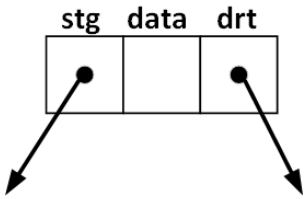
# L09. Arbori binari de căutare
## Binary Search Trees

**Link tutoriale online pentru laboratoare**
corina.is7s.com
User: student
Password: st_aia
Activitate didactica ->SD/SDA

**Centralizator prezență online Google Meet (loguri GSuite)**
https://docs.google.com/spreadsheets/d/1zN39la7NmRpx2srfYRE-N4hVWW4l1xKIZUkeWMi1mWM/edit?usp=sharing

**Structura unui BST**




```
struct Nod{
      int data;
      Nod *stg, *drt;
};
```

**Proprietatea unui BST**
Pentru orice nod din arbore, nodurile localizate în subarborele stâng au valori mai mici decât valoarea din rădăcină și valoarea din rădăcină este mai mică decât toate valorile din subarborele drept.

**Operațiile de bază pe arbori BST**

1. Inițializarea
2. Inserarea
3. Parcurgerea
4. Căutarea
5. Ștergerea
6. Eliberarea memoriei

1. **Inițializarea**

presupune inițializarea rădăcinii arborelui cu NULL

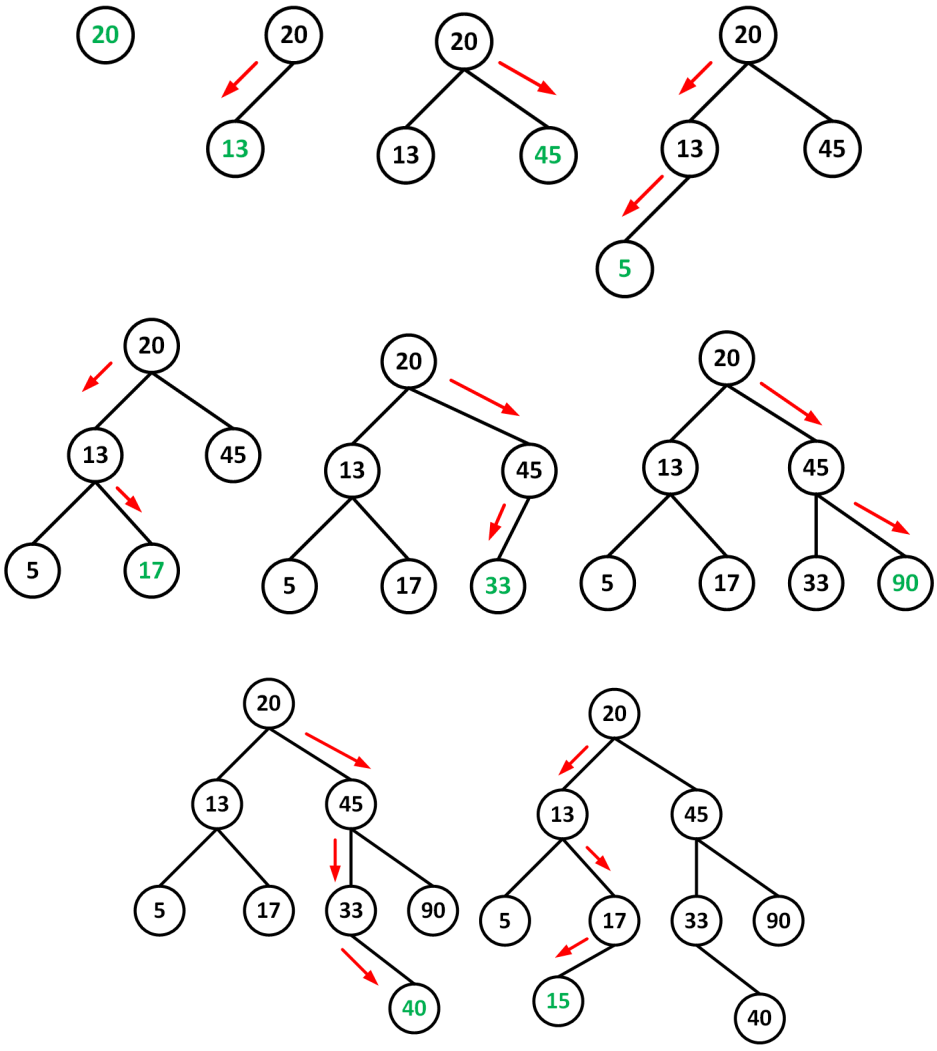
```
void InitTree(Nod* & rad)
{
	rad = 0;
}
```

se va apela în main cu

```
Nod* rad;
InitTree(rad);
```

2. **Inserarea**

Inserez într-un arbore BST valorile: 20, 13, 45, 5, 17, 33, 90 15, 40




Funcția de inserare se va folosi de o funcție pentru crearea unui Nod (în aceasta unui nod i se va aloca memorie și i se vor inițializa câmpurile)

```
Nod* MakeNod(int val)
{
	Nod* p;
	p=new Nod;
	p->data=val;
	p->stg=0;
	p->drt=0;
	return p;
}
```

