

Universitatea "Politehnica" din București
Facultatea de Electronică, Telecomunicații și
Tehnologia Informației

Structuri de Date și Algoritmi (limbajul C)

Curs 3 – Liste de date

Prof. Bogdan IONESCU

2015-2016

Cuprins

- 3.1. Lucrul cu liste
- 3.2. Lucrul cu stive
- 3.3. Lucrul cu cozi

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016

1/96

Structuri complexe de date

> Limbajul C pune la dispoziția programatorilor o diversitate foarte mare de modalități de reprezentare a informației:

- *tipuri de date de bază* (ex. **int**, **float**, **char**, **void**);
- *constante* (ex. **#define**, **const**);
- *tipuri omoloage* (ex. **typedef**);
- *tipuri de date structurate* (matrice, șiruri de caractere);
- *tipuri de date compuse* (**struct**, **union**, **enum**);
- etc.

... totuși, problemele de calcul complexe necesită de cele mai multe ori pentru implementare **formalizarea unor modalități noi de reprezentare a informației**;

> aceste modalități nu sunt definite de limbaj, sunt doar o convenție de reprezentare a informației.

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016

2/96

3.1. Lucrul cu liste

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016

3/96

Enunț: să se realizeze un program ce permite stocarea și manipularea paginilor unei prezentări electronice (slides).

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016

4/96

Enunț: să se realizeze un program ce permite stocarea și manipularea paginilor unei prezentări electronice (*continuare*).

> să analizăm bine problema ...

(1) **cum stocăm informația?**

> informația de bază este o pagină ("slide"):

- text;
- valori;
- matrice (ex. imagini);
etc.

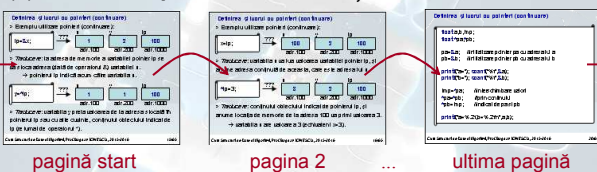
> acestea se repetă pentru valori diferite ale datelor/informațiilor afișate de acesta;

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016

5/96

P Enunț: să se realizeze un program ce permite stocarea și manipularea paginilor unei prezentări electronice (*continuare*).

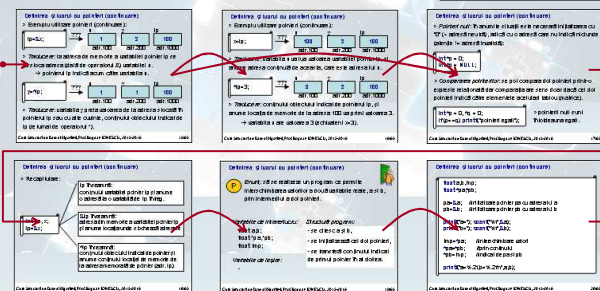
(2) cum este reprezentată informația?



- > paginile sunt înălțuite după o anumită ordine temporală;
- > exista o pagină de **start** (fără predecesor) și o pagină de **sfârșit** (fără succesori);

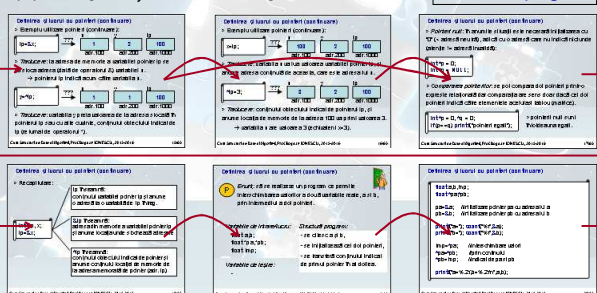
P Enunț: să se realizeze un program ce permite stocarea și manipularea paginilor unei prezentări electronice (*continuare*).

(3) ce operații trebuie să putem efectua? **ștergere pagină**



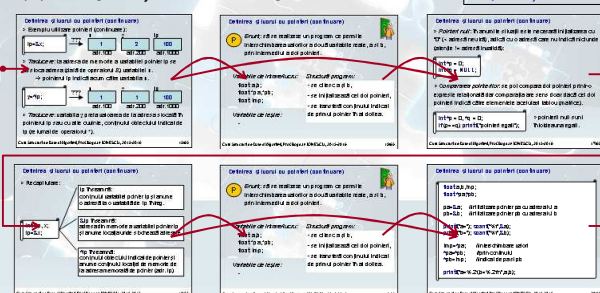
P Enunț: să se realizeze un program ce permite stocarea și manipularea paginilor unei prezentări electronice (*continuare*).

(3) ce operații trebuie să putem efectua? **inserare pagină**



P Enunț: să se realizeze un program ce permite stocarea și manipularea paginilor unei prezentări electronice (*continuare*).

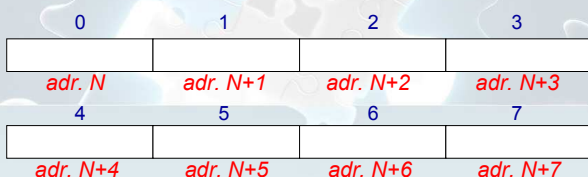
(3) ce operații trebuie să putem efectua? **repoziționare**



P Enunț: să se realizeze un program ce permite stocarea și manipularea paginilor unei prezentări electronice (*continuare*).

> ce soluții avem pentru a putea implementa o astfel de reprezentare în limbajul C?

> putem folosi vectori?



Liste simplu înălțuite de date

- > definim (formal) o **structură dinamică** de date ce permite toate aceste operații: **lista înălțuită de date**;
- > aceasta conține:
 - **noduri (date)**: **colecție de variabile** ce stochează practic informația din listă și asigură relaționarea elementelor listei.

Definire nod listă:

```
struct NOD
{
    <lista variabile>;
    ...
};
```

am definit tipul de date **NOD** prin intermediul unei structuri; aceasta permite stocarea informațiilor din listă prin **variabilele** definite;

Liste simplu înlănțuite de date (continuare)

> definim (formal) o *structură dinamică* de date ce permite toate aceste operații: *lista înlănțuită de date*;

> aceasta conține:

- **noduri (date)**: *colecție de variabile* ce stochează practic informația din listă și asigură relaționarea elementelor listei.

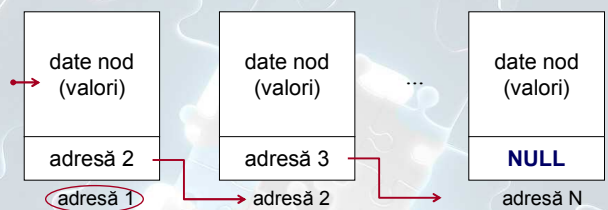
Definire nod listă (continuare):

```
struct NOD
{
    <lista variabile>;
    struct NOD *urmatorulNOD;
};
```

Cum definim relaționarea cu un alt element de tip nod?

variabila **urmatorulNOD** este un pointer la o structură de tip **NOD** (indică un alt nod);

Liste simplu înlănțuite de date (continuare)

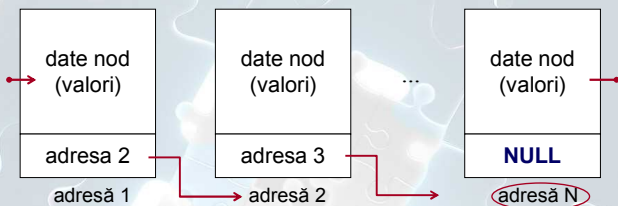


> cazuri particulare:

- **primul nod**: nu este relaționat anterior cu nici un alt nod (este practic începutul listei);

> este practic relaționat direct și indirect cu restul de noduri, accesarea întregii liste se poate face astfel prin el.

Liste simplu înlănțuite de date (continuare)



> cazuri particulare (continuare):

- **ultimul nod**: nu este relaționat următor cu nici un alt nod (este practic sfârșitul listei – **urmatorulNOD == NULL**);

> este la fel de important precum primul nod pentru a cunoaște unde se termină lista.

