Implementarea listelor simplu inlantuite

gr. B-2121, B-2122

Implementarea listelor simplu inlantuite

Continut

- Metode de implementare
- Declararea unei liste
- Clasificarea listelor
- Algoritmi pentru prelucrarea listelor simplu inlantuite

Metode de implementare

- **LISTA** – structura de date logica, liniara, cu date omogene, in care fiecare element are un succesor si un predecesor, exceptand primul element, care nu are decat succesor si ultimul element care nu are decat predecesor.
- **Implementarea listelor:**

  a) in functie de modul de alocare a memoriei interne:

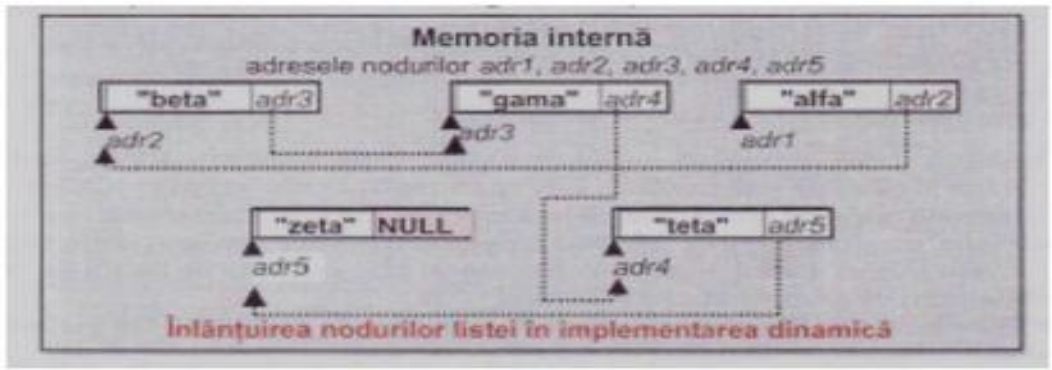
  - implementare statica (folosind vectori)
  - implementare dinamica (folosind pointeri)

  b) in functie de modul de aranjare a elementelor listei:

  - secventiala – numai statica
  - inlantuita – statica si dinamica

Implementarea prin alocare inlantuita

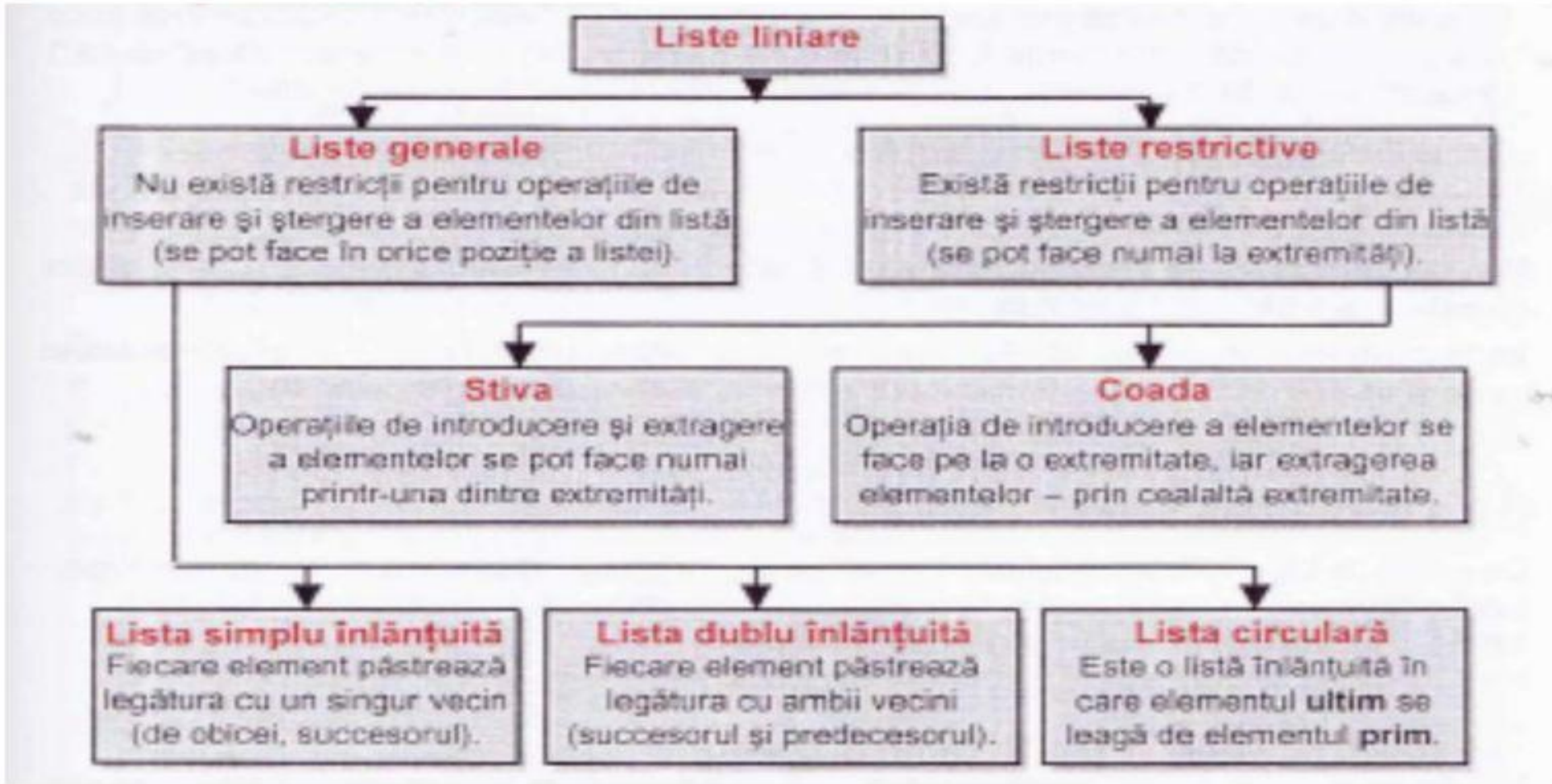
- Nodurile sunt aranjate aleatoriu in memorie.
- Necesita un mecanism prin care sa se precizeze ordinea reala a nodurilor adica:
  - pozitia primului nod (prim)
  - pozitia ultimului nod (ultim)
  - succesorul fiecarui nod (urm)



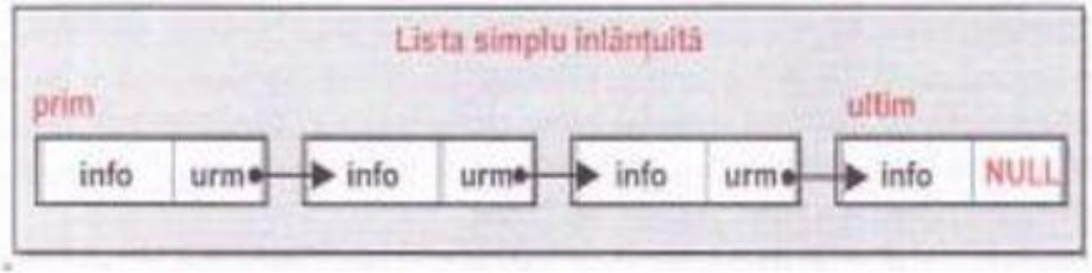
Înlănțuirea nodurilor listei în implementarea dinamică