### 2.1. Inserarea în arbore - varianta recursivă

```
void Insert(Nod* &rad, int val)
{
	//daca rad este null si inserez primul nod din arbore
	//    atunci rad primeste nodul returnat de MakeNod pentru valoarea val
	//    altfel//aici o sa tratez cazul cu rad nenula si trebuie sa
	//                //vad cum este val comparativ cu valoarea pe care o am
	//                //in radacina
	//          daca val este mai mica decat valoarea din radacina rad
	//          atunciapelez Insert-ul pentru subarborele stang
	//          altfel
	//                daca val este egala cu valoarea din radacina rad
	//                atunci      afisez duplicat
	//                altfelapelez Insert-ul pentru subarborele drept
	//                sf.daca
	//          sf.daca
	//sf.daca
}
```

### 2.2. Inserarea în arbore - varianta iterativă

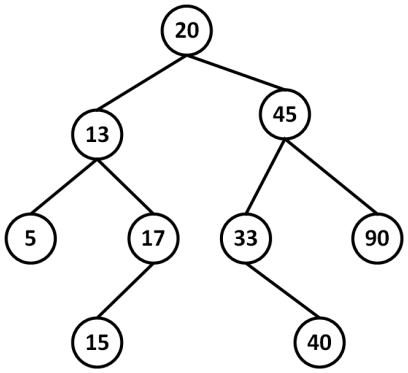
```
void Insert(Nod*& rad, int val)
{
	// daca rad este nula
	// atunci rad primeste un nod nou initializat cu valoarea val
	// altfel   consider un pointer p care primeste rad
	//    cat timp p nu este null executa
	//          p1 primeste p
	//          daca val este mai mica decat valoarea din p
	//          atuncip primeste copilul stang
	//          altfel daca val este mai mare decat valoarea din p
	//                   atunci p primeste copilul drept
	//                   sf.daca
	//          sf.daca
```

```
        // sf.c.t.
        // daca val este mai mic decat valoarea din p1
        // atunci copilul stang al lui p1 primeste un nod nou initializat cu val
        // altfel copilul drept al lui p1 primeste un nod nou initializat cu val
        // sf.daca
}
```

Pentru început implementați varianta recursivă și o să continuăm cu o parcurgere ca să vedem cum afișăm arborele construit.

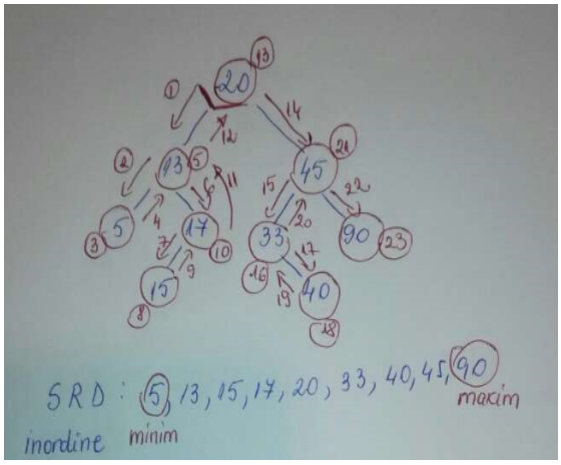
## 3. Parcurgeri

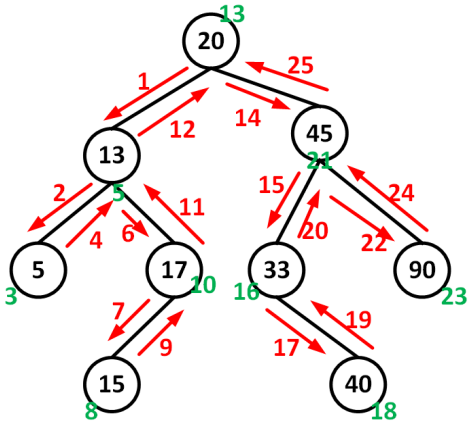
Puteți folosi pentru simulare: https://www.cs.usfca.edu/~galles/visualization/BST.html
Vom insera într-un arbore BST nodurile: 20, 13,45, 5,17, 33, 90, 15, 40. Arborele va arăta în felul următor:




### 3.1. Parcurgerea in inordine (S-R-D)

Se referă la vizitarea subarborelui stâng, a rădăcinii și a subarborelui drept

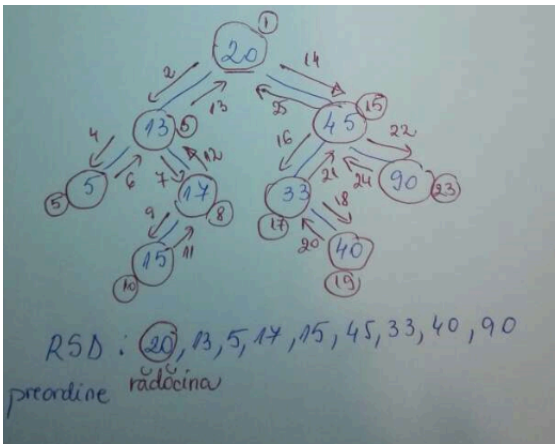


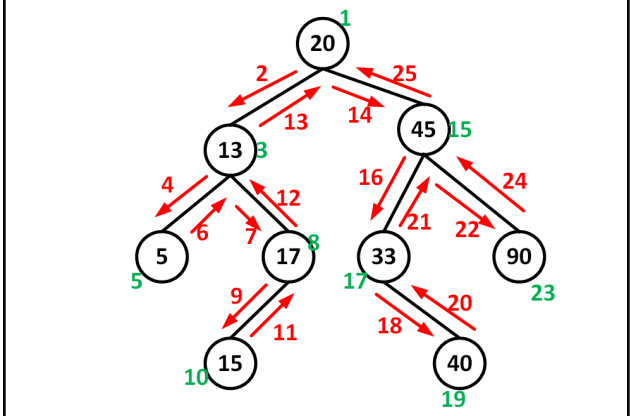




**SRD:** **5, 13, 15, 17, 20, 33, 40, 45, 90**
**(inordine)** **minim** **maxim**

```
void Inordine(Nod* rad)
{      if(rad)
       {
              Inordine(rad->stg);
              cout<<rad->data<<" ";
              Inordine(rad->drt);
       }
}
```

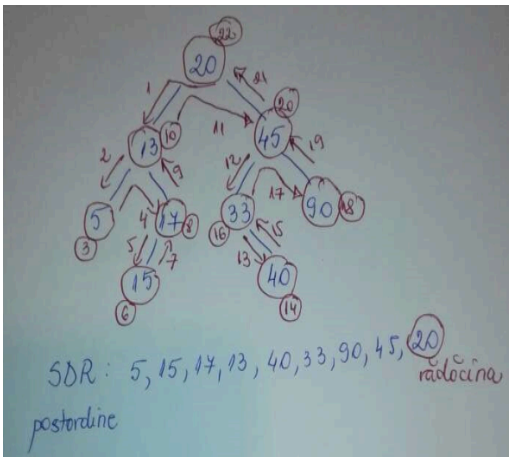
**3.2.Parcurgerea in preordine (R-S-D)** Întâi procesez rădăcina, după care continui cu subarborii în ordine de la stânga la dreapta

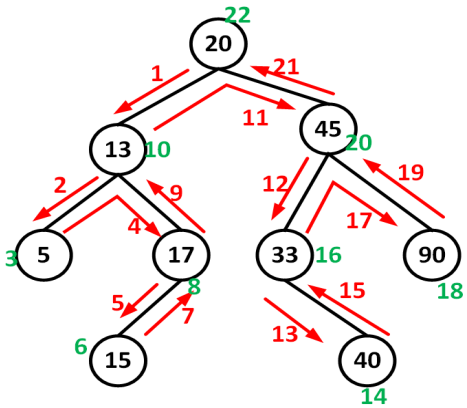



**RSD:** **20, 13, 5, 17, 15, 45, 33, 40, 90**
**(preordine)** **rădăcina**

**3.3.Parcurgerea in postordine (S-D-R)** Întâi procesez subarborii stâng și drept, la final rădăcina



**SDR: 5, 15, 17, 13, 40, 33, 90, 45, 20**
**(postordine) rădăcina**

Implementările parcurgerilor R-S-D și S-D-R sunt similare cu S-R-D cu convenția respectării ordinii de procesare

Să vedem cum le apelăm în main

```
Nod *rad;
cout<<"\n Initializare…";
InitTree(rad);
cout<<"\n Inserare valori in arbore ....";
for(int i=1;i<=5;i++)
      Insert(rad, i);
cout<<"\n Inordine: ";
Inordine(rad);
```

Desenați pe hârtie cum va arăta arborele nostru în acest caz

Dar dacă folosim o secvență de inițializare de tipul

```
Nod *rad;
cout<<"\n Initializare…";
InitTree(rad);
cout<<"\n Inserare valori in arbore ....";
for(int i=5;i>=1;i--)
      Insert(rad, i);
cout<<"\n Inordine: ";
Inordine(rad);
```

Dar în cazul în care procedăm astfel

```
Nod *rad;
int val;
cout<<"\n Initializare…";
InitTree(rad);
cout<<"\n Inserare valori in arbore ....";
cout<<"\n Se vor introduce pe rand valorile 7, 4, 12, 10, 5, 3, si 8\n\n";
for(int i=1;i<=7;i++)
{
      cout<<"\n Introduceti valoarea: ";
      cin>>val;
      Insert(rad, val);
}
cout<<"\n Inordine: ";
Inordine(rad);
```

În fișierul pe care trebuie să îl încărcați la finalul ședinței, vă rog să includeți și desenele.