Liste simplu înlănțuite de date (continuare)

Exemplu definire listă:

```
struct NOD
{
    int x;
    struct NOD *urmatorulNOD;
};
```

elementul unei liste este definit prin structura **NOD** ce conține o variabilă **x** de tip **int** și pointerul **urmatorulNOD** la o structură de tip **NOD**;

variabila **primaMeaLista** este un pointer la o structură de tip **NOD** și constituie practic primul element din listă;

> De ce este necesar ca primul element din listă să fie definit ca pointer și nu ca variabilă normală?

lista trebuie să fie **dinamică** (se alocă și se șterg elemente).

Liste simplu înlănțuite de date (continuare)

Exemplu definire listă (continuare):

```
struct NOD
{
    int x;
    struct NOD *urmatorulNOD;
};

struct NOD *primaMeaLista;
```

primaMeaLista
#?%43
3467123

> **Efect**: în memorie se alocă o locație pentru pointer-ul **primaMeaLista** ce va stoca o adresă la o structură de tip **NOD**.

> Ce valoare are implicit pointerul **primaMeaLista**?
lista (pointerul primului element) trebuie inițializată.

Liste simplu înlănțuite de date (continuare)

#1 Crearea listei

```
struct NOD *CreareLista(int x)
{
    //Pasul 1: trebuie să alocăm memorie
    //pentru valorile unui nod, cu a cărei
    //adresă o sa initializam primaMeaLista
```

crearea primului element din listă și inițializare valori;

funcția **CreareLista** primește la intrare valoarea de inițializare pentru **x** și returnează un pointer la o structură de tip **NOD**;

```
struct NOD *tmp;
tmp=(struct NOD *)malloc(sizeof(struct NOD));
```

tmp este un pointer la o structură de tip **NOD**, se alocă memorie cu **malloc** și adresa obținută este stocată în **tmp**;

Liste simplu înlănțuite de date (continuare)

#1 Crearea listei (continuare)

```
struct NOD *CreareLista(int x)
{
    struct NOD *tmp;
    tmp=(struct NOD *)malloc(sizeof(struct NOD));
    if (tmp==NULL)
    {
        printf("Memoria nu a putut fi alocata");
        return NULL;
    }
    //Pasul 2: initializam valorile lui tmp
    tmp->x=x;
    tmp->urmatorulNOD=NULL;
}
```

verificăm dacă memoria a putut fi alocată;

atenție că **tmp** este pointer: **tmp->x** este inițializat cu **x** iar legătura la următorul nod este inițializată cu **NULL**;

Liste simplu înlănțuite de date (continuare)

#1 Crearea listei (continuare)

```
struct NOD *CreareLista(int x)
{
    struct NOD *tmp;
    tmp=(struct NOD *)malloc(sizeof(struct NOD));
    if (tmp==NULL)
    {
        printf("Memoria nu a putut fi alocata");
        return NULL;
    }
    tmp->x=x;
    tmp->urmatorulNOD=NULL;
    //Pasul 3: returnăm adresa lui tmp
    return tmp;
}
```

funcția returnează adresa noului nod alocat; în urma terminării funcției, se dezalocă variabilele locale (și anume **tmp**).

Liste simplu înlănțuite de date (continuare)

#1 Crearea listei (continuare)

```
struct NOD *CreareLista(int x)
{
    struct NOD *tmp;
    tmp=(struct NOD *)malloc(sizeof(struct NOD));
    if (tmp==NULL)
    {
        printf("Memoria nu a putut fi alocata");
        return NULL;
    }
    tmp->x=x;
    tmp->urmatorulNOD=NULL;
    return tmp;
}
```

Cum se apelează funcția în program?

```
struct NOD
{
    int x;
    struct NOD *urmatorulNOD;
} *primaMeaLista;
primaMeaLista=CreareLista(10);
```

Liste simplu înlănțuite de date (continuare)

#2 Populare valori listă

determinăm numărul de valori dorite în listă;

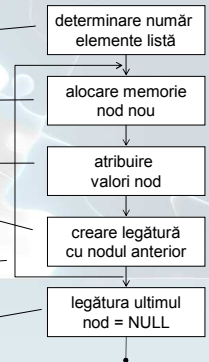
alocăm memorie pentru un nod nou pe care îl vom adăuga;

atribuire valori nod;

realizăm legătura nodului nou cu nodul anterior;

repetăm procesul până obținem numărul dorit de elemente;

ultimul nod adăugat devine nod final cu legătura NULL;

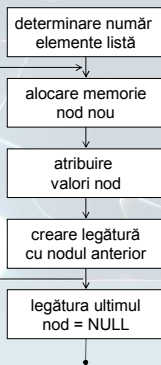


Liste simplu înlănțuite de date (continuare)

#2 Populare valori listă (continuare)

```
void PopulareListaGoala(struct NOD *prim)
{
    //variabile
    int n;
    struct NOD *nodNou, *nodCurent;

    strategie
    {
        nod 1      nod nou
        NULL       NULL
        adresă 1   adresă 2
        ↑         ↑
        prim      nodNou
    }
}
```

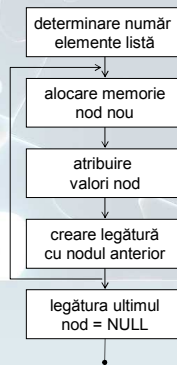


Liste simplu înlănțuite de date (continuare)

#2 Populare valori listă (continuare)

```
void PopulareListaGoala(struct NOD *prim)
{
    //variabile
    int n;
    struct NOD *nodNou, *nodCurent;

    strategie
    {
        nod 1      nod 2      nod nou
        adresă 1   NULL       NULL
        adresă 2   ↑         adresă 3
        prim      nodNou     nodNou
    }
}
```

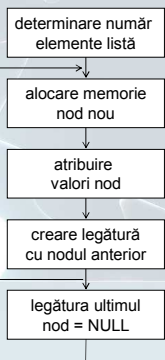


Liste simplu înlănțuite de date (continuare)

#2 Populare valori listă (continuare)

```
void PopulareListaGoala(struct NOD *prim)
{
    //variabile
    int n;
    struct NOD *nodNou, *nodCurent;

    strategie
    {
        nod 1      nod nou
        NULL      NULL
        adresă 1   adresă 2
        prim      nodCurent nodNou
    }
}
```

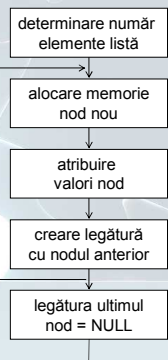


Liste simplu înlănțuite de date (continuare)

#2 Populare valori listă (continuare)

```
void PopulareListaGoala(struct NOD *prim)
{
    //variabile
    int n;
    struct NOD *nodNou, *nodCurent;

    strategie
    {
        nod 1      nod 2      nod nou
        adresă 2   NULL      NULL
        adresă 1   adresă 2   adresă 3
        prim      nodCurent nodCurent nodNou
    }
}
```

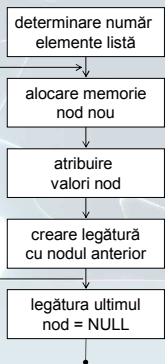


Liste simplu înlănțuite de date (continuare)

#2 Populare valori listă (continuare)

```
void PopulareListaGoala(struct NOD *prim)
{
    //variabile
    int n;
    struct NOD *nodNou, *nodCurent;

    strategie
    {
        nod 1      nod 2      nod nou      ...
        adresă 2   adresă 3   NULL
        adresă 1   adresă 2   adresă 3
        prim      nodCurent nodCurent nodNou
    }
}
```

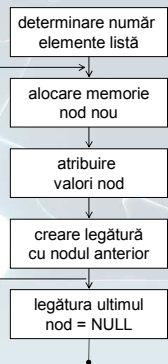


Liste simplu înlănțuite de date (continuare)

