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

Structuri de Date și Algoritmi (limbajul C)

Curs 4 – Arbori de date

Prof. Bogdan IONESCU

2015-2016

Cuprins

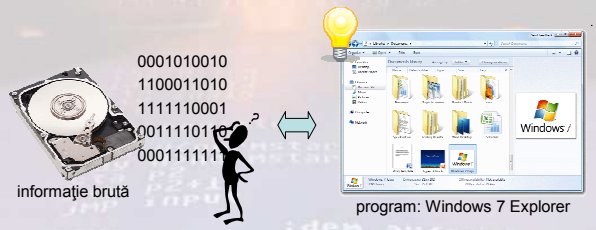
4.1. Introducere

4.2. Lucrul cu arbori binari

4.1. Introducere

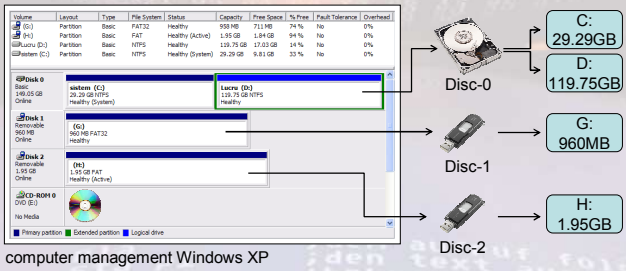
P *Enunţ*: să se realizeze un program ce permite stocarea și manipularea structurii de fișiere a unui sistem de calcul (PC).

> Datele ce sunt păstrate pe termen lung sunt stocate permanent în memoria externă (ex. HDD);



P *Enunţ*: să se realizeze un program ce permite stocarea și manipularea structurii de fișiere a unui sistem de calcul (*continuare*).

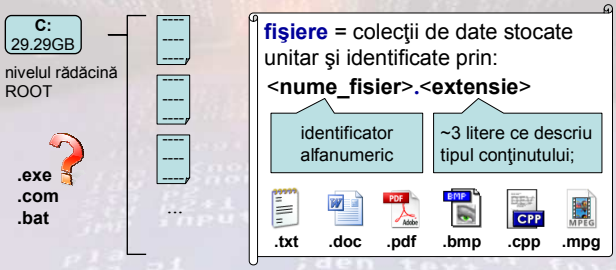
⇒ Pentru a putea accesa informaţia de pe dispozitivele externe de stocare aceasta trebuie indexată sau organizată;



computer management Windows XP

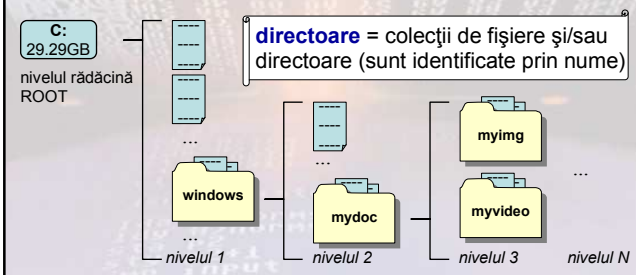
P *Enunţ*: să se realizeze un program ce permite stocarea și manipularea structurii de fișiere a unui sistem de calcul (*continuare*).

⇒ informaţia stocată pe fiecare unitate logică (partiţie) este organizată ierarhic:



P

Enunţ: să se realizeze un program ce permite stocarea și manipularea structurii de fișiere a unui sistem de calcul (*continuare*).



P

Enunţ: să se realizeze un program ce permite stocarea și manipularea structurii de fișiere a unui sistem de calcul (*continuare*).

> să analizăm bine problema ...

(1) **cum stocăm informația**?

> informația de bază este un fișier și informațiile despre acesta:

- text;
- valori;
- binar;
- descriere fișier;

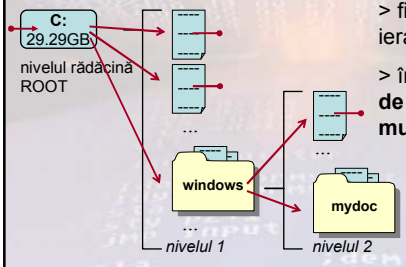
etc.

> acestea se repetă pentru valori diferite ale datelor stocate de utilizatori;

P

Enunţ: să se realizeze un program ce permite stocarea și manipularea structurii de fișiere a unui sistem de calcul (*continuare*).

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



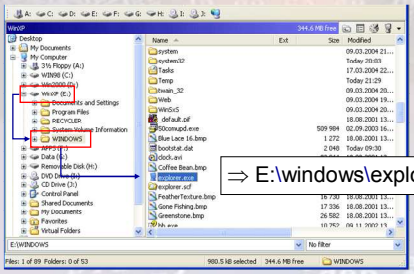
> fișierele sunt înlănțuite ierarhic pe diferite niveluri;

> în ierarhie, există **un nod de plecare** și există **mai multe noduri terminale**.