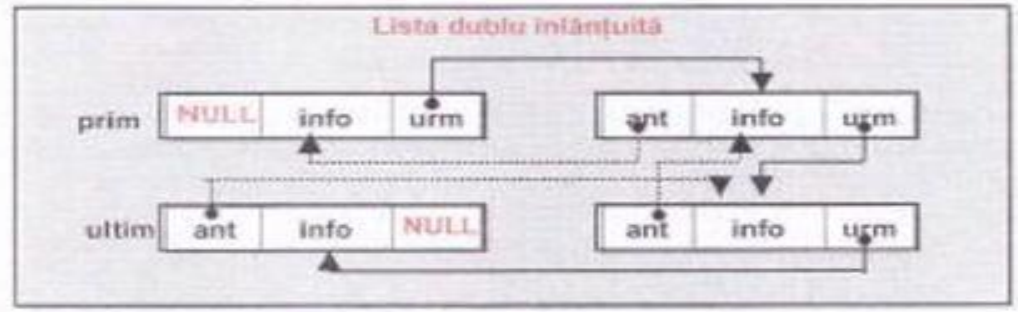
- Metoda folosita este ca un nod al listei sa contina doua tipuri de informatii:
  - informatia propriu-zisa
  - informatia de legatura – adresa urmatorului nod.

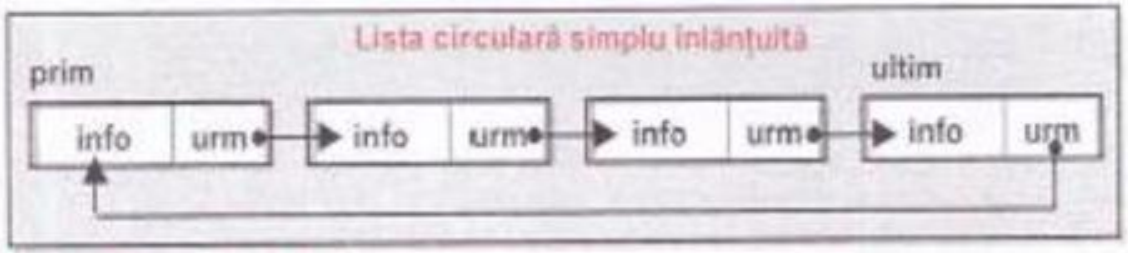
Clasificarea Listelor



Clasificarea listelor







Declararea listei

```
const unsigned NMAX=100;
typedef unsigned  adresa;
struct nod
{<tip_1>  <info_11>,<info_12>,.....,<info_1n>;
  ..................................
 <tip_m>  <info_m1>,<info_m2>,.....,<info_mn>;
  adresa urm;};
 nod lista [NMAX+1];
```

Algoritmi pentru prelucrarea listelor

Operatii:

- Initializarea
- Crearea
- Inserarea unui element
- Eliminarea unui element
- Parcurgerea
- Cautarea unui element
- Concatenarea
- Divizarea

Algoritmi pentru prelucrarea listelor

Se considera ca un nod al listei contine numai un camp cu informatie si campul pentru realizarea legaturii:

```
const unsigned NMAX=100;
typedef unsigned  adresa;

struct nod
{int info;
adresa urm;};
 nod lista [NMAX+1];

adresa prim, ultim, p;
unsigned nr_el, liber[NMAX];
int n;
```

unde:

prim – adresa primului nod

ultim – adresa ultimului nod

p – adresa nodului curent

n – valoarea atribuita campului info

nr_el – lungimea listei

liber – vector harta a nodurilor

- liber [p] are valoarea 1 daca nodul p este liber si 0 daca este ocupat

Algoritmi pentru prelucrarea listelor

Testarea listei:

a)Lista vida

```
int este_vida (adresa prim )
{ return prim = = NULL; }
```

b)Lista plina

```
int este_plina ( )
{ return nr_el = = NMAX; }
```

Algoritmi pentru prelucrarea listelor

Initializarea listei: se creeaza lista vida

```
void init ( adresa &prim, adresa &ultim )
{ ultim = prim = NULL;
  nr_el = 0;
  for (  p = 1; p < = NMAX; p ++ )
        liber [ p ] = 1;
}
```

Algoritmi pentru prelucrarea listelor

Alocarea memoriei: se gaseste prima pozitie libera si se aloca memorie pentru noul nod.

```
adresa aloc_mem ( )
{
  for ( p = 1; ! liber [ p ]; p ++ ) ;
  liber [ p ] = 0 ; nr_el ++;
}
```