#2 Populare valori listă (continuare)

```
void PopulareListaGoala(struct NOD *prim)
{
    int n;
    struct NOD *nodNou, *nodCurent;

    //verificare daca lista este goala
    if (prim->urmatorulNOD!=NULL)
    {
        printf("\nEroare: lista nu este vida!");
        return;
    }
}
```

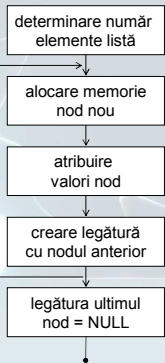


Liste simplu înlănțuite de date (continuare)

#2 Populare valori listă (continuare)

```
void PopulareListaGoala(struct NOD *prim)
{
    int n;
    struct NOD *nodNou, *nodCurent;
    ...

    //citire numar de elemente dorite
    printf("Numar elemente dorite:");
    scanf("%d",&n);
}
```

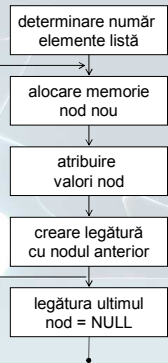


Liste simplu înlănțuite de date (continuare)

#2 Populare valori listă (continuare)

```
void PopulareListaGoala(struct NOD *prim)
{
    int n;
    struct NOD *nodNou, *nodCurent;
    ...

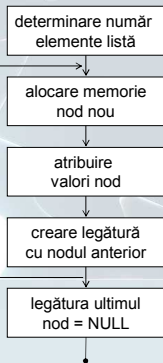
    //parcure elemente
    nodCurent=prim;
    for(int i=0;i<n;i++)
    {
    }
}
```



Liste simplu înlănțuite de date (continuare)

#2 Populare valori listă (continuare)

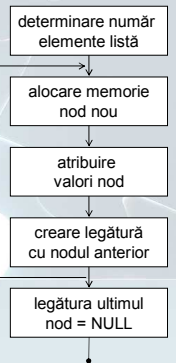
```
void PopulareListaGoala(struct NOD *prim)
{
    int n;
    struct NOD *nodNou, *nodCurent;
    ...
    nodCurent=prim;
    for(int i=0;i<n;i++)
    {
        //alocare memorie nod nou
        nodNou=(struct NOD *)
            malloc(sizeof(struct NOD));
    }
}
```



Liste simplu înlănțuite de date (continuare)

#2 Populare valori listă (continuare)

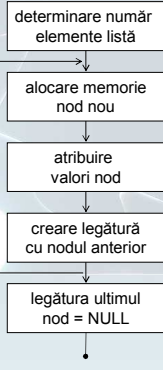
```
void PopulareListaGoala(struct NOD *prim)
{
    int n;
    struct NOD *nodNou, *nodCurent;
    ...
    nodCurent=prim;
    for(int i=0;i<n;i++)
    {
        nodNou=(struct NOD *)
            malloc(sizeof(struct NOD));
        //citire valoare x
        scanf("%d",&nodNou->x);
    }
}
```



Liste simplu înlănțuite de date (continuare)

#2 Populare valori listă (continuare)

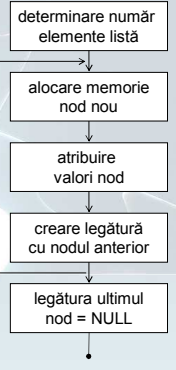
```
void PopulareListaGoala(struct NOD *prim)
{
    int n;
    struct NOD *nodNou, *nodCurent;
    ...
    nodCurent=prim;
    for(int i=0;i<n;i++)
    {
        nodNou=(struct NOD *)
            malloc(sizeof(struct NOD));
        scanf("%d",&nodNou->x);
        //inserare nod si pozitionare pe acesta
        nodCurent->urmatorulNOD=nodNou;
        nodCurent=nodCurent->urmatorulNOD;
    }
}
```



Liste simplu înlănțuite de date (continuare)

#2 Populare valori listă (continuare)

```
void PopulareListaGoala(struct NOD *prim)
{
    int n;
    struct NOD *nodNou, *nodCurent;
    ...
    nodCurent=prim;
    for(int i=0;i<n;i++)
    {
        nodNou=(struct NOD *)
            malloc(sizeof(struct NOD));
        scanf("%d",&nodNou->x);
        nodCurent->urmatorulNOD=nodNou;
        nodCurent=nodCurent->urmatorulNOD;
    }
    nodCurent->urmatorulNOD=NULL;
}
```

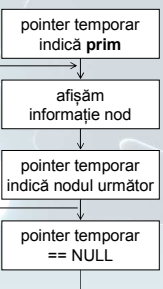


Liste simplu înlănțuite de date (continuare)

#3 Parcurgere listă

```
void ParcurgereLista(struct NOD *prim)
{
    struct NOD *tmp; int i=0;
    printf("\n#Parcurgere lista#\n");
    tmp=prim;
    if (prim==NULL)
    {
        printf("Atentie lista este goala.\n");
        return;
    }

    while (tmp!=NULL)
    {
        printf("nod %d: %d\n",++i,tmp->x);
        tmp=tmp->urmatorulNOD;
    }
}
```



Liste simplu înlănțuite de date (continuare)

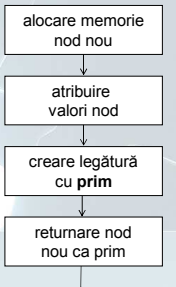
#4 Adăugare nod nou la începutul listei

alocăm memorie pentru un nod nou pe care îl vom adăuga;

se stochează informațiile dorite în noul nod creat;

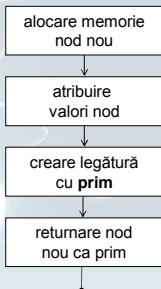
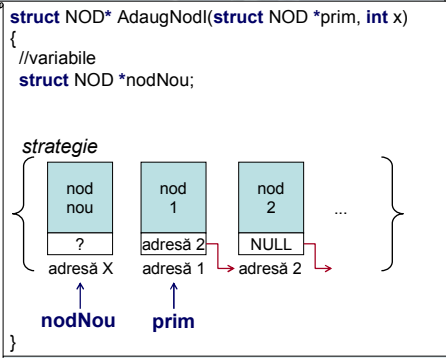
nodul nou indică către prim;

prim devine nodul nou;



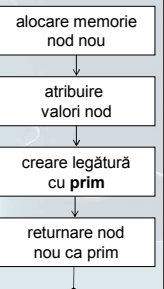
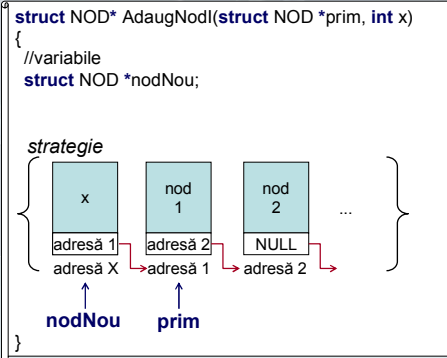
Liste simplu înlănțuite de date (continuare)

#4 Adăugare nod nou la începutul listei (continuare)



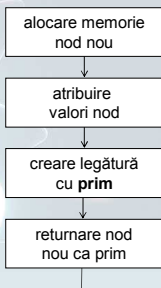
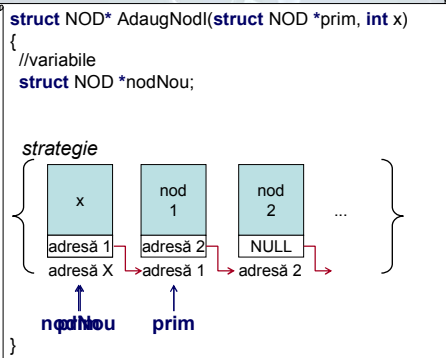
Liste simplu înlănțuite de date (continuare)

#4 Adăugare nod nou la începutul listei (continuare)