P

Enunţ: să se realizeze un program ce permite stocarea și manipularea structurii de fișiere a unui sistem de calcul (*continuare*).

(3) **ce operații trebuie să putem efectua**? accesare fișier (parcurgere)



E:

↳ **windows**

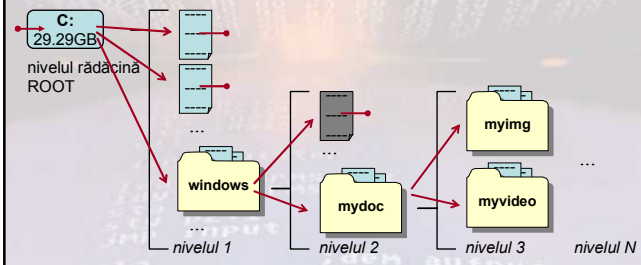
↳ **explorer.exe**

⇒ E:\windows\explorer.exe

P

Enunţ: să se realizeze un program ce permite stocarea și manipularea structurii de fișiere a unui sistem de calcul (*continuare*).

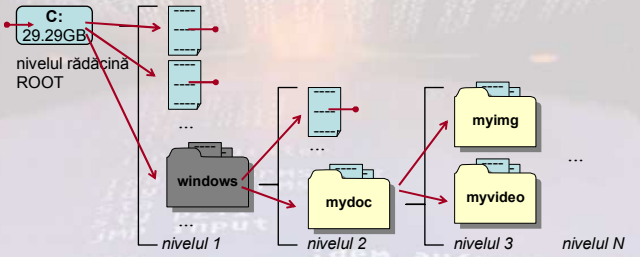
(3) **ce operații trebuie să putem efectua**? stergere fișier



P

Enunţ: să se realizeze un program ce permite stocarea și manipularea structurii de fișiere a unui sistem de calcul (*continuare*).

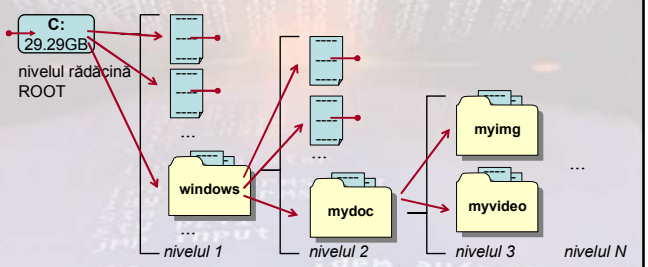
(3) **ce operații trebuie să putem efectua**? stergere director



P

Enunţ: să se realizeze un program ce permite stocarea și manipularea structurii de fișiere a unui sistem de calcul (*continuare*).

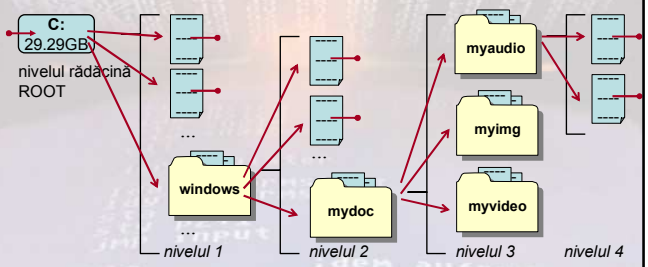
(3) **ce operații trebuie să putem efectua**? inserare fișier



P

Enunţ: să se realizeze un program ce permite stocarea și manipularea structurii de fișiere a unui sistem de calcul (*continuare*).

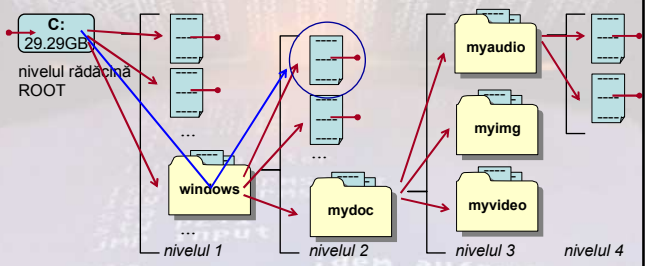
(3) **ce operații trebuie să putem efectua**? inserare director



P

Enunţ: să se realizeze un program ce permite stocarea și manipularea structurii de fișiere a unui sistem de calcul (*continuare*).

