COADA-IMPLEMENTARE DINAMICĂ

CURS 7- 13.04.2021

Titular: Şef. Lucr. Dr. Mat. Cărbureanu Mădălina

Universitatea Petrol-Gaze din Ploieşti

OBIECTIVE:

- **NOȚIUNEA DE COADĂ (QUEUE);**
- **MODURI IMPLEMENTARE COADĂ;**
- **PRINCIPIUL FIFO;**
- **MOD DE LUCRU;**
- **DECLARARE COADĂ;**
- **TIPURI DE OPERAȚII;**
- **EXEMPLU APLICAȚIE;**
- **APLICAȚII PROPUSE.**

- ❑ Implementare statică;
- ❑ Implementare dinamică;

- ❑ Listă simplu înlănțuită de tip COADĂ, conform FIFO;

- ❑ **Operații de bază;**
- ❑ Alte tipuri de operații;

NOȚIUNEA DE COADĂ (QUEUE)

Coada-tip particular de listă;

- **Coada ⇨ caz particular de listă simplu înlănțuită;**
- **Coada ⇨ operația de adăugare a unui element se realizează la un capăt al cozii (la sfârșitul cozii), iar operația de ștergere se realizează la celălalt capăt al cozii (la începutul cozii).**
- **Exemple de cozi: coada de clienți la o bancă, coada de așteptare la un service auto, etc.**
- **Principiu ⇨ FIFO (First-In, First-Out);**
- **TDA Coada.**

MODURI IMPLEMENTARE COADĂ

Se utilizează?

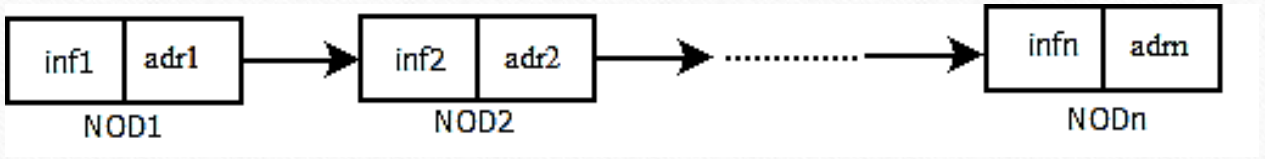
- **Implementare statică** ⇔ **implementare secvențială;**
- **Implementare statică** ⇨ **elemente vecine sunt memorate automat la adrese consecutive de memorie;**
- **Implementare statică;**
  - ❑ Avantaj;
  - ❑ Dezavantaj;

MODURI IMPLEMENTARE COADĂ

Se utilizează? COADA-listă simplu înlănțuită!

- **Implementare dinamică** ⇨ **implementare înlănțuită;**
- **Implementare dinamică** ⇨ **elemente vecine nu sunt memorate automat la adrese consecutive, memorarea fiecărui element se realizează în diferite zone de memorie;**

