Ministerul Educaţiei, Culturii și Cercetării

al Republicii Moldova   
  
Universitatea Tehnică a Moldovei   
  
  
  
  
  
  
  
RAPORT

Lucrarea de laborator nr. 3   
la Structuri de date si algoritmi

A efectuat: st. gr. TI-211 Popa Cătălin   
  
A verificat: Stratulat Ștefan

UTM, Chișinău 2021

**Tema:**

Implementarea tipului de date abstract “Arbore binar” în limbajul C.

**Scopul lucrării**

Scopul lucrării este de a familiariza studentul cu mecanismul de creare a arborelui binar și operații elementare asupra acestuia, utilizînd pentru aceasta limbajul C.

**Problema:**

1. **Să se elaboreze un program ce va aloca dinamic un arbore binar de structuri și va realiza următoarele funcții, unde funcțiile date reprezintă opțiuni organizate într-un meniu în cadrul programului:**

* citirea de la tastatură a elementelor arborelui;
* afișarea la consolă a elementelor arborelui;
* calcularea adîncimii arborelui;
* căutarea unui nod în baza unei chei de căutare;
* calcularea adîncimii unui nod ( nodul va fi selectat în baza unei chei de căutare );
* calcularea înălțimii unui nod ( nodul va fi selectat în baza unei chei de căutare );
* afișarea tuturor frunzelor arborelui;
* curățirea elementelor arborelui;
* eliberarea memoriei arborelui.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <conio.h>

#define COUNT 10

//////////////////////////////////////////////////////////////////

typedef struct BinaryTreeNode

{

int data;

struct BinaryTreeNode \* left;

struct BinaryTreeNode \* right;

}BinaryTreeNode;

typedef struct BinaryTree

{

struct BinaryTreeNode \* root;

}BinaryTree;

//////////////////////////////////////////////////////////////////

BinaryTreeNode \* newTreeNode(int data)

{

BinaryTreeNode \* tree\_node = calloc(sizeof(BinaryTreeNode),1);

tree\_node->data = data;

return tree\_node;

}

//////////////////////////////////////////////////////////////////

BinaryTree \* create\_binary\_tree()

{

BinaryTree \* local = calloc(sizeof(BinaryTree),1);

local->root = NULL;

return local;

}

//////////////////////////////////////////////////////////////////

void insert\_new\_node(BinaryTreeNode \*current\_node, BinaryTreeNode \* new\_node )

{

if(current\_node)

{

if(current\_node->data>new\_node->data)

{

if(current\_node->left)

{

insert\_new\_node(current\_node->left,new\_node);

}

else

{

current\_node->left = new\_node;

}

}

else

{

if(current\_node->right)

{

insert\_new\_node(current\_node->right,new\_node);

}

else

{

current\_node->right = new\_node;

}

}

}

}

//////////////////////////////////////////////////////////////////

void insert\_data\_in\_binary\_tree(BinaryTree \*insert)

{

int data;

if(insert)

{

printf("Introduceti data: \n");

scanf("%d",&data);

BinaryTreeNode \* new\_node = newTreeNode(data);

if(insert->root)

{

insert\_new\_node(insert->root,new\_node);

}

else

{

insert->root = new\_node;

}

}

}

//////////////////////////////////////////////////////////////////

//parcurgere in inordine (SVD)

void inorder(BinaryTreeNode \* nodes)

{

if (nodes != NULL)

{

inorder(nodes->left);

printf("|%d|", nodes->data);

inorder(nodes->right);

}

}

//////////////////////////////////////////////////////////////////

//parcurgere in preordine (VSD)

void preorder(BinaryTreeNode \* nodes)

{

if(nodes != NULL)

{

printf("|%d|",nodes->data);

preorder(nodes->left);

preorder(nodes->right);

}

}

//////////////////////////////////////////////////////////////////

//parcurgere in postordine (SDV)

void postorder(BinaryTreeNode \* nodes)

{

if(nodes != NULL)

{

postorder(nodes->left);

postorder(nodes->right);

printf("|%d|",nodes->data);

}

}

//////////////////////////////////////////////////////////////////

void Tree2D(BinaryTreeNode \* node, int space)

{

if (node == NULL)

return;

space += COUNT;

Tree2D(node->right, space);

printf("\n");

for (int i = COUNT; i < space; i++)

printf(" ");

printf("%d\n", node->data);

Tree2D(node->left, space);

}

void print2(BinaryTreeNode \*node)

{

Tree2D(node, 0);

}

//////////////////////////////////////////////////////////////////

void frunze(BinaryTreeNode \*root)

{

if(!root)

{

return;

}

if(root->left == NULL && root->right == NULL)

{

printf("|%d| ",root->data);

return;

}

if(root->left)

{

frunze(root->left);

}

if(root->right)

{

frunze(root->right);

}

}

//////////////////////////////////////////////////////////////////

void DeleteNode(BinaryTreeNode \*node)