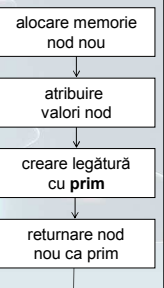
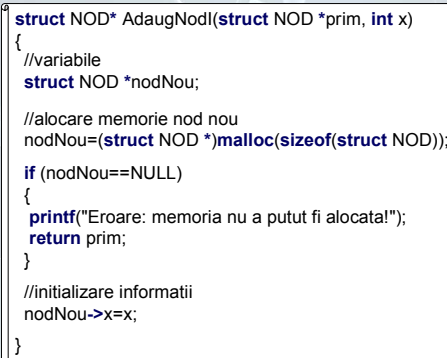
Liste simplu înlănțuite de date (continuare)

#4 Adăugare nod nou la începutul listei (continuare)



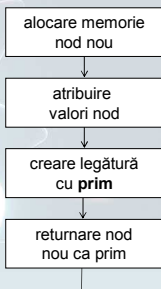
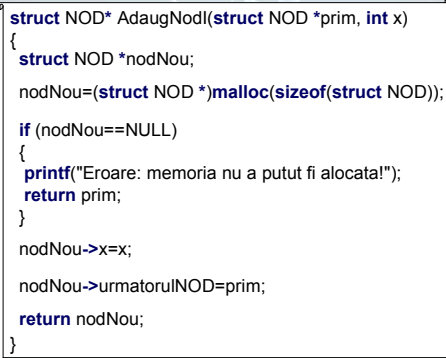
Liste simplu înlănțuite de date (continuare)

#4 Adăugare nod nou la începutul listei (continuare)



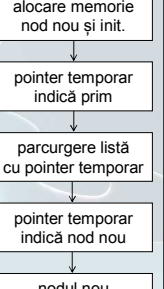
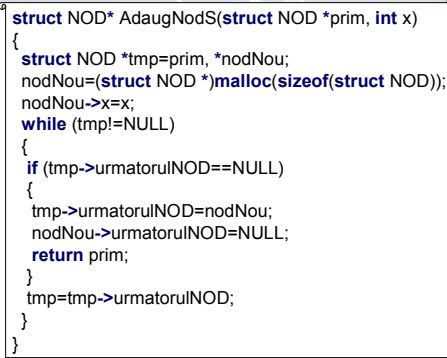
Liste simplu înlănțuite de date (continuare)

#4 Adăugare nod nou la începutul listei (continuare)



Liste simplu înlănțuite de date (continuare)

#5 Adăugare nod nou la finalul listei



Liste simplu înlănțuite de date (continuare)

#6 Adăugare nod nou pe o poziție în listă

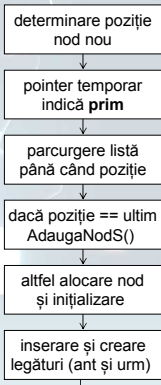
cunoaștem poziția pe care vom insera nodul (ex. după nodul n);

parcurgem lista cu un nod temporar până pe poziție;

dacă poziția dorită este ultimul nod apelăm funcția anterioară;

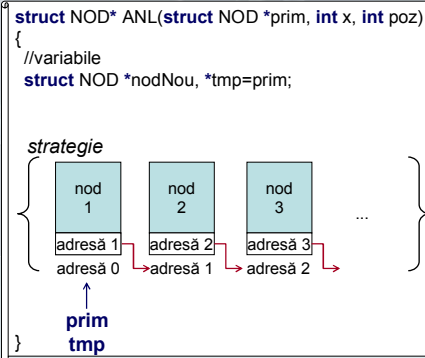
altfel, alocăm nodul nou și inițializăm valorile acestuia;

îl inserăm în listă creând legăturile cu nodul dinainte și cu următorul;



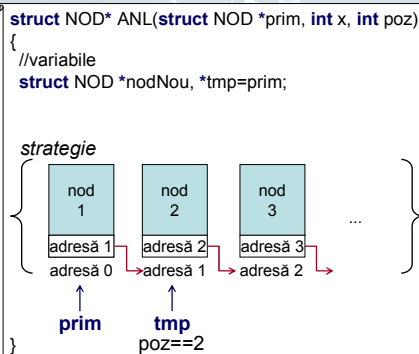
Liste simplu înlănțuite de date (continuare)

#6 Adăugare nod nou în listă (continuare)



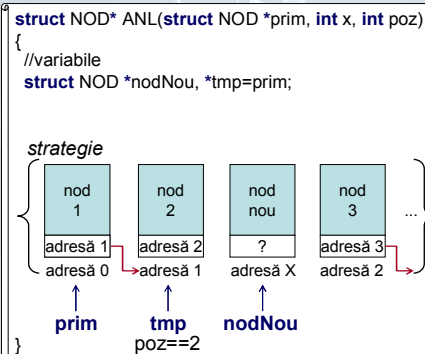
Liste simplu înlănțuite de date (continuare)

#6 Adăugare nod nou în listă (continuare)



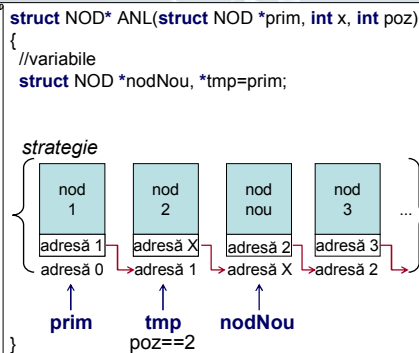
Liste simplu înlănțuite de date (continuare)

#6 Adăugare nod nou în listă (continuare)



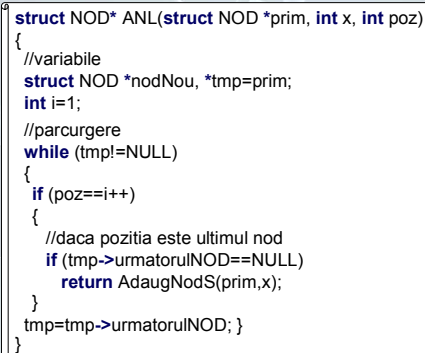
Liste simplu înlănțuite de date (continuare)

#6 Adăugare nod nou în listă (continuare)



Liste simplu înlănțuite de date (continuare)

#6 Adăugare nod nou în listă (continuare)



Liste simplu înlănțuite de date (continuare)

#6 Adăugare nod nou în listă (continuare)

```

struct NOD* ANL(struct NOD *prim, int x, int poz)
{
    ...
    while (tmp!=NULL)
    {
        if (poz==i++)
        { ...
        }
        else
        {
            //alocare memorie
            nodNou=(struct NOD *)malloc(sizeof(struct NOD));
            //scriere informatii
            nodNou->x=x;
        }
        tmp=tmp->urmatorulNOD;
    }
}

```



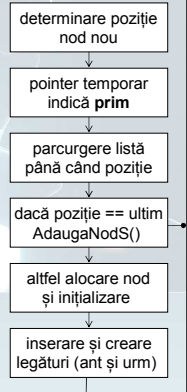
Liste simplu înlănțuite de date (continuare)

#6 Adăugare nod nou în listă (continuare)

```

struct NOD* ANL(struct NOD *prim, int x, int poz)
{
    ...
    while (tmp!=NULL)
    {
        if (poz==i++)
        { ...
        }
        else
        {
            ...
            //legatura nod urmator
            nodNou->urmatorulNOD=
            tmp->urmatorulNOD->urmatorulNOD;
        }
        tmp=tmp->urmatorulNOD;
    }
}

```



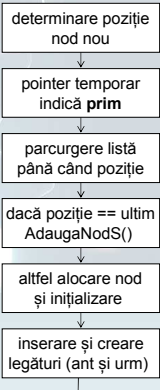
Liste simplu înlănțuite de date (continuare)

#6 Adăugare nod nou în listă (continuare)

```

struct NOD* ANL(struct NOD *prim, int x, int poz)
{
    ...
    while (tmp!=NULL)
    {
        if (poz==i++)
        { ...
        }
        else
        {
            ...
            //legatura nod anterior
            tmp->urmatorulNOD=nodNou;
            return prim;
        }
    }
    tmp=tmp->urmatorulNOD;
}