4. **Căutarea în BST**

Ne vom folosi de proprietatea definitorie a unui BST referitoare la relația de ordine dintre valorile din rădăcină și din subarbori pentru a putea ghida căutarea în subarborele corespunzător

### 4.1. Varianta recursivă a funcției de căutare

```
int Search(Nod*rad, int val)
{
	// daca exista radacina
	// atunci	daca valoarea din radacina este egala cu val
	//			atunci		returnez 1;
	//		altfeldaca valoarea din radacina este mai mare ca val
	//				atunci caut in subarborele stang
	//				altfelcaut in subarborele drept
	//			sf.daca
	//		sf.daca
	// altfel	returnez 0
	// sf.daca
}
```

### 4.2. Varianta iterativă a funcției de căutare

```
int Search(Nod* rad, int val)
{
	// salvam radacina arborelui intr-un p
	// cat timp p nu este NULL si data din p este diferita de val
	//	daca data din p este mai mare decat val
	//	atunci		p se muta pe subarborele stang
	//	altfel		p se muta pe subarborele drept
	//	sf.daca
	// sf.c.t.
	// daca p este nenul
	//	atunci retuneaza 1
	//	altfel returneaza 0
	// sf.daca
}
```

### 5. Ștergerea

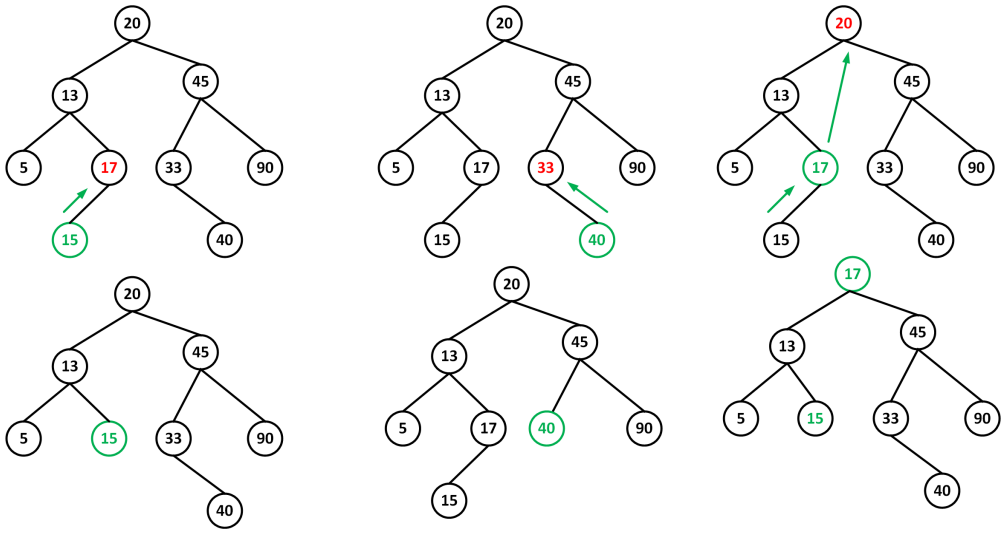
Pentru operația de ștergere vom avea de tratat mai multe cazuri

Ștergerea unui nod oarecare

Ștergerea rădăcinii (va trebui să înlocuim rădăcina ștearsă cu un nod din arbore astfel încât să se păstreze proprietatea de BST).