  ⇨ **fiecare nod al listei prezintă două câmpuri;**

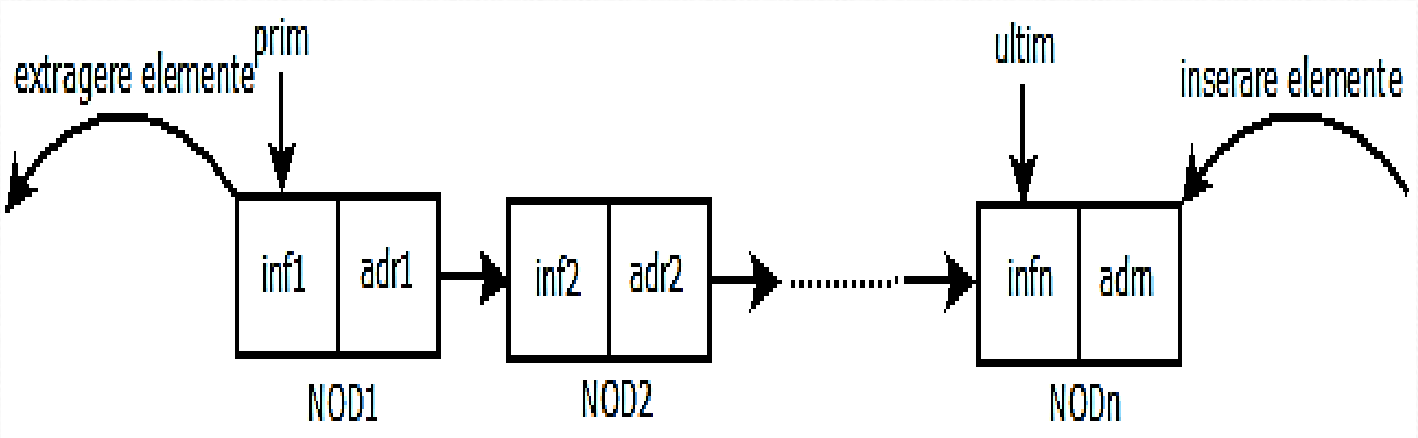


prim **ultim**

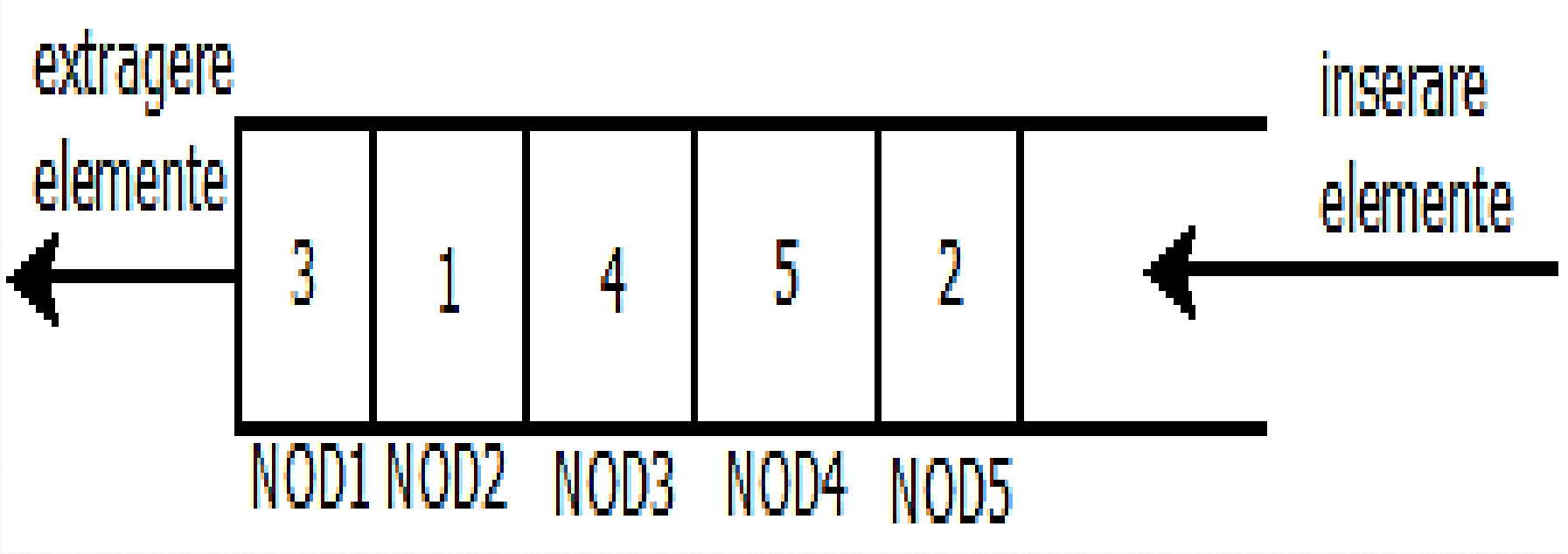
- **Implementare dinamică**
  - ❑ Avantaj;
  - ❑ Dezavantaj;

