte noduri***

- se alocă memorie în **HEAP** pentru noul nod:

```
nou=new nod;
```

- se completează câmpul de informaţie utilă:

```
nou->info=x;
```

- se completează adresa către nodul precedent:

```
nou->prec=ultim;
```

- se completează adresa către nodul următor:

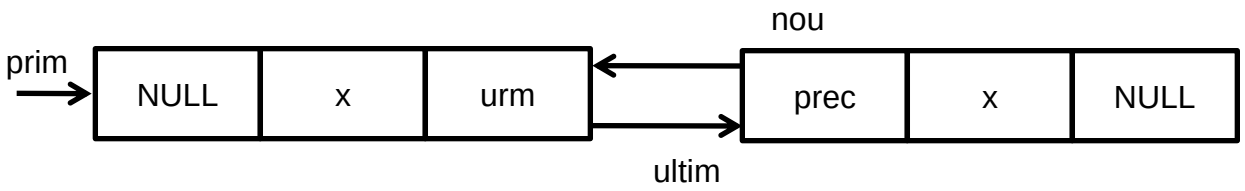
```
nou->urm=NULL;
```

- se stabileşte legătura între cele două noduri:

```
ultim->urm=nou;
```

- se memorează în variabila pointer **ult**im adresa ultimului nod al liste:

```
ultim=nou;
```



**Parcurgerea listei de la primul nod la ultimul nod**

- se utilizează un pointer către primul nod al listei.

```
p=prim;
while(p!=NULL)      //sau  while(p)
{
      <prelucrează element>
      p=p->urm;
}
```

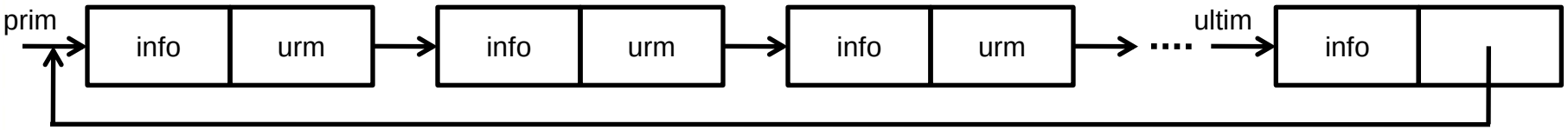
**Parcurgerea listei de la ultimul nod la primul nod**

- se utilizează un pointer către ultimul nod al listei.

```
q=ultim;
while(q!=NULL)      //sau  while(q)
{
      <prelucrează element>
      q=q->prec;
}
```

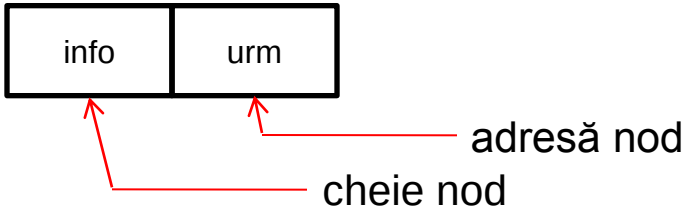
O listă circulară reprezintă o listă simplu înlănţuită în care ultimul element al listei reţine în loc de valoarea **`NULL`** adresa primului element al listei.

O listă circulară este o structură de forma:



Fiecare nod al listei conţine două informaţii:
- **`info`** reprezintă informaţia proproiu-zisă (cheia nodului);
- **`urm`** reprezintă informaţia de legătură (adresa nodului următor din listă).

Pentru a parcurge nodurile listei trebuie cunoscută adresa primului element al listei (**`prim`**).



**Crearea listei**

- se realizează în mod similar listelor simplu înlănţuite, cu deosebirea că la sfârşitul creării listei se realizează legătura între primul nod şi ultimul nod al listei:

```
ultim->urm=prim;
```

**Parcurgerea listei**

- se utilizează un pointer către primul nod al listei.

```
p=prim;
q=prim;
do
{
      <prelucrează element>
      p=p->urm;
} while(p!=q);
```

**Fişe de lucru**

- Întrebări liste simplu înlănţuite, liste dublu înlănţuite, liste circulare
- Aplicaţii liste simplu înlănţuite, liste dublu înlănţuite, liste circulare

## 7. Bibliografie şi webografie


1. Miloşecsu M., *Informatica. Manual pentru clasa a X*, Editura Didactică şi Pedagogică, Bucureşti, 2005
2. Mateescu G, Moraru P., *Informatica. Manual pentru clasa a X*, Editura Donaris, Sibiu, 2006
3. Popescu C., *Culegere de probleme de informatică*, Editura Donaris-Info, Sibiu, 2002
4. http://en.wikipedia.org/wiki/Pointer_(computer_programming)
5. http://www.cplusplus.com/articles/EN3hAqkS/
6. http://www.cplusplus.com/doc/tutorial/dynamic/
7. http://www.cplusplus.com/articles/G6vU7k9E/