Algoritmi pentru prelucrarea listelor

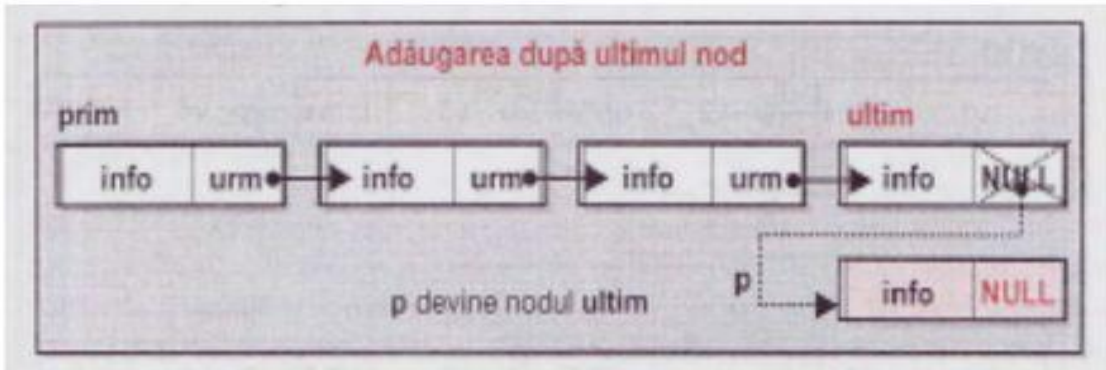
Adaugarea primului nod:

```
void adaug_prim ( adresa &prim, adresa &ultim, int n )
{ prim = aloc_mem ( );
  lista [ prim ] . info =  n;
  lista [ prim ] .urm = NULL;
  ultim = prim;
}
```

Algoritmi pentru prelucrarea listelor

Adaugarea dupa ultimul nod:

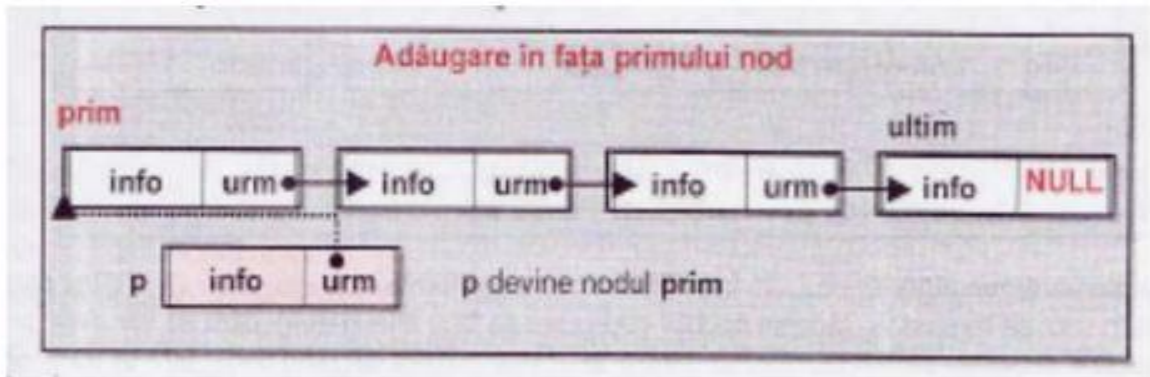
```
void adaug_dupa_ultim ( adresa &prim, adresa &ultim, int n )
{ p = aloc_mem ( );
  lista [ p ] . info =  n;
  lista [ p ] .urm = NULL;
  lista [ultim ].urm = p;
  if (este_vida ( prim ))   prim = p;
  ultim = p;
}
```



Algoritmi pentru prelucrarea listelor

Adaugarea in fata primului nod:

```
void adaug_inainte_prim ( adresa &prim, adresa &ultim, int n )
{ p = aloc_mem ( );
  lista [ p ] . info =  n;
  lista [ p ] .urm = prim;
  if (este_vida ( prim ))   ultim = p;
  prim = p;
}
```