PRINCIPIUL FIFO

- **FIFO- PRIMUL ELEMENT INTRODUS (PRIMUL SOSIT) ESTE ȘI PRIMUL ELEMENT EXTRAS (PRIMUL SERVIT);**



MOD DE LUCRU L.S.Î DE TIP COADĂ CONFORM FIFO



DECLARARE COADĂ

```
typedef  struct lista
{int inf;
 struct lista*leg;
}COADA;
```

TIPURI DE OPERAȚII

COADA- listă simplu înlănţuită generală!

- **Operaţii de bază:**
  - **Crearea cozii (conform FIFO);**
  - **Parcurgerea elementelor cozii (de la primul element către ultimul element, conform FIFO) ;**
  - **Ștergerea primului element din coadă (conform FIFO).**
  - **Adăugarea unui element la sfârșitul cozii, respectiv după ultimul element al cozii (conform FIFO);**
- **Alte tipuri de operaţii ⇨ similare celor din cursul 6.**

OPERAȚII DE BAZĂ

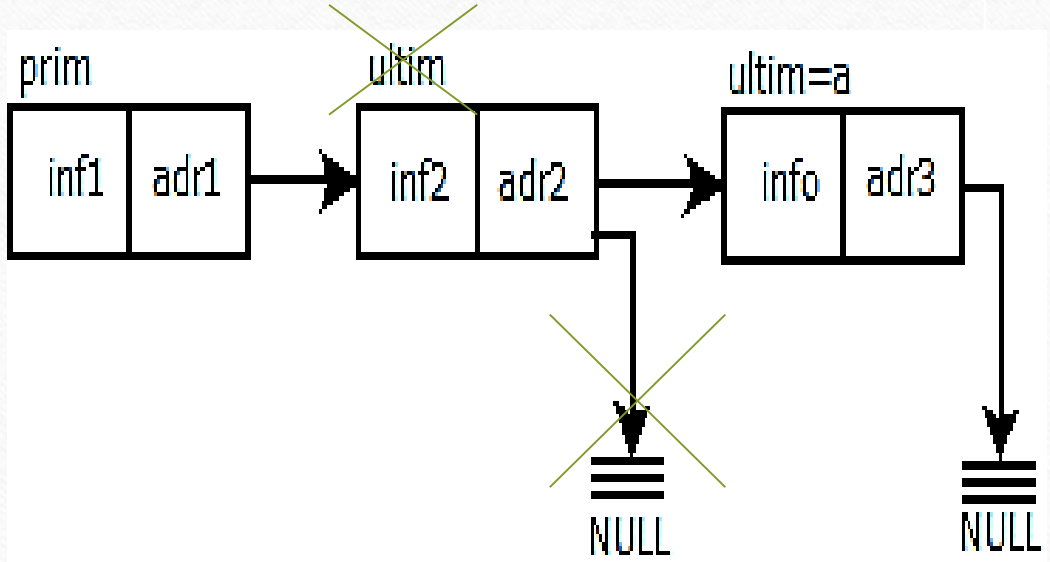
- **Crearea cozii ⇨ adăugarea treptată de elemente, respecând FIFO, prin citirea informației utile de la tastatură;**
- **Pași:**
  - se declară nodul ce se dorește a se adăuga;
  - se declară un pointer de tip COADA numit *prim*, necesar pentru a cunoaște la orice moment de timp adresa primului element (nod) din coadă;
  - se declară un pointer de tip COADA numit *ultim*, necesar pentru a cunoaște la orice moment de timp adresa ultimului element (nod) din coadă;
  - se citește de la tastatură informația utilă a fiecărui element component al cozii (*info*);