Vom discuta trei situații: Dacă rădăcina are doar un subarbore, ea va fi înlocuită de unul din nodurile copil, stânga sau dreapta. Dacă rădăcina are ambii subarbori, atunci noua rădăcină va trebui sa fie mai mare decât toate nodurile din stânga și mai mică decât toate nodurile din dreapta. Rădăcina va fi înlocuită astfel de nodul cu valoarea maximă din subarborele stâng al vechii rădăcini) iar RemoveGreatest este funcția care va depista nodul cu pricina.

```
void Delete(Nod*&rad, int val)
{
	// daca rad nu este nula
	// atunci	daca valoarea din radacina este egala cu val
	//		atunci sterg radacina rad
	//		altfel daca valoarea din radacina este mai mica decat val
	//			atunci apelez stergerea pentru subarborele drept
	//			altfel daca valoarea din radacina este mai mare decat
	//				val
	//				atunci apelez stergerea pentru subarborele stang
	//				altfelvaloarea nu exista in arbore
	//				sf.daca
	//			sf.daca
	//		sf.daca
	// sf.daca
}
```




```
void DeleteRoot(Nod* &rad)
{
	// daca nodul radacina exista
	// atunci	daca nu exista copilul stanga al radacinii
	//		atunci p primeste copilul dreapta al radacinii
	//			sterg radacina
	//			radacina primeste ce am salvat in p
	//		altfel daca radacina nu are copil dreapta
	//			atunci p primeste copilul stanga al radacinii
	//				sterg radacina
	//				radacina primeste ce am salvat in p
	//			altfel p primeste nodul cel mai mare eliminat din
	//				subarborele stang
```

```
	//                          subarborele stang al lui p primeste
	//                              subarborele stang al lui rad
	//                          subarborele drept al lui p primeste
	//                              subarborele drept al lui rad
	//                          sterg radacina
	//                          radacina primeste ce am salvat in p
	//                   sf.daca
	//           sf.daca
	// sf.daca
}

Nod* RemoveGreatest(Nod*& rad)
{
	// daca radacina rad nu este nula
	// atunci   daca copilul drept al radacinii este nul
	//          atuncip primeste r
	//                r isi primeste subarborele stang
	//                returnam p
	//          altfelreturnam nodul cel mai mare eliminat de pe
	//                              subarborele drept al radacinii
	//          sf.daca
	// altfel returnam NULL
	// sf.daca
}
```

În plus veți mai avea de implementat și variantele iterative pentru inserare, căutare și ștergere

### 6. Eliberarea de memorie

Eliberez mai întâi subarborii și la final revin și eliberez rădăcina:

```
void FreeMemTree(Nod*& rad)
{     //declar un pointer p
	// daca rad nu este NULL
	//    atunci      salvez radacina arborelui in p
	//                apelez eliberarea de memorie pe subarborele stang
	//                apelez eliberarea de memorie pe subarborele drept
	//                sterg p, p devine NULL
	// sf.daca
}
```

**Header.h**

```
#pragma once
#include <iostream>
using namespace std;

struct Nod{
	int data;
	Nod *stg, *drt;
};

void InitTree(Nod*& rad);
Nod* MakeNod(int val);
void Insert(Nod*&rad, int val);
void BuildTree(Nod*& rad);//SD

void Inorder(Nod* rad);
void Preorder(Nod* rad);
void Postorder(Nod* rad);
void IndentedListing(Nod* rad, int level);//SD

int Search(Nod* rad, int val);

void Delete(Nod*&rad, int val);
void DeleteRoot(Nod*&rad);
Nod* RemoveGreatest(Nod*&rad);

void FreeMemTree(Nod*&rad);

void InsertIter(Nod*&rad, int val); // SDA
int SearchIter(Nod* rad, int val); //SDA
void DeleteIter(Nod*&rad, int val); //SDA

bool MinimVal(Nod* rad, int& min); //SD
bool MaximVal(Nod* rad, int& max); //SD
```

**Functii.cpp**

```
#include "Header.h"

void InitTree(Nod* & rad)
{
	rad = 0;
}

Nod* MakeNod(int val)
{
	Nod* p;
```

```
	p=new Nod;
	p->data=val;
	p->stg=0;
	p->drt=0;
	return p;
}
//...
void Inordine(Nod* rad)
{
	if(rad)
	{
		Inordine(rad->stg);
		cout<<rad->data<<" ";
		Inordine(rad->drt);
	}
}

//...
```

**Main.cpp**

```
#include "Header.h"
int main()
{
	Nod *rad;
	int val;

	cout<<"\n Initializare…";
	InitTree(rad);

	cout<<"\n Inserare valori in arbore ....";
	cout<<"\n Se vor introduce pe rand valorile 7, 4, 12, 10, 5, 3, si
	8\n\n";
	for(int i=1;i<=7;i++)
	{
		cout<<"\n Introduceti valoarea: ";
		cin>>val;
		Insert(rad, val);
	}

	cout<<"\n Afisare in \n \t Inordine: ";
	Inordine(rad);

	//...

	system("PAUSE");
	return 0;
}
```