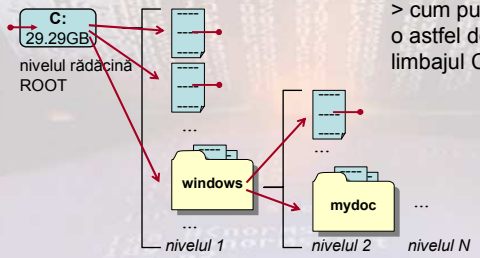
(3) **ce operații trebuie să putem efectua**? căutare fișier



P

Enunţ: să se realizeze un program ce permite stocarea și manipularea structurii de fișiere a unui sistem de calcul (*continuare*).

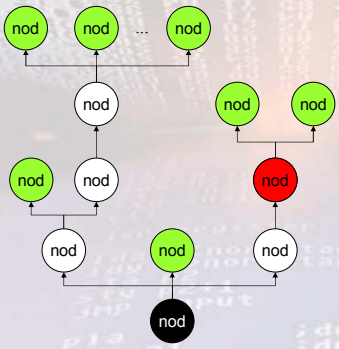
> cum putem implementa o astfel de structură în limbajul C?



arbori (“trees”): un ansamblu de structuri de date, dinamice, recursive, relaționate ierarhic pe diferite niveluri;

Arbori de date

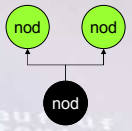
> arbore = o mulțime finită de noduri relaționate ierarhic;



primul nod = **nod rădăcină** (“root”), nu are predecesori;

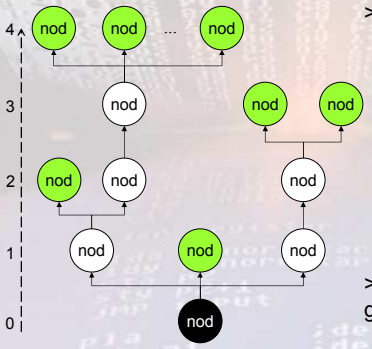
ultimele noduri = **noduri frunză** (“leaf”), nu au succesori;

celelalte noduri = formează **sub-arbori** (definit recursiv);



Arbori de date (continuare)

> arbore = o mulțime finită de noduri relaționate ierarhic;



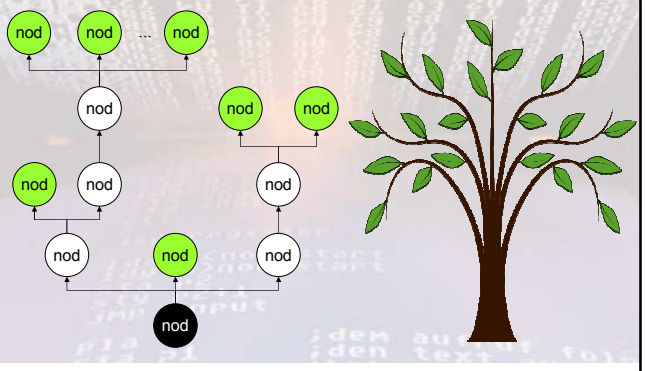
> terminologie:

- un nod rădăcină se numește “*nod tată*”;
- subarborii nodului tată se numesc “descendenți”;
- rădăcinile subarborilor unui nod tată se numesc “fii”.

> ierarhie: nodurile se găsesc pe niveluri ierarhice;

Arbori de date (continuare)

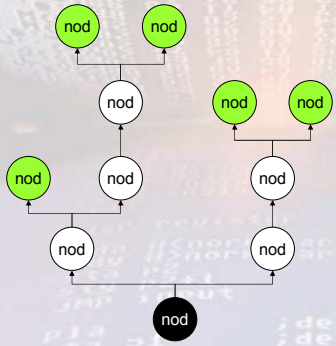
> arbore = o mulțime finită de noduri relaționate ierarhic;



4.2. Lucrul cu arbori binari

Arbori binari

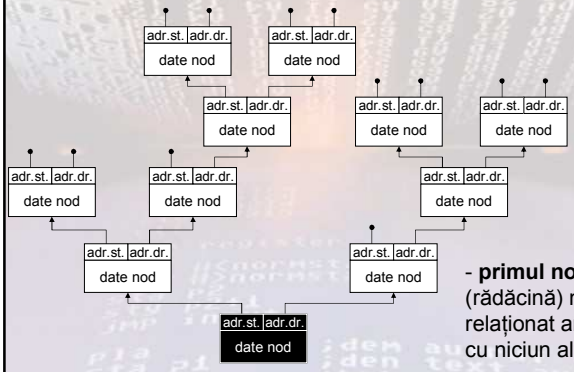
> arbore binar = un caz particular de arbore în care fiecare nod rădăcină are maxim doi descendenți;



- fiecare nod este astfel relaționat cu maxim două noduri: **nod stânga** și **nod dreapta**;

Arbori binari (continuare)