```



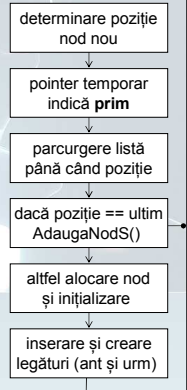
Liste simplu înlănțuite de date (continuare)

#6 Adăugare nod nou în listă (continuare)

```

struct NOD* ANL(struct NOD *prim, int x, int poz)
{
    ...
    while (tmp!=NULL)
    {
        if (poz==i++)
        { ...
        }
        else
        {
            ...
            return prim;
        }
    }
    tmp=tmp->urmatorulNOD;
    return prim;
}

```



Liste simplu înlănțuite de date (continuare)

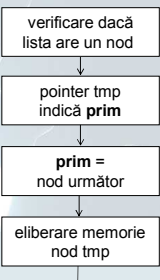
#7 Stergere nod de la începutul listei

dacă lista are un singur nod atunci ștergerea implică ștergerea listei;

pentru a putea șterge un nod avem nevoie de un pointer la acesta;

nodul nou prim este nodul următor;

se eliberează memoria fostului nod prim indicat acum de tmp;



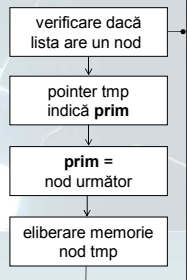
Liste simplu înlănțuite de date (continuare)

#7 Stergere nod de la începutul listei (continuare)

```

struct NOD* StergePrimulNod(struct NOD *prim)
{
    //variabile
    struct NOD *nodSters;
    //lista are un singur nod?
    if (prim->urmatorulNOD==NULL)
    {
        free(prim);
        printf("lista a fost stearsa.\n");
        return NULL;
    }
}

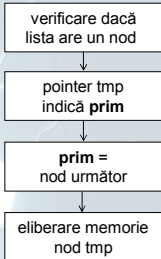
```



Liste simplu înlănțuite de date (continuare)

#7 Stergere nod de la începutul listei (continuare)

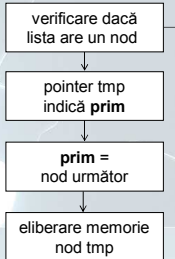
```
struct NOD* StergePrimulNod(struct NOD *prim)
{
    //variabile
    struct NOD *nodSters;
    //lista are un singur nod?
    if (prim->urmatorulNOD==NULL)
    {
        ...
    }
    else
    {
        //indicam nod care va fi sters
        nodSters=prim;
    }
}
```



Liste simplu înlănțuite de date (continuare)

#7 Stergere nod de la începutul listei (continuare)

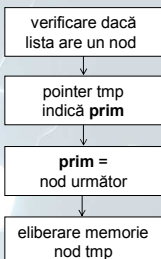
```
struct NOD* StergePrimulNod(struct NOD *prim)
{
    //variabile
    struct NOD *nodSters;
    //lista are un singur nod?
    if (prim->urmatorulNOD==NULL)
    {
        ...
    }
    else
    {
        nodSters=prim;
        //primul nod indica catre urmatorul
        prim=prim->urmatorulNOD;
    }
}
```



Liste simplu înlănțuite de date (continuare)

#7 Stergere nod de la începutul listei (continuare)

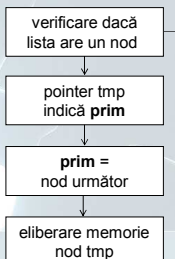
```
struct NOD* StergePrimulNod(struct NOD *prim)
{
    //variabile
    struct NOD *nodSters;
    //lista are un singur nod?
    if (prim->urmatorulNOD==NULL)
    {
        ...
    }
    else
    {
        nodSters=prim;
        prim=prim->urmatorulNOD;
        free(nodSters);
    }
}
```



Liste simplu înlănțuite de date (continuare)

#7 Stergere nod de la începutul listei (continuare)

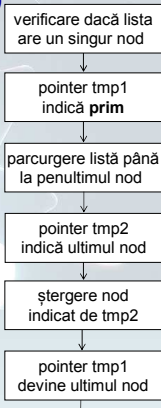
```
struct NOD* StergePrimulNod(struct NOD *prim)
{
    //variabile
    struct NOD *nodSters;
    //lista are un singur nod?
    if (prim->urmatorulNOD==NULL)
    {
        ...
    }
    else
    {
        nodSters=prim;
        prim=prim->urmatorulNOD;
        free(nodSters);
        return prim;
    }
}
```



Liste simplu înlănțuite de date (continuare)

#8 Stergere nod de la sfârșitul listei

```
struct NOD* StergeUltimulNod(struct NOD *prim)
{
    struct NOD *tmp=prim, *nodSters;
    if (prim->urmatorulNOD==NULL)
    { free(prim); return NULL; }
    while (tmp->urmatorulNOD!=NULL)
    {
        if (tmp->urmatorulNOD->urmatorulNOD==NULL)
        {
            nodSters=tmp->urmatorulNOD;
            tmp->urmatorulNOD=NULL;
            free(nodSters); return prim;
        }
        tmp=tmp->urmatorulNOD;
    }
}
```



Liste simplu înlănțuite de date (continuare)

#9 Ștergere nod de pe o poziție din listă

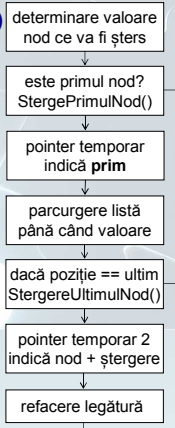
dacă nodul ce trebuie șters este primul, atunci apelăm funcție;