O varianta de implementare poate fi

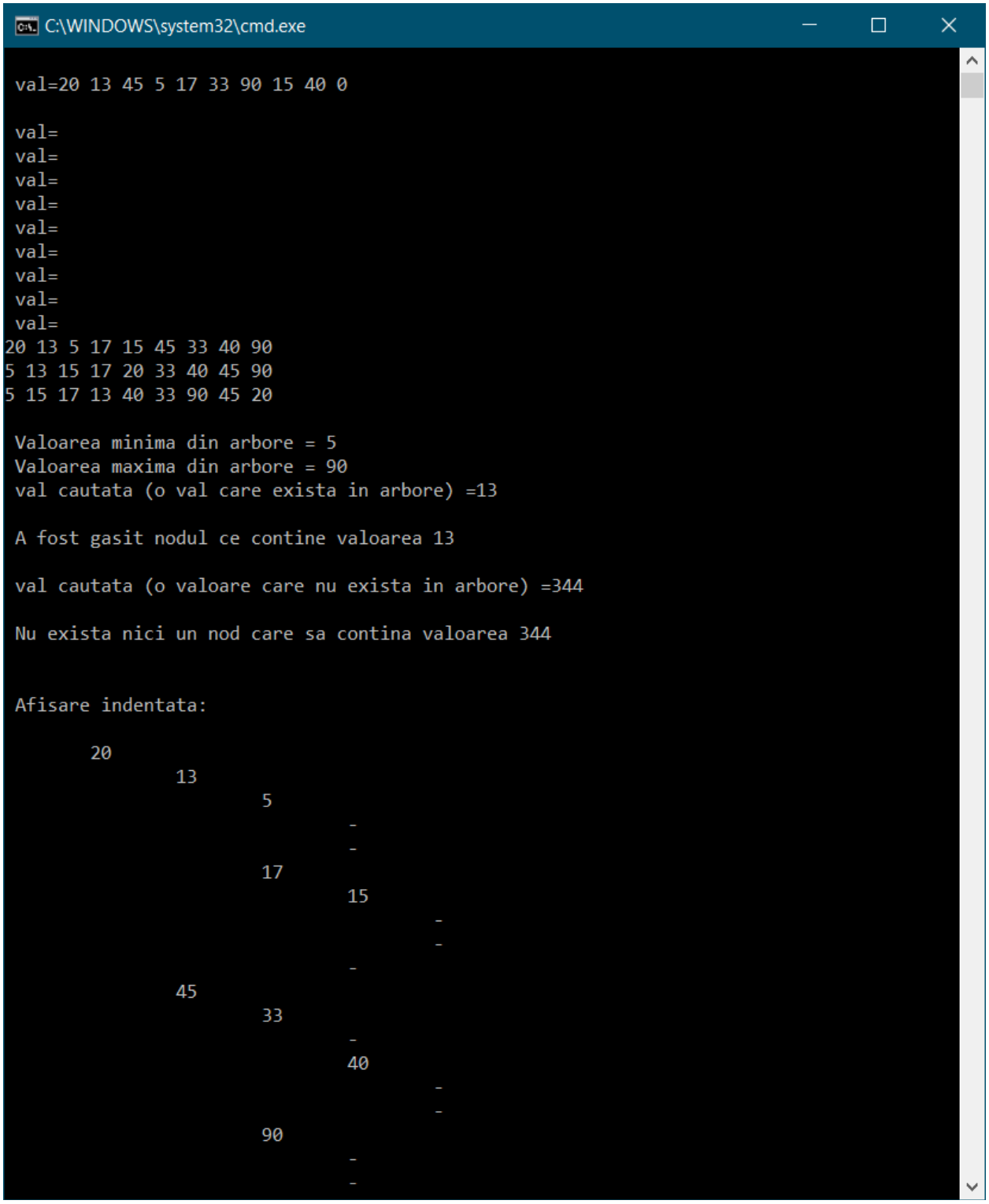
BST (Global Scope)