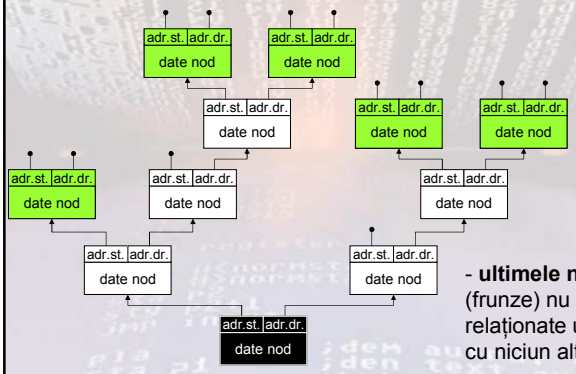
> fiecare nod va stoca două adrese de noduri (stânga/dreapta);



- **primul nod** (rădăcină) nu este relaționat anterior cu niciun alt nod;

Arbori binari (continuare)

> fiecare nod va stoca două adrese de noduri (stânga/dreapta);



- **ultimele noduri** (frunze) nu sunt relaționate următor cu niciun alt nod;

Arbori binari (continuare)

Exemplu definire arbore binar:

```
struct NOD
{
 int x;
 struct NOD *NODstanga;
 struct NOD *NODdreapta;
};

struct NOD *prim;
```

prim

#*@5!

3467100

> ***Efect***: în memorie se alocă o locație pentru pointer-ul *prim*. Acesta va stoca o adresă la o structură de tip NOD și reprezintă nodul rădăcină al arborelui;

> pointerii trebuie inițializați.

Arbori binari (continuare)

Exemplu definire arbore binar:

```
struct NOD
{
 int x;
 struct NOD *NODstanga;
 struct NOD *NODdreapta;
};

struct NOD *prim=NULL;
```

prim

NULL

3467100

> ***Efect***: în memorie se alocă o locație pentru pointer-ul *prim*. Acesta va stoca o adresă la o structură de tip NOD și reprezintă nodul rădăcină al arborelui;

> pointerii trebuie inițializați.

Arbori binari de căutare

> pentru exemplificarea algoritmilor, vom considera mai departe un caz particular de arbori binari, și anume arbori binari de căutare;

> **arbore binar de căutare** = un arbore binar care stochează pe lângă informațiile dorite, o valoare numerică, cheie (*n*), cu rol de ordonare, astfel:

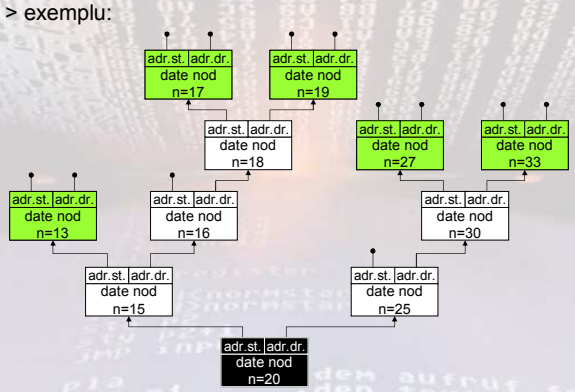
- pentru oricare nod, descendenții acestuia din subarborele din **stânga** au valoarea *n* < decât a nodului;

- pentru oricare nod, descendenții acestuia din subarborele din **dreapta** au valoarea *n* >= cu a nodului;

> o astfel de implementare a unui arbore permite o parcurgere mai rapidă (eficientă) a valorilor (de ce anume?).

Arbori binari de căutare (continuare)

> exemplu:



Arbori binari de căutare (continuare)

Exemplu definire arbore binar de căutare:

```
struct NOD
{
 ...
 int n; // cheie ordonare
 struct NOD *NODstanga;
 struct NOD *NODdreapta;
};

struct NOD *prim=NULL;
```

prim

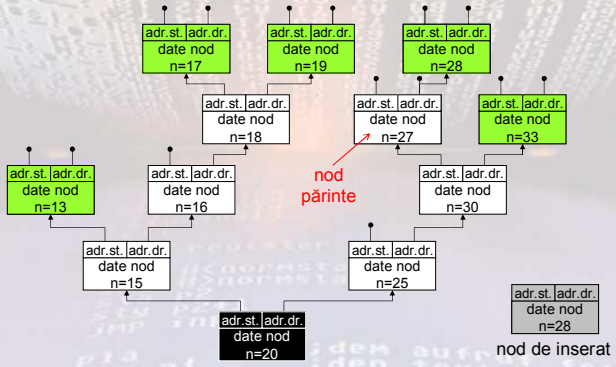
NULL

3467100

> ***Efect***: în memorie se alocă o locație pentru pointer-ul *prim*. Acesta va stoca o adresă la o structură de tip NOD și reprezintă nodul rădăcină al arborelui;