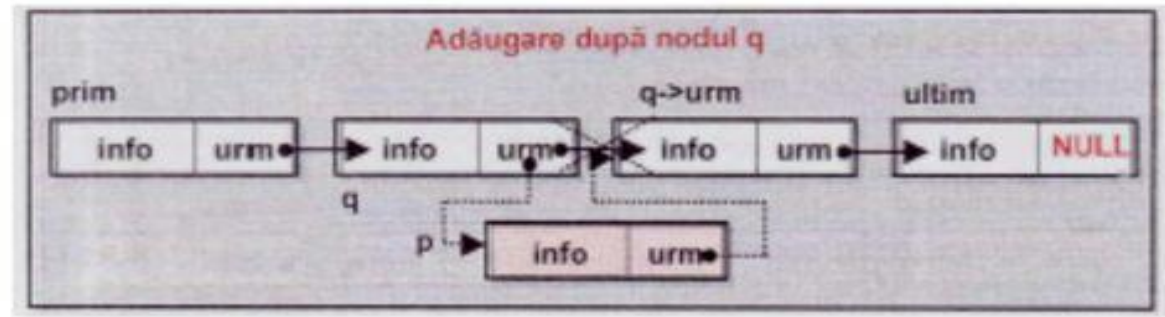


Algoritmi pentru prelucrarea listelor

Adaugarea in interiorul liste:

a) Dupa nodul q

```
void adaug_dupa ( adresa q, adresa &ultim, int n )
{ p = aloc_mem ( );
  lista [ p ] . info =  n;
  lista [ p ] .urm = lista [ q ].urm;
  lista [ q ] .urm = p;
  if ( lista [ p ] .urm = = NULL )   ultim = p;
  }
```

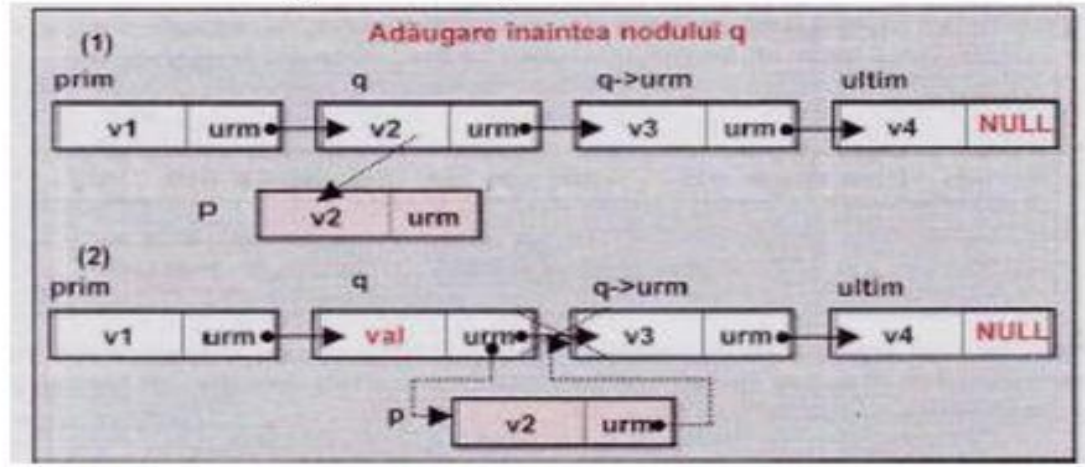


Algoritmi pentru prelucrarea listelor

Adaugarea in interiorul liste:

b) Inainte de nodul q

```
void adaug_in_fata ( adresa q, adresa &ultim, int n )
{ p = aloc_mem ( );
  lista [ p ] . info =  lista [ q ] . info;
  lista [ q ] . info = n;
  lista [ p ] .urm = lista [ q ].urm;
  lista [ q ] .urm = p;
  if ( lista [ p ] .urm = = NULL )
ultim = p;
  }
```



Algoritmi pentru prelucrarea listelor

Parcurgerea listei:

```
void parcurge ( adresa prim )
{ for ( p = prim; p ! = NULL; p = lista [ p ] . urm )
    // se prelucreaza lista [ p ] . info;
```

Implementarea listelor simplu inlantuite

Vezi fisierul - Sarcini ListeSimpleInlaantuite