{

if(node)

{

DeleteNode(node->left);

DeleteNode(node->right);

free(node);

}

}

//////////////////////////////////////////////////////////////////

void clear\_tree(BinaryTree \*tree)

{

if(tree)

{

if(tree->root)

{

DeleteNode(tree->root);

tree->root = NULL;

}

}

}

//////////////////////////////////////////////////////////////////

void DeleteTree(BinaryTree \*tree)

{

if(tree)

{

clear\_tree(tree);

free(tree);

}

}

//////////////////////////////////////////////////////////////////

int TreeH(BinaryTreeNode \* node)

{

if(node == NULL)

{

return -1;

}

else

{

int mLeft = TreeH(node->left);

int mRight = TreeH(node->right);

if(mLeft>mRight)

{

return (mLeft+1);

}

else

{

return (mRight+1);

}

}

}

//////////////////////////////////////////////////////////////////

BinaryTreeNode \*search(BinaryTreeNode \*node, int key,int num)

{

if (node == NULL)

{

return NULL;

}

if(node->data == key)

{

printf("Nodul -%d- :\n",key);

printf("Adancimea este : %d\n",num);

return node;

}

if(node && key<node->data)

{

num++;

search(node->left,key,num);

}

if(node && key>node->data)

{

num++;

search(node->right,key,num);

}

// printf("%d\n",num );

}

//////////////////////////////////////////////////////////////////

void TreeDepth(BinaryTreeNode \* node, int key,int counter)

{

BinaryTreeNode \*current\_node = NULL;

current\_node = search(node,key,counter);

}

//////////////////////////////////////////////////////////////////

int HTree(BinaryTreeNode \*node)

{

int left,right;

if(node == 0)

{

return 0;

}

if (node->left != NULL)

{

left = HTree(node->left);

}

else

{

left = -1;

}

if (node->right != NULL)

{

right = HTree(node->right);

}

else

{

right = -1;

}

int max = left > right ? left : right;

return max+1;

}

//////////////////////////////////////////////////////////////////

void NodeHkey(BinaryTreeNode \* node,int key)

{

BinaryTreeNode \*current\_node = NULL;

current\_node = search(node,key,0);

printf("Inaltimea nodului este %d",HTree(current\_node));

}

//////////////////////////////////////////////////////////////////

int menu()

{

int error\_input\_found = 0;

menu\_start:

system("cls");

printf("|--------------------------------------|\n");

printf("| A efectuat: Popa Catalin |\n");

printf("| Grupa: TI-211 |\n");

printf("| A verificat: Stratulat Stefan |\n");

printf("|--------------------------------------|\n");

printf("|----------------------//Menu\\-------------------------|\n");

printf("|1.Creaza arborele |\n");

printf("|2.Afisarea elementelor |\n");

printf("|3.Calculara adancimii arborelui |\n");

printf("|4.Cautarea unui nod in baza unei chei de cautare |\n");

printf("|5.Calcularea adancimii unui nod |\n");

printf("|6.Calcularea inaltimii unui nod |\n");

printf("|7.Afisarea tuturor frunzelor arborelui |\n");

printf("|8.Curatirea elementelor arborelui |\n");

printf("|9.Eliberarea memoriei arborelui |\n");

printf("|0.Opriti programul |\n");

printf("|------------------------------------------------------|\n");

int command = 100;

if(error\_input\_found)

{

printf("\n\nCommanda trebuie sa fie o optiune de mai sus!!![0-9]\n");

}

printf("\n\nSelectati commanda: ");

scanf("%d",&command);

if((command>9)||(command<0))

{

error\_input\_found = 1;

goto menu\_start;

}

return command;

}

//////////////////////////////////////////////////////////////////

int main()

{

int num,key;

BinaryTree \*tree = NULL;

int command=100;

while(command)

{

command = menu();

if(command == 1)

{

if(tree != NULL)

{

DeleteTree(tree);

}

tree = create\_binary\_tree();

printf("Dati numarul de noduri\n");

scanf("%d",&num);

for(int i=0; i<num; i++)

{

insert\_data\_in\_binary\_tree(tree);

}

}

switch (command)

{

case 2:

{

if(tree->root == NULL)

{

printf("Arborele este null\n");

}

else

{

print2(tree->root);

printf("\nIn inordine(SVD):\n");

inorder(tree->root);

printf("\nIn postordine(SDV):\n");

postorder(tree->root);

printf("\nIn preordine(VSD):\n");

preorder(tree->root);

}

break;

}

case 3:

{

printf("Adancimea arborelui este: %d\n",TreeH(tree->root));

break;

}

case 4:

{

printf("Dati un nod cautat:\n");

scanf("%d",&key);

if(search(tree->root,key,0))

{

printf("Nodul este prezent in arbore\n");

}

else

{

printf("Nodul respectiv nu este in arbore\n");

}