Arbori binari de căutare (continuare)

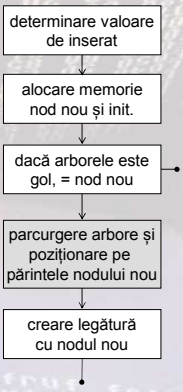
#1 Inserare nod în arbore



Arbori binari de căutare (continuare)

#1 Inserare nod în arbore (continuare)

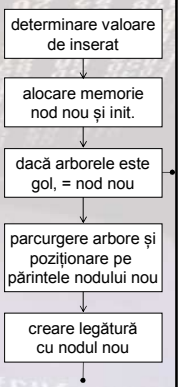
- determinare valoare cheie;
- alocăm memorie pentru nodul nou și inițializăm valorile;
- dacă arborele este gol, atunci nodul nou devine primul nod;
- conform principiului de arbore binar ordonat, trebuie să ne poziționăm pe nodul unde vom lega noul nod;
- creare legătură (stânga sau dreapta) cu noul nod;



Arbori binari de căutare (continuare)

#1 Inserare nod în arbore (continuare)

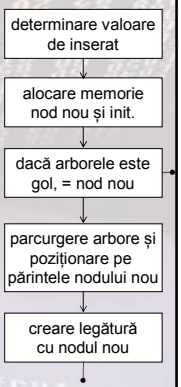
```
struct NOD* InserareNod(struct NOD *prim, int n)
{
 //variabile
 struct NOD *nodNou, *nodCurent, *nodParinte;
 //alocare memorie nod nou
 nodNou=(struct NOD *)malloc(sizeof(struct NOD));
 //initializare valori nod nou
 nodNou->n=n;
 nodNou->NODstanga=NULL;
 nodNou->NODdreapta=NULL;

}
```



Arbori binari de căutare (continuare)

#1 Inserare nod în arbore (continuare)

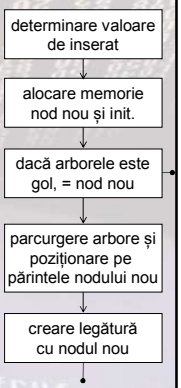
```
struct NOD* InserareNod(struct NOD *prim, int n)
{
 //variabile
 struct NOD *nodNou, *nodCurent, *nodParinte;
 ...
 //este arborele vid?
 if (prim==NULL)
 {
  prim=nodNou;
  return prim;
 }

}
```



Arbori binari de căutare (continuare)

#1 Inserare nod în arbore (continuare)

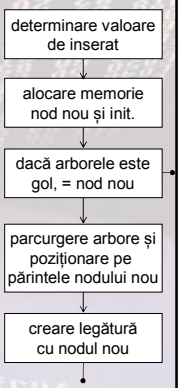
```
struct NOD* InserareNod(struct NOD *prim, int n)
{
 //variabile
 struct NOD *nodNou, *nodCurent, *nodParinte;
 ...
 //pozitionare in arbore pe parintele nodului nou
 nodCurent=prim;
 while (nodCurent!=NULL)
 {
  nodParinte=nodCurent;
  if (n<nodCurent->n)
   nodCurent=nodCurent->NODstanga; //stanga
  else
   nodCurent=nodCurent->NODdreapta; //dreapta
 }
}
```



Arbori binari de căutare (continuare)

#1 Inserare nod în arbore (continuare)

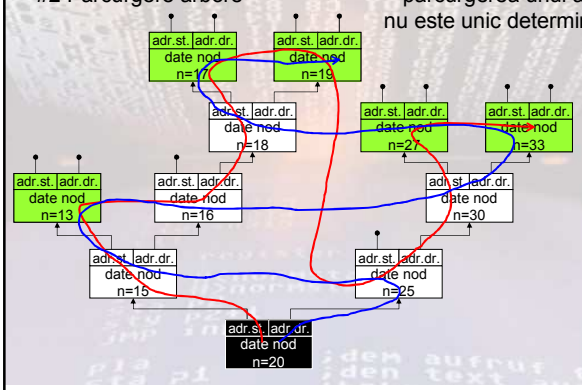
```
struct NOD* InserareNod(struct NOD *prim, int n)
{
 //variabile
 struct NOD *nodNou, *nodCurent, *nodParinte;
 ...
 //creare legatura nod parinte cu nodul nou
 if (n<nodParinte->n)
  //inserare la stanga
  nodParinte->NODstanga=nodNou;
 else
  //inserare la dreapta
  nodParinte->NODdreapta=nodNou;

 return prim;

}
```



Arbori binari de căutare (continuare)