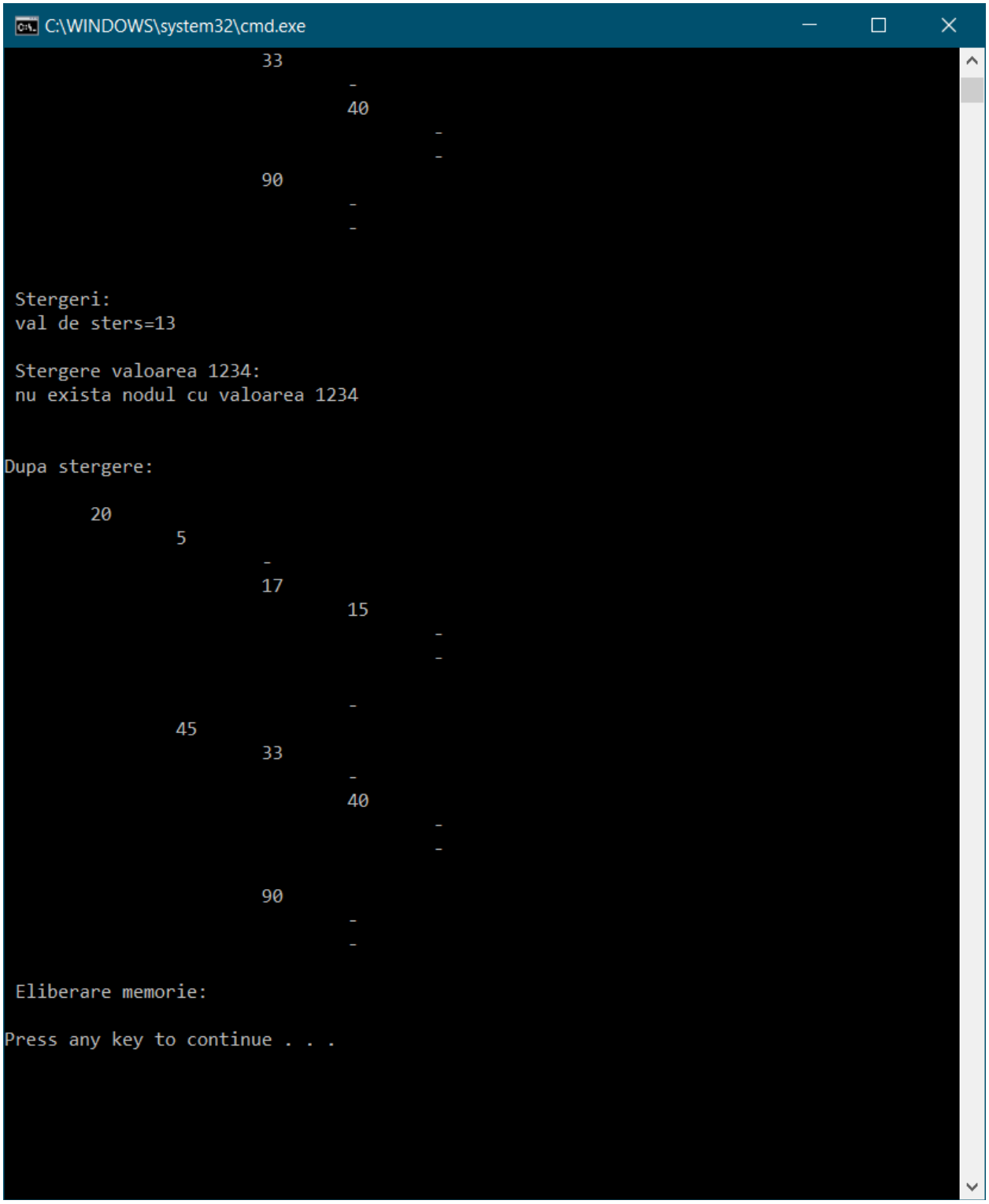
```
#pragma once
#include <iostream>
using namespace std;

struct Nod {
    int data;
    Nod* stg;
    Nod* drt;
};

void Insert(Nod*& root, int val);
void BuildTree(Nod*& root);

void Preorder(Nod* root);
void Inorder(Nod* root);
void Postorder(Nod* root);
void IndentedListsing(Nod* root, int level);

void DeleteRoot(Nod*& root);
void DeleteNod(Nod*& root, int val);
Nod* RemoveGreatest(Nod*& root);

Nod* Search(Nod* root, int val);
int SearchValue(Nod* root, int val);

void FreeMemory(Nod*& root);

bool MinimVal(Nod* rad, int& min);
bool MaximVal(Nod* rad, int& max);
```

BST (Global Scope)

```
#include "Header.h"

int main()
{
    Nod* root = NULL;
    Nod* searched = NULL;
    int val;

    BuildTree(root);

    cout << endl;
    Preorder(root);
    cout << endl;
    Inorder(root);
    cout << endl;
    Postorder(root);
    cout << endl;

    if (MinimVal(root, val))
        cout << "\n Valoarea minima din arbore = " << val;
    if (MaximVal(root, val))
        cout << "\n Valoarea maxima din arbore = " << val;

    cout << "\n val cautata (o val care exista in arbore) =";
    cin >> val;
    searched = Search(root, val);
    if (searched)
        cout << "\n A fost gasit nodul ce contine valoarea " << searched->data << "\n";
    else
        cout << "\n Nu exista nici un nod care sa contina valoarea " << val;

    cout << "\n val cautata (o valoare care nu exista in arbore) =";
    cin >> val;
    if (SearchValue(root, val))
        cout << "\n A fost gasit nodul ce contine valoarea " << searched->data << "\n";
    else
        cout << "\n Nu exista nici un nod care sa contina valoarea " << val;
```

```
    cout << "\n\n\n Afisare indentata:\n";
    IndentedListsing(root, 1);

    cout << endl << endl;
    cout << "\n Stergeri:";
    cout << "\n val de sters=";
    cin >> val;
    DeleteNod(root, val);

    cout << "\n Stergere valoarea 1234:";
    DeleteNod(root, 1234);
    cout << endl << endl << "Dupa stergere:\n";
    IndentedListsing(root,1);
    cout << endl;

    cout << "\n Eliberare memorie:";
    FreeMemory(root);
    root = NULL;
    cout << endl << endl;

    system("PAUSE");
    return 0;
}
```