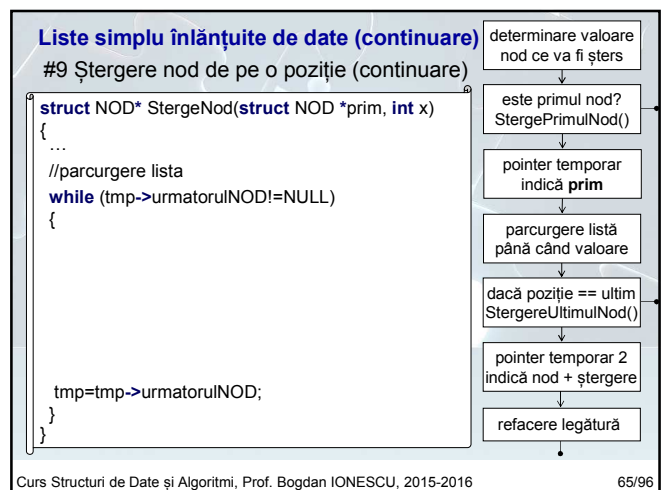
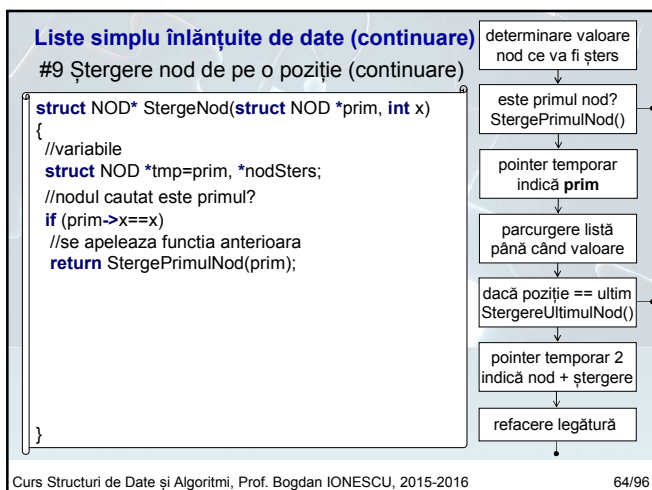
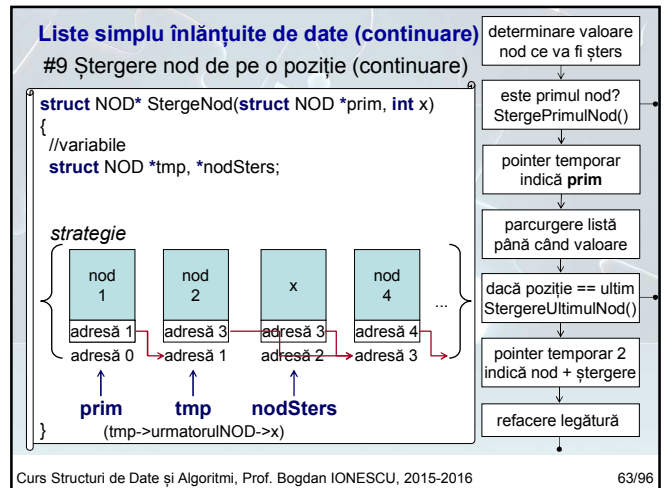
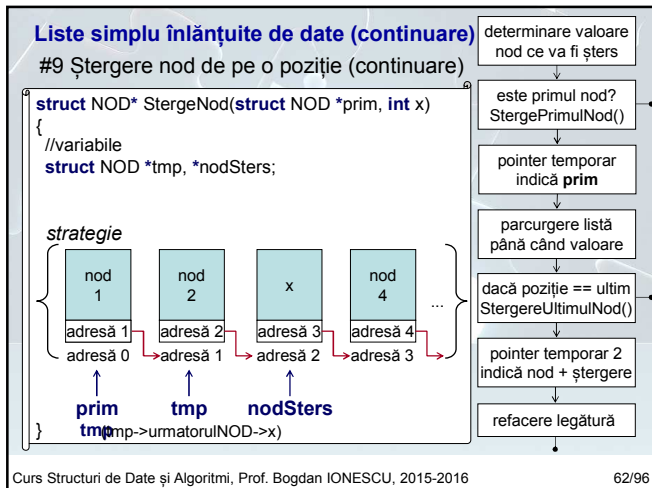
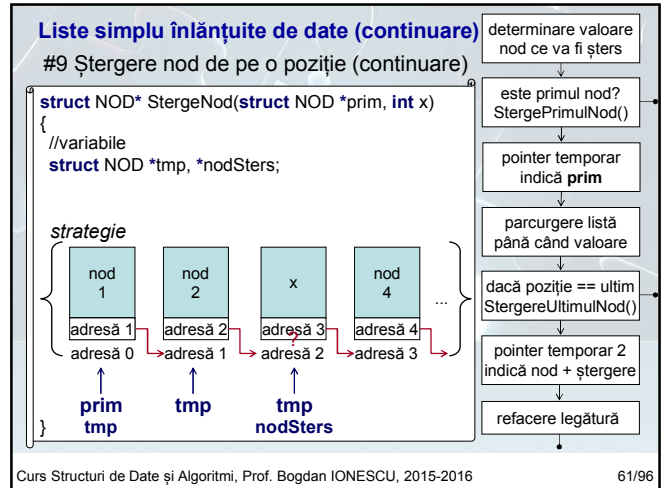
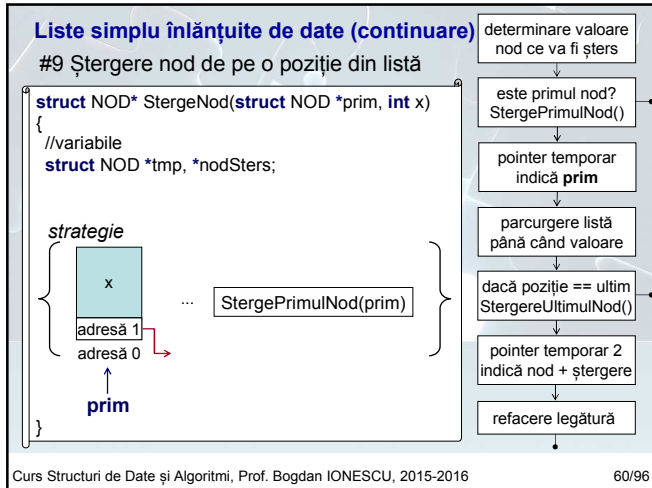
parcursim lista cu un nod temporar până găsim valoare;

dacă poziția dorită este ultimul nod apelăm funcție;

altfel, preluăm nod de șters într-un pointer temporar și ștergem;

refacem legătura între noduri;





Liste simplu înlănțuite de date (continuare)
#9 Ștergere nod de pe o poziție (continuare)

```

struct NOD* StergeNod(struct NOD *prim, int x)
{
    ...
    while (tmp->urmatorulNOD!=NULL)
    {
        //localizare nod cautat
        if (tmp->urmatorulNOD->x==x)
        {
            ...
        }
        tmp=tmp->urmatorulNOD;
    }
}

```

determinare valoare nod ce va fi șters
 ↓
 este primul nod?
 StergePrimulNod()
 ↓
 pointer temporar indică **prim**
 ↓
 parcurgere listă până când valoare
 ↓
 dacă poziție == ultim
 StergereUltimulNod()
 ↓
 pointer temporar 2 indică nod + ștergere
 ↓
 refacere legătură

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016 66/96

Liste simplu înlănțuite de date (continuare)
#9 Ștergere nod de pe o poziție (continuare)

```

struct NOD* StergeNod(struct NOD *prim, int x)
{
    ...
    while (tmp->urmatorulNOD!=NULL)
    {
        if (tmp->urmatorulNOD->x==x)
        {
            //nodul cautat este ultimul?
            if (tmp->urmatorulNOD->urmatorulNOD==NULL)
                return StergereUltimulNodLista(prim);
        }
        tmp=tmp->urmatorulNOD;
    }
}

```

determinare valoare nod ce va fi șters
 ↓
 este primul nod?
 StergePrimulNod()
 ↓
 pointer temporar indică **prim**
 ↓
 parcurgere listă până când valoare
 ↓
 dacă poziție == ultim
 StergereUltimulNod()
 ↓
 pointer temporar 2 indică nod + ștergere
 ↓
 refacere legătură

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016 67/96

Liste simplu înlănțuite de date (continuare)
#9 Ștergere nod de pe o poziție (continuare)

```

struct NOD* StergeNod(struct NOD *prim, int x)
{
    ...
    while (tmp->urmatorulNOD!=NULL)
    {
        if (tmp->urmatorulNOD->x==x)
        {
            else
            {
                nodSters=tmp->urmatorulNOD;
                tmp->urmatorulNOD=
                    tmp->urmatorulNOD->urmatorulNOD;
                free(nodSters); return prim;
            }
        }
        tmp=tmp->urmatorulNOD;
    }
}

```

determinare valoare nod ce va fi șters
 ↓
 este primul nod?
 StergePrimulNod()
 ↓
 pointer temporar indică **prim**
 ↓
 parcurgere listă până când valoare
 ↓
 dacă poziție == ultim
 StergereUltimulNod()
 ↓
 pointer temporar 2 indică nod + ștergere
 ↓
 refacere legătură

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016 68/96

Liste simplu înlănțuite de date (continuare)
#9 Ștergere nod de pe o poziție (continuare)

```

struct NOD* StergeNod(struct NOD *prim, int x)
{
    ...
    while (tmp->urmatorulNOD!=NULL)
    {
        if (tmp->urmatorulNOD->x==x)
        {
            else
            {
                nodSters=tmp->urmatorulNOD;
                tmp->urmatorulNOD=
                    tmp->urmatorulNOD->urmatorulNOD;
                free(nodSters); return prim;
            }
        }
        tmp=tmp->urmatorulNOD;
    }
    return prim;
}

```

determinare valoare nod ce va fi șters
 ↓
 este primul nod?
 StergePrimulNod()
 ↓
 pointer temporar indică **prim**
 ↓
 parcurgere listă până când valoare
 ↓
 dacă poziție == ultim
 StergereUltimulNod()
 ↓
 pointer temporar 2 indică nod + ștergere
 ↓
 refacere legătură

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016 69/96

Liste simplu înlănțuite de date (continuare)
#10 Ștergere listă

```

struct NOD* StergeLista(struct NOD *prim)
{
    struct NOD *tmp=prim;
    while (tmp!=NULL)
    {
        //nodul curent devine urmatorul
        tmp=tmp->urmatorulNOD;
        //se elibereaza memorie primul nod
        free(prim);
        //primul nod indica catre nodul curent
        prim=tmp;
    }
    return NULL;
}

```

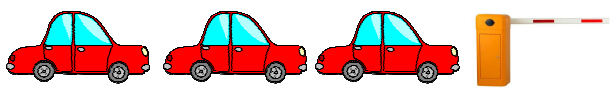
pointer temporar indică **prim**
 ↓
 pointer temporar == NULL?
 ↓
 pointer temporar indică nodul următor
 ↓
 ștergere nod **prim**
 ↓
 pointer **prim** indică nodul temporar

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016 70/96

3.2. Lucrul cu cozi

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016 71/96

P Enunț: să se realizeze un program ce permite gestionarea traficului auto la o barieră de control al accesului într-o instituție.