OPERAȚII DE BAZĂ

- se evaluează condiția "*atâta timp cât info este diferit de 0*" ; dacă condiția este indeplinită se realizează următoarele operații:
  - se alocă memorie pentru elementul ce se dorește a se adăuga în coadă;
  - se completează câmpul informație al elementului *a* și se realizează legătura la *NULL* pentru elementul introdus *a*;
  - dacă nu mai există niciun alt element în coadă, atunci elementul introdus *a* devine atât primul, cât și ultimul element al cozii; altfel (mai există elemente în coadă), se realizează legătura ultimului element din coadă cu noul element introdus *a,* iar ultimul element din coadă devine acum *a;*
  - se citește de la tastatură informația utilă, până în momentul în care valoarea acesteia este 0;
- se returnează adresa de bază a cozii, respectiv adresa primului element al cozii.

OPERAȚII DE BAZĂ⇨CREAREA COZII

```
COADA*creare()
{COADA*prim,*ultim,*a;
int info; prim=NULL; ultim=NULL;
cout<<"\nInfo. utila elem. coada:";
cin>>info;
while(info!=0)
{a=(COADA*)malloc(sizeof(COADA));
 a→inf=info;
 a→leg=NULL;
 if(prim==NULL)
  {prim=a; ultim=a;}
  else {ultim→leg=a; ultim=a;}
cout<<"\nInfo. utila elem. coada:";
cin>>info;
 }
return prim;
}
```

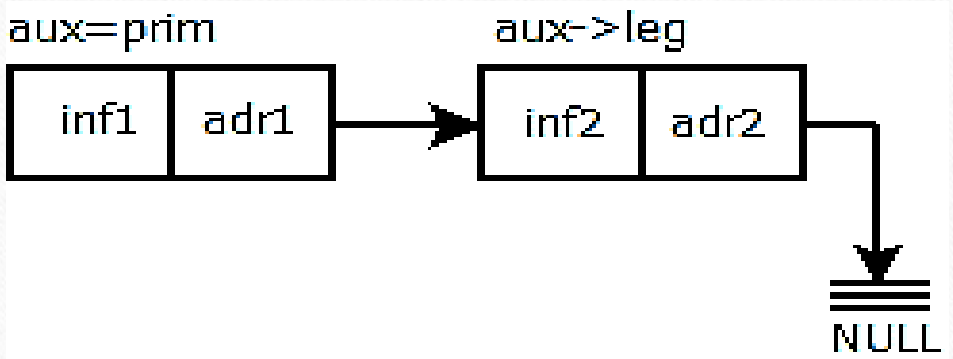


OPERAȚII DE BAZĂ

- **Operația de parcurgere** ⇨ parcurgerea fiecărui nod în parte, pornind de la primul element până la ultimul element al cozii și afișarea informației utile asociate fiecărui element (nod) parcurs astfel:
  - se declară un nou nod *aux* de tip pointer către *COADA*;
  - dacă există elemente în coadă, atunci se realizează următorele operații:
    - *aux* devine primul nod din coadă;
    - se evaluează condiția "*atâta timp cât aux este diferit de NULL*"; dacă condiția este îndeplinită se afișează informația nodului curent și se trece la următorul element din coadă a cărui informație utilă este afișată.

OPERAȚII DE BAZĂ ⇨ PARCURGEREA ELEMENTELOR COZII

```
void parcurgere(COADA*prim)
{COADA*aux;
 if(prim!=NULL)
     {aux=prim;
      while(aux!=NULL)
          {cout<<aux→inf;
            aux=aux→leg;
          }
      }
}
```