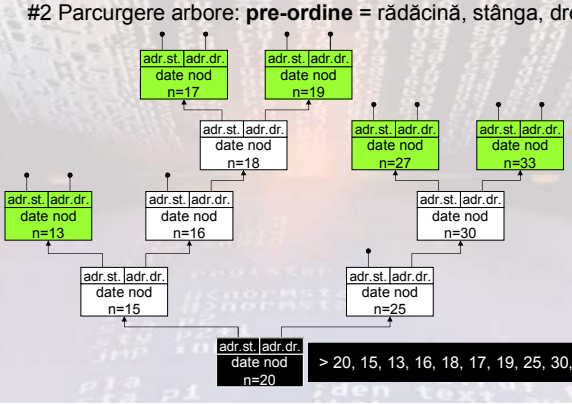
#2 Parcurgere arbore

> parcurgerea unui arbore nu este unic determinată;



Arbori binari de căutare (continuare)

#2 Parcurgere arbore: **pre-ordine** = rădăcină, stânga, dreapta

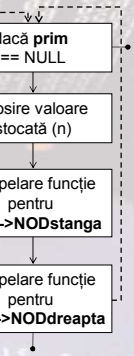


> 20, 15, 13, 16, 18, 17, 19, 25, 30, 27, 33.

Arbori binari de căutare (continuare)

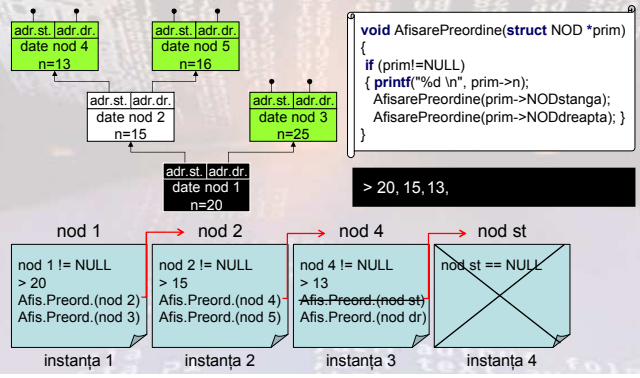
#2 Parcurgere arbore: **pre-ordine** (r-s-d; *continuare*)

```
void AfisarePreordine(struct NOD *prim)
{
 if (prim!=NULL)
 {
  //parcurgere radacina
  printf("%d \n", prim->n);

  //parcurgere sub-arbore stanga
  AfisarePreordine(prim->NODstanga);

  //parcurgere sub-arbore dreapta
  AfisarePreordine(prim->NODdreapta);
 }
}
```



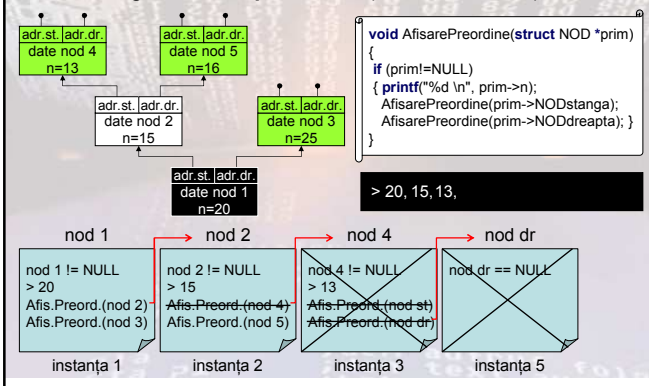
Arbori binari de căutare (continuare)

#2 Parcurgere arbore: **pre-ordine** (r-s-d; *continuare*)



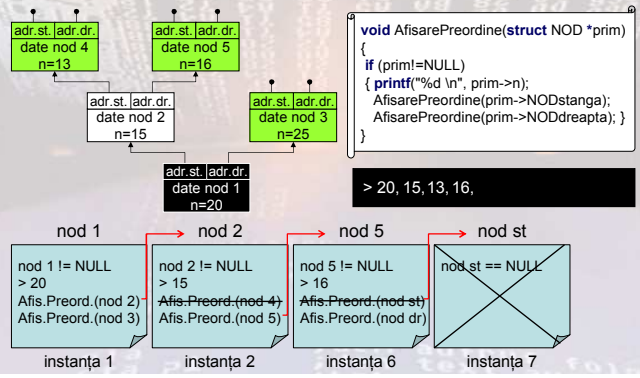
Arbori binari de căutare (continuare)

#2 Parcurgere arbore: **pre-ordine** (r-s-d; *continuare*)



