L04. ARBORI BINARI DE CĂUTARE (BST)

Header.h

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

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

void IndentedListsing(Nod* root, int level);

void FreeMemory(Nod*& root);
```

Functii.cpp

```
#include "Header.h"

void Insert(Nod*& root, int val)
{
	if (root == 0)
	{
		root = new Nod;
		root->data = val;
		root->stg = NULL;
		root->drt = NULL;
	}
	else
```

```
            if (val < root->data)
                  Insert(root->stg, val);
            else
                  if (val > root->data)
                        Insert(root->drt, val);
}

void BuildTree(Nod*& root)
{
      int val;
      cout << "\n val=";
      cin >> val;
      while (val)
      {
            Insert(root, val);
            cout << "\n val=";
            cin >> val;
      }
}

void Preorder(Nod* root)
{
      if (root)
      {
            cout << root->data << " ";
            Preorder(root->stg);
            Preorder(root->drt);
      }
}

void Inorder(Nod* root)
{
      if (root)
      {
            Inorder(root->stg);
            cout << root->data << " ";
            Inorder(root->drt);
      }
}

void Postorder(Nod* root)
{
      if (root)
      {
            Postorder(root->stg);
            Postorder(root->drt);
            cout << root->data << " ";
      }
}

void DeleteRoot(Nod*& root)
```

```
{
	Nod* p = root;
	if (root->stg == 0)
		root = root->drt;
	else
		if (root->drt == 0) root = root->stg;
	else
	{
		root = RemoveGreatest(root->stg);
		root->stg = p->stg;
		root->drt = p->drt;
	}
	delete p;
}

void DeleteNod(Nod*& root, int val)
{
	if (root == 0)
		cout << "\n nu exista nodul cu valoarea " << val << "\n";
	else
	{
		if (root->data == val)
			DeleteRoot(root);
		else
			if (root->data > val)
				DeleteNod(root->stg, val);
			else
				DeleteNod(root->drt, val);
	}
}

Nod* RemoveGreatest(Nod*& root)
{
	Nod* p;
	if (root->drt == 0)
	{
		p = root;
		root = root->stg;
		return p;
	}
	else
		return RemoveGreatest(root->drt);
}

Nod* Search(Nod* root, int val)
{
	if (root)
	{
		if (root->data == val)
			return root;
		else
```

```
                if (root->data > val)
                      return Search(root->stg, val);
                else
                      return Search(root->drt, val);
      }
      else
            return NULL;
}


void IndentedListsing(Nod* root, int level)
{
      if (root)
      {
            cout << endl;
            for (int i = 0;i < level;i++)
                  cout << "\t";
            cout << root->data;
            for (int i = 0;i < level;i++)
                  cout << "\t";
            IndentedListsing(root->stg, level+1);
            for (int i = 0;i < level;i++)
                  cout << "\t";
            IndentedListsing(root->drt, level+1);
            for (int i = 0;i < level;i++)
                  cout << "\t";
      }
      else
      {
            cout << endl;
            for (int i = 0;i < level;i++)
                  cout << "\t";
            cout << "-";
      }
}

void FreeMemory(Nod*& root)
{
      Nod* p;
      if (root)
      {
            p = root;
            FreeMemory(root->stg);
            FreeMemory(root->drt);
            delete p;
            p = NULL;
      }
}
```

Main.cpp

```
#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;

	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;
	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\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;

	return 0;
}
```