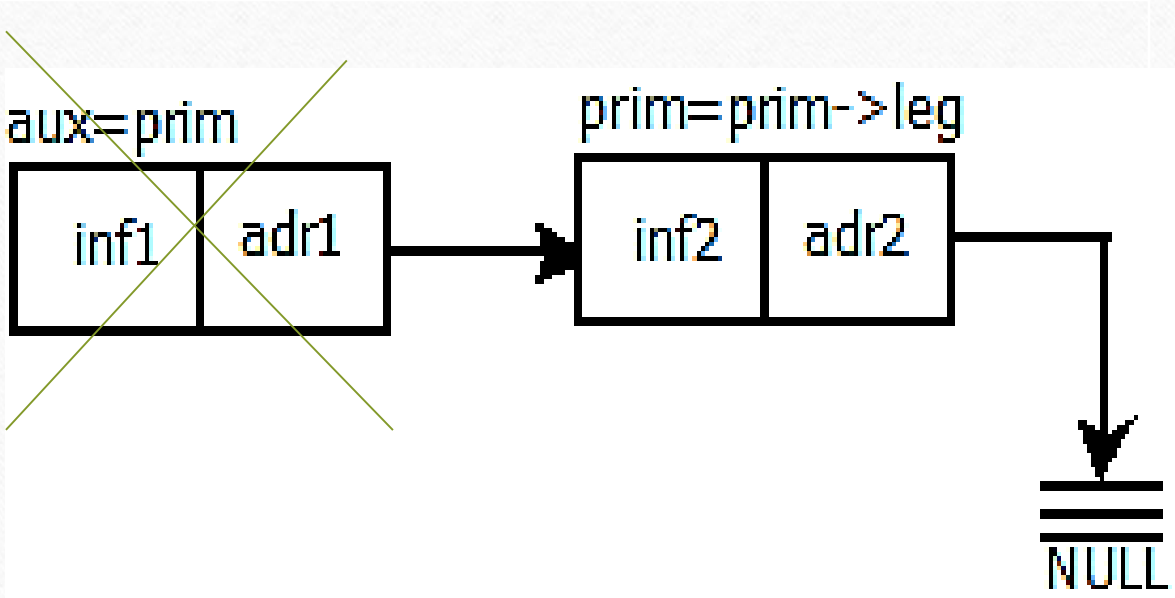


OPERAȚII DE BAZĂ

- **Ștergerea primului element din coadă (conform FIFO)** presupune parcurgerea următorilor pași:
  - se declară un nou nod *aux* de tip pointer către COADA*;*
  - noul nod *aux* devine primul element al cozii;
  - adresa de bază devine adresa următorului element din coadă, respectiv a primului element din coadă;
  - se șterge primul element din coadă (se eliberează memoria ocupată de primul element)*;*
  - se returnează coada (noul capăt al cozii) fără primul element al său.

OPERAȚII DE BAZĂ⇨ȘTERGEREA PRIMULUI ELEMENT

```
COADA*sterg_prim(COADA
*prim)
{COADA*aux;
aux=prim;
prim=prim→leg;
free(aux);
return prim;}
```