C:\WINDOWS\system32\cmd.exe
val=20 13 45 5 17 33 90 15 40 0
val=
val=
val=
val=
val=
val=
val=
val=
val=
20 13 5 17 15 45 33 40 90
5 13 15 17 20 33 40 45 90
5 15 17 13 40 33 90 45 20
Valoarea minima din arbore = 5
Valoarea maxima din arbore = 90
val cautata (o val care exista in arbore) =13
A fost gasit nodul ce contine valoarea 13
val cautata (o valoare care nu exista in arbore) =344
Nu exista nici un nod care sa contina valoarea 344
Afisare indentata:
20
13
5
-
-
17
15
-
-
-
45
33
-
40
-
-
90
-
-


C:\WINDOWS\system32\cmd.exe
33
-
40
-
-
90
-
-
Stergeri:
val de sters=13
Stergere valoarea 1234:
nu exista nodul cu valoarea 1234
Dupa stergere:
20
5
-
17
15
-
-
-
45
33
-
40
-
-
90
-
-
Eliberare memorie:
Press any key to continue . . .

## Tema de laborator:

Assignment L09 pe Moodle - un singur document cu rezolvările
Deadline +1h față de ultima oră de laborator

## Structuri de Date și Algoritmi (anul II AIA)

Cod și rezultate la rulare pentru

**1.** Se citeste de la intrare un sir de valori numerice intregi, pe o linie, separate de spatii, sir care se incheie cu o valoare 0.

a) Sa se introduca valorile citite intr-un arbore binar de cautare (exclusiv valoarea 0 care incheie sirul).

b) Sa se afiseze continutul arborelui in inordine, preordine si postordine.

c) Se citeste o valoare pentru care sa se raspunda daca este sau nu continuta in arbore.

d) Se citeste o valoare care sa fie stearsa din arbore si apoi sa se afiseze arborele in inordine.

*Implementati variantele recursive ale functiilor de cautare, inserare, stergere.*

**2.** Reluati problema anterioara, dar de aceasta data realizati implementarile iterative (nerecursive) ale functiilor de cautare, inserare, stergere.

Analitic veți avea de rezolvat

Exerciții:

1. Se introduc următoarele valori (chei) într-un BST: 30, 20, 38, 10, 35, 25, 42, 40, 22. Precizați care este înălțimea arborelui binar de căutare și câte frunze sunt. Cum arată afișarea în postordine?
2. Reprezentarea în preordine a unui arbore binar este: 22, 11, 5, 7, 15, 33, 30, 41, 35. Este acesta un BST? Nodurile de cheie minimă și maximă sunt frunze?
3. Reprezentați BST-ul corespunzător următoarei afișări în postordine: 15, 8, 30, 25, 20, 38, 40, 44, 35.

***NOTARE***

Problema 1: citire valori 0.5p; a) 1.5p; b) 1.5p; c) 1.5p; d) 2p;

Problema 2: cautare 1.5p; inserare 1.5p; stergere 3p.

# Structuri de Date (anul I CTI)

Cod și rezultate la rulare pentru:

Se citește de la intrare un șir de valori numerice întregi, pe o linie, separate de spații, șir care se încheie cu o valoare 0 :

a) Să se introducă valorile citite într-un arbore binar de căutare (exclusiv valoarea 0 care încheie șirul) ;
b) Să se afișeze în inordine conținutul arborelui ;
c) Se citește o valoare pentru care să se verifice dacă este sau nu conținută în arbore ;
d) Se citește o valoare care să fie ștearsă din arbore și apoi să se afișeze arborele în inordine ;
e) Să se scrie funcții pentru afișarea nodurilor de cheie minimă respectiv maximă din arborele binar de căutare ;
f) Să se afișeze arborele indentat, asemănător afișării folderelor în Explorer. De exemplu, arborele


va fi afișat ( ”-” semnifică descendent nul) :

```
20
        13
                5
                        -
                        -
                17
                        15
                                -
                                -
                        -
        45
                33
                        -
                        40
                                -
                                -
                90
                        -
                        -
```

g) Dealocați zona de memorie ocupată de arbore.

Analitic rezolvare pentru

Exerciții:

1. Se introduc următoarele valori (chei) într-un BST: 30, 20, 38, 10, 35, 25, 42, 40, 22. Precizați care este înălțimea arborelui binar de căutare și câte frunze sunt. Cum arată afișarea în postordine?
2. Reprezentarea în preordine a unui arbore binar este: 22, 11, 5, 7, 15, 33, 30, 41, 35. Este acesta un BST? Nodurile de cheie minimă și maximă sunt frunze?
3. Reprezentați BST-ul corespunzător următoarei afișări în postordine: 15, 8, 30, 25, 20, 38, 40, 44, 35.