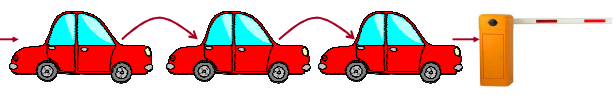


> să analizăm bine problema ...

(1) cum stocăm informația?

- > informația de bază sunt datele mașinii (ex. număr de înmatriculare, imagine, data-ora, etc)
- > acestea se repetă pentru mașini diferite;

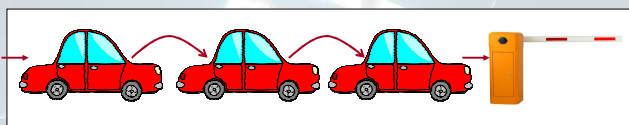
P Enunț: să se realizeze un program ce permite gestionarea traficului auto la o barieră de control al accesului într-o instituție (continuare).



(2) cum este reprezentată informația?

- > mașinile sunt înlănțuite temporal;
- > exista o **ultimă** mașină (fără predecesor) și o **primă** mașină (fără succesor);

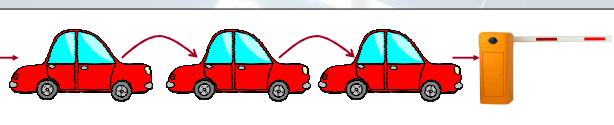
P Enunț: să se realizeze un program ce permite gestionarea traficului auto la o barieră de control al accesului într-o instituție.



(3) ce operații trebuie să putem efectua?

- > introducerea unei mașini (se realizează întotdeauna la **sfârșitul** cozii);
- > "accesul" unei mașini (se realizează "întotdeauna" la **începutul** cozii);

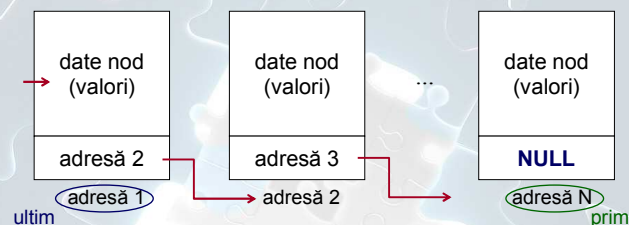
P Enunț: să se realizeze un program ce permite gestionarea traficului auto la o barieră de control al accesului într-o instituție (continuare).



> ce soluții avem pentru a putea implementa o astfel de structură în limbajul C?

cozi ("queue"): un caz particular de listă înlănțuită în care operațiile sunt gestionate pe principiul "primul sosit – primul servit" (**First-In-First-Out - FIFO**);

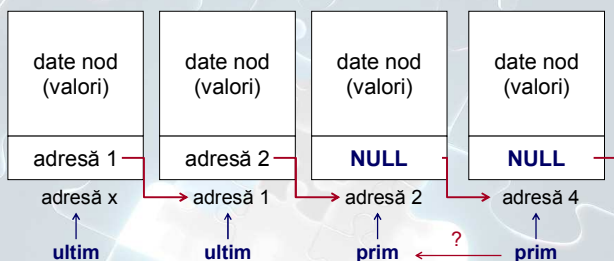
Cozi de date



> pentru a putea implementa o astfel de structură, trebuie să avem acces la:

- adresa **ultimului** nod pentru a insera elemente (*ultim*);
- adresa **primului** nod pentru a scoate elemente (*prim*);

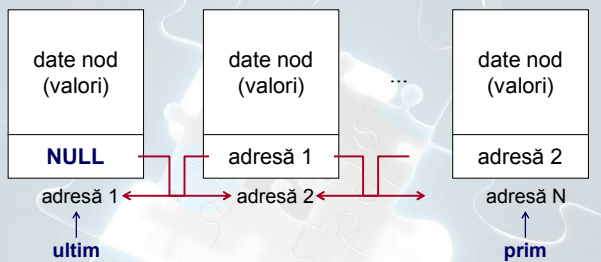
Cozi de date (continuare)



*sunt corect înlănțuite nodurile în reprezentarea de mai sus?

- cum inserăm un nod nou în listă?
- cum extragem un nod din listă?

Cozi de date (continuare)



> în acest fel:

- se păstrează convenția de la liste, legăturile pornesc de la primul element spre ultimul;
- putem realiza ușor cele două operații (inserare/extragere).

Cozi de date (continuare)

Exemplu definire coadă:

```
struct NOD
{
    int x;
    struct NOD *urmatorulNOD;
};

struct NOD *prim, *ultim;
```



> **Efect:** în memorie se alocă o locație pentru pointer-ul *prim* și o locație pentru *ultim*. Acestea vor stoca o adresă la o structură de tip NOD;

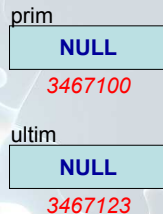
> pointerii trebuie inițializați.

Cozi de date (continuare)

Exemplu definire coadă (continuare):

```
struct NOD
{
    int x;
    struct NOD *urmatorulNOD;
};

struct NOD *prim=NULL, *ultim=NULL;
```



> **Efect:** în memorie se alocă o locație pentru pointer-ul *prim* și o locație pentru *ultim*. Acestea vor stoca o adresă la o structură de tip NOD;

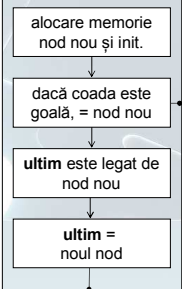
> pointerii trebuie inițializați.

Cozi de date (continuare)

#1 Adăugare nod

```
void AdaugareNod(int x)
{
    struct NOD *nodNou;
    nodNou=(struct NOD*)malloc(sizeof(struct NOD));
    nodNou->x=x;
    nodNou->urmatorulNOD=NULL;

    if (prim==NULL)
    {
        prim=nodNou; ultim=prim;
    }
    else
    {
        ultim->urmatorulNOD=nodNou; ultim=nodNou;
    }
}
```



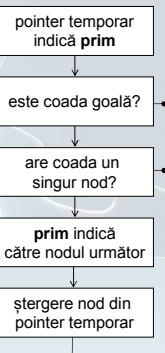
Cozi de date (continuare)

#2 Extragere nod

```
void ExtragereNod()
{
    struct NOD *tmp=prim;
    if (tmp==NULL)
        return;

    if (prim==ultim)
    {
        prim=NULL;
        ultim=NULL;
    }
    else
        prim=prim->urmatorulNOD;

    free(tmp);
}
```



Cozi de date (continuare)

#3 Parcurgere coadă

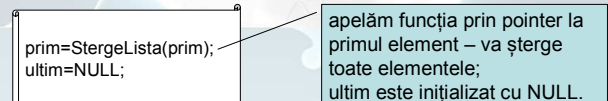
> având în vedere că este un caz particular de listă, folosim funcția definită în cazul listelor (slide 34):

void ParcurgereLista(struct NOD *prim)