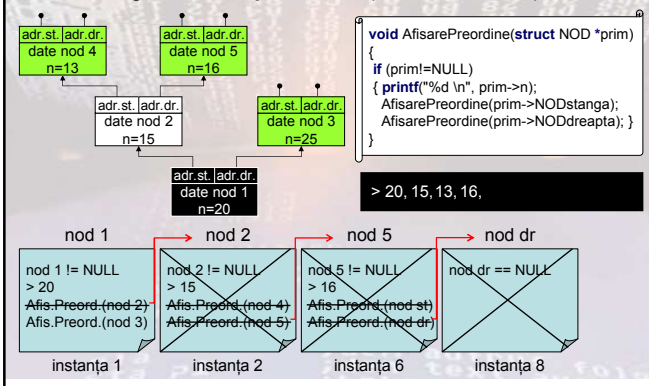
Arbori binari de căutare (continuare)

#2 Parcurgere arbore: **pre-ordine** (r-s-d; *continuare*)



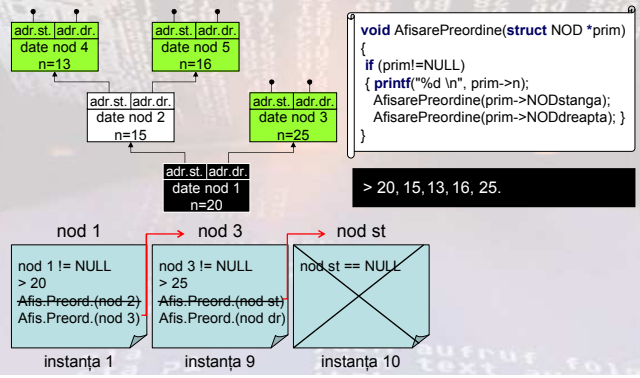
Arbori binari de căutare (continuare)

#2 Parcurgere arbore: **pre-ordine** (r-s-d; *continuare*)



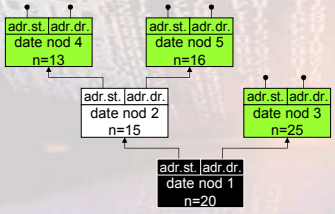
Arbori binari de căutare (continuare)

#2 Parcurgere arbore: **pre-ordine** (r-s-d; *continuare*)



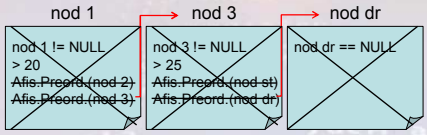
Arbori binari de căutare (continuare)

#2 Parcurgere arbore: **pre-ordine** (r-s-d; *continuare*)



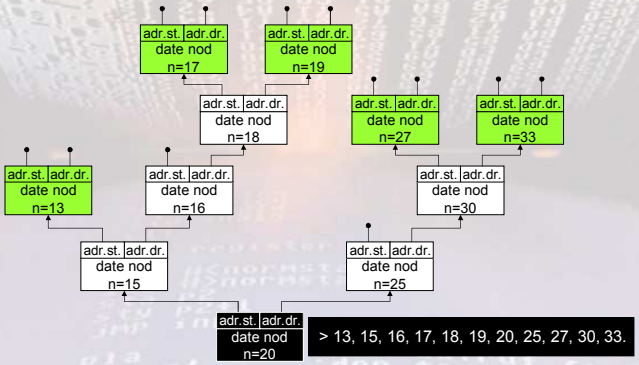

```
void AfisarePreordine(struct NOD *prim)
{
 if (prim!=NULL)
 { printf("%d \n", prim->n);
   AfisarePreordine(prim->NODstanga);
   AfisarePreordine(prim->NODdreapta); }
}
```

> 20, 15, 13, 16, 25.



Arbori binari de căutare (continuare)

#2 Parcurgere arbore: **in-ordine** = stânga, rădăcină, dreapta

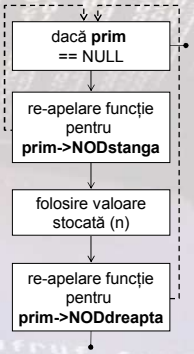


> 13, 15, 16, 17, 18, 19, 20, 25, 27, 30, 33.

Arbori binari de căutare (continuare)

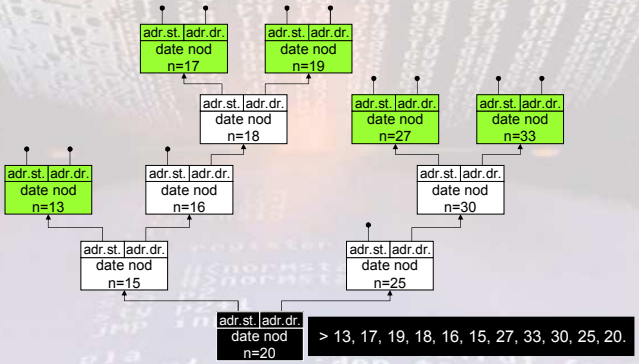
#2 Parcurgere arbore: **in-ordine** (s-r-d; *continuare*)

```
void AfisareInordine(struct NOD *prim)
{
 if (prim!=NULL)
 {
  //parcurgere sub-arbore stanga
  AfisarePreordine(prim->NODstanga);

  //parcurgere radacina
  printf("%d \n", prim->n);

  //parcurgere sub-arbore dreapta
  AfisarePreordine(prim->NODdreapta);
 }
}
```