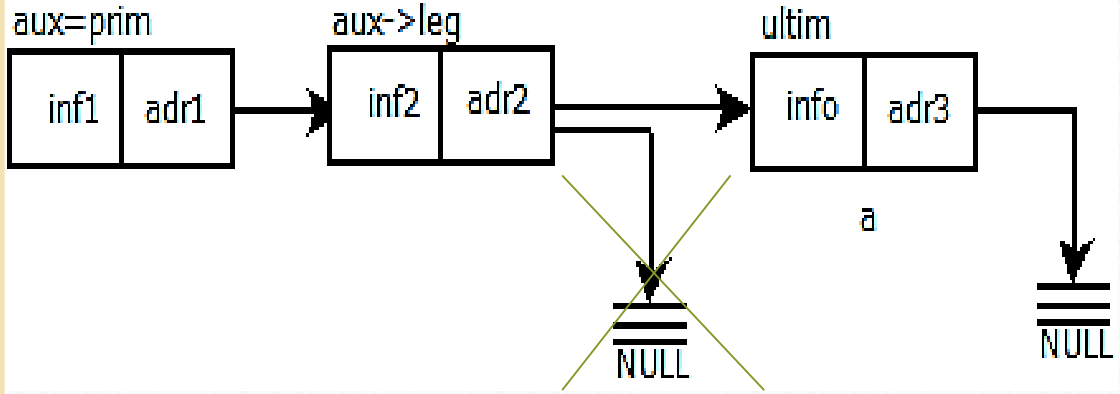


OPERAȚII DE BAZĂ

- **Operaţia de adăugare a unui element la sfârşitul cozii, respectiv după ultimul element din coadă, presupune:**
  - declararea un nou nod *aux* de tip pointer către COADA*;*
  - declararea noului nod *a* de tip pointer către COADA (*COADA*a*), acesta reprezentând nodul ce se doreşte a se adăuga la sfârşitul cozii;
  - alocarea de memorie pentru noul nod *a;*
  - completarea informaţiei utile pentru nodul *a*;
  - noul nod *aux* devine primul element al cozii*;*
  - se evaluează condiţia "*atâta timp cât nu am ajuns la ultimul element din coadă*" ; dacă condiţia este îndeplinită, se trece la următorul element din coadă, se realizează legătura ultimului element din coadă cu noul element *a*, element ce devine ultimul element din coadă prin realizarea legăturii la NULL;
  - se returnează coada, pornind de la primul către ultimul element, cu noul element adăugat la sfârşitul cozii.

OPERAȚII DE BAZĂ ⇨ ADĂUGAREA UNUI ELEMENT LA SFÂRȘITUL COZII

```
COADA*adaug_sf(COADA* prim, int
info)
{COADA*a,*aux;
a=(COADA*)malloc(sizeof(COADA));
a→inf=info; aux=prim;
while(aux→leg!=NULL)
    aux=aux→leg;
    aux→leg=a;
    a→leg=NULL;
    return prim;
}
```



EXEMPLU APLICAȚIE ⇨CONCATENAREA A DOUĂ COZI

```
#include<iostream.h>.....
#include<alloc.h>
typedef struct lista
{int inf;
struct lista*leg;
}COADA;
COADA*creare()
{COADA*prim,*ultim,*a;
int info;
prim=NULL; ultim=NULL;
cout<<"\nInfo. utila elem.
coada:";cin>>info;
while(info!=0)
{a=(COADA*)malloc(sizeof(COADA));
 a→inf=info;  a→leg=NULL;
 if(prim==NULL)
  {prim=a; ultim=a;}
  else {ultim→leg=a; ultim=a;}
 cout<<"\nInfo. utila elem.
coada:";cin>>info;}
return prim;}
```

```
void afisare(COADA*prim)
{COADA*aux;
  if(prim!=NULL)
    {aux=prim;
     while(aux!=NULL)
         {cout<<aux→inf;
          aux=aux→leg;
         }
     }
}
COADA*concatenare(COADA*x,CO
ADA*y,COADA*z)
{COADA*aux;
z=x;
aux=z;
while(aux→leg!=NULL)
  aux=aux→leg;
  aux→leg=y;
  return x;
}
```

```
void main()
{clrscr();
COADA*x,*y,*z;
 x=creare();
 cout<<"\nCOADA1 ESTE:";
 afisare(x);
 y=creare();
 cout<<"\nCOADA2 ESTE:";
 afisare(y);
 z=concatenare(x,y,z);
 cout<<"\nCOADA
OBTINUTA PRIN
CONCATENARE ESTE:";
 afisare(z);
 getch();
}
```

APLICAȚII PROPUSE

- **Testarea aplicațiilor 6.2.1, 6.2.2 și 6.2.3, pag. 110-115;**
- **Testarea celorlalte tipuri de operații (de adăugare și de ștergere), pag. 103-109;**
- **Aplicații propuse: 8, 9 și 14, pag. 116.**
- **Obs: Se va utiliza cartea "*Elemente de proiectarea algoritmilor. Ghid teoretic și practic*", autor Șef lucr. dr. mat. Cărbureanu Mădălina, Editura Univeristății Petrol-Gaze din Ploiești, 2021.**

Să vă fie de folos și spor la lucru!