#4 Ștergere coadă

> folosim funcția definită în cazul listelor (slide 69):

struct NOD* StergeLista(struct NOD *prim)



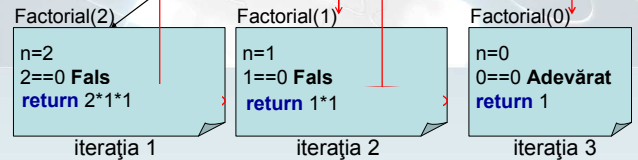
3.3. Lucrul cu stive



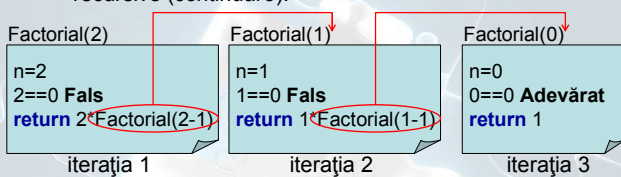
Enunț: Să se realizeze un program ce permite gestionarea memoriei pentru apelarea unei funcții recursive.

```
int main()
{
    printf("%d", 2*1*1);
}
```

```
int Factorial(int n)
{
    if (n==0)
        return 1;
    else
        return n*Factorial(n-1);
}
```



Enunț: Să se realizeze un program ce permite gestionarea memoriei pentru apelarea unei funcții recursive (*continuare*).



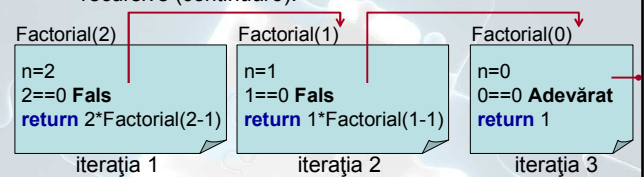
> să analizăm bine problema ...

(1) **cum stocăm informația?**

- > informația de bază sunt datele de lucru ale unei funcții;
- > acestea se repetă pentru instanțe diferite;



Enunț: Să se realizeze un program ce permite gestionarea memoriei pentru apelarea unei funcții recursive (*continuare*).

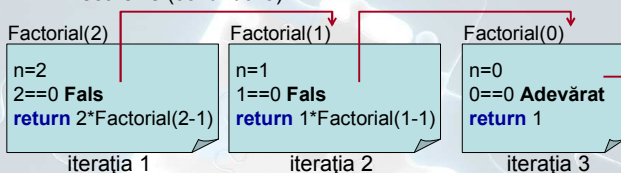


(2) **cum este reprezentată informația?**

- > instanțele funcțiilor sunt înălțuite temporar;
- > exista o **primă** instanță (fără predecesor) și o **ultimă** instanță (fără succesor);



Enunț: Să se realizeze un program ce permite gestionarea memoriei pentru apelarea unei funcții recursive (*continuare*).

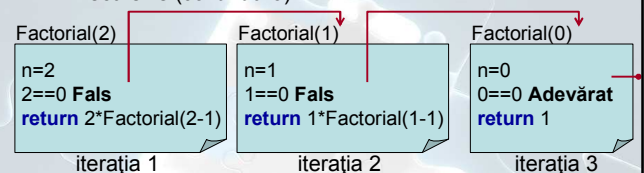


(3) **ce operații trebuie să putem efectua?**

- > adăugarea unei instanțe (se realizează întotdeauna în continuarea **ultimei** instanțe);
- > eliminarea unei instanțe (se realizează întotdeauna prin **ultima** instanță);



Enunț: Să se realizeze un program ce permite gestionarea memoriei pentru apelarea unei funcții recursive (*continuare*).



> ce soluții avem pentru a putea implementa o astfel de structură în limbajul C?

stive ("stack"): un caz particular de listă înălțuită în care operațiile sunt gestionate pe principiul "ultimul sosit – primul servit" (**Last-In-First-Out - LIFO**);

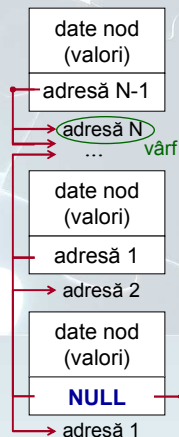
Stive de date

> pentru a putea implementa o astfel de structură, trebuie să avem acces la:

- adresa **ultimului nod** pentru a insera elemente (*vârf*);
- adresa **ultimului nod** pentru a șterge elemente (*vârf*);

> stiva este astfel complet definită doar prin accesul la adresa ultimului nod;

*sunt corect înălțuite nodurile în reprezentarea din dreapta?



Stive de date (continuare)

Exemplu definire stivă:

```
struct NOD
{
    int x;
    struct NOD *urmatorulNOD;
};

struct NOD *varf;
```

varf
#*@5!
3467100

> **Efect:** în memorie se alocă o locație pentru pointer-ul *varf*. Acesta va stoca o adresă la o structură de tip NOD;

> pointerii trebuie inițializați.

Stive de date (continuare)

Exemplu definire stivă (continuare):

```
struct NOD
{
    int x;
    struct NOD *urmatorulNOD;
};

struct NOD *varf=NULL;
```

varf
NULL
3467100

> **Efect:** în memorie se alocă o locație pentru pointer-ul *varf*. Acesta va stoca o adresă la o structură de tip NOD;

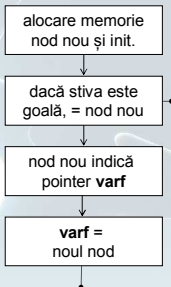
> pointerii trebuie inițializați.

Stive de date (continuare)

#1 Adăugare nod

```
struct NOD* AdaugareNod(struct NOD *varf, int x)
{
    struct NOD *nodNou;
    nodNou=(struct NOD*)malloc(sizeof(struct NOD));
    nodNou->x=x;
    nodNou->urmatorulNOD=NULL;

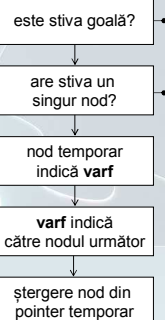
    if (varf==NULL)
        return nodNou;
    else
    {
        nodNou->urmatorulNOD=varf;
        varf=nodNou;
        return varf;
    }
}
```



Stive de date (continuare)

#2 Extragere nod

```
struct NOD* ExtragereNod(struct NOD *varf)
{
    struct NOD *nodSters;
    if (varf==NULL)
        return NULL;
    else if (varf->urmatorulNOD==NULL)
    {
        free(varf); return NULL;
    }
    else
    {
        nodSters=varf;
        varf=varf->urmatorulNOD;
        free(nodSters); return varf;
    }
}
```



Stive de date (continuare)

#3 Parcurgere stivă

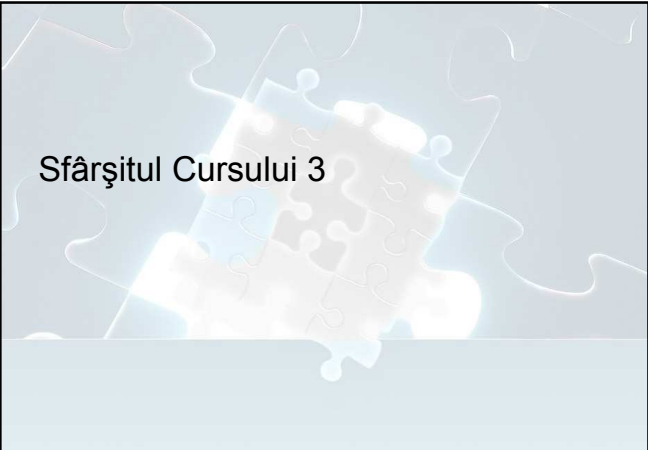
> având în vedere că este un caz particular de listă, folosim funcția definită în cazul listelor (slide 34):

void ParcurgereLista(struct NOD *prim)

#4 Ștergere stivă

> folosim funcția definită în cazul listelor (slide 69):

struct NOD* StergeLista(struct NOD *prim)



Sfârșitul Cursului 3