Arbori binari de căutare (continuare)

#2 Parcurgere arbore: **post-ordine** = stânga, dreapta, rădăcină

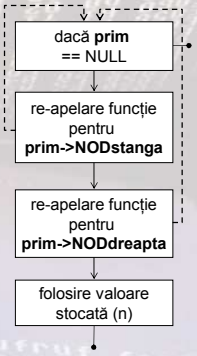


> 13, 17, 19, 18, 16, 15, 27, 33, 30, 25, 20.

Arbori binari de căutare (continuare)

#2 Parcurgere arbore: **post-ordine** (s-d-r; *continuare*)

```
void AfisarePostordine(struct NOD *prim)
{
 if (prim!=NULL)
 {
  //parcurgere sub-arbore stanga
  AfisarePreordine(prim->NODstanga);

  //parcurgere sub-arbore dreapta
  AfisarePreordine(prim->NODdreapta);

  //parcurgere radacina
  printf("%d \n", prim->n);
 }
}
```



Arbori binari de căutare (continuare)

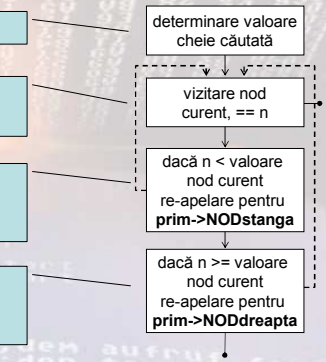
#3 Căutarea unui nod

determinare valoare cheie, *n*;

verificăm dacă nodul curent este cel căutat;

dacă nu, dacă valoarea lui *n* este mai mică, căutăm în sub-arbore stâng;

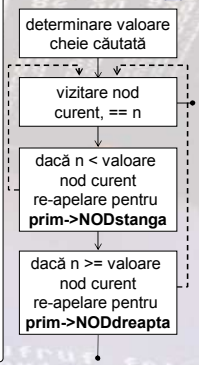
dacă nu, dacă valoarea lui *n* este mai mare sau egală, căutăm în sub-arbore drept;



Arbori binari de căutare (continuare)

#3 Căutarea unui nod (continuare)

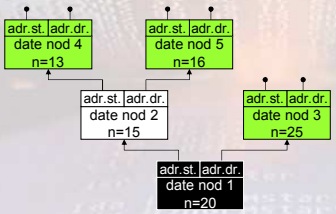
```
struct NOD *CautaNod(struct NOD *tmp, int n)
{
 if (tmp!=NULL)
 {
  if (n==tmp->n)
   return tmp;
  else if (n<tmp->n)
   //cautare sub-arbore stang
   return CautaNod(tmp->NODstanga, n);
  else
   //cautare sub-arbore drept
   return CautaNod(tmp->NODdreapta, n);
 }
 else
  return NULL; //nu a fost gasit
}
```



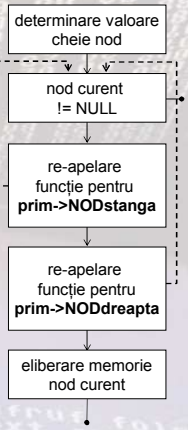
Arbori binari de căutare (continuare)

#4 Ștergere arbore

> există foarte multe posibilități de abordare a acestei probleme (exemple?)



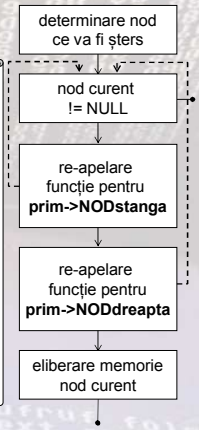
> *propunere*: furnizăm nodul de pornire și ștergem doate nodurile următoare;



Arbori binari de căutare (continuare)

#4 Ștergere arbore (continuare)

```
struct NOD *StergereArbore(struct NOD *tmp)
{
 if (tmp!=NULL)
 {
  //stergere sub-arbore stanga
  StergereArbore(tmp->NODstanga);
  //stergere sub-arbore dreapta
  StergereArbore(tmp->NODdreapta);
  //eliberare memorie nod curent
  free(tmp);
 }
 return NULL;
}
```



Sfârşitul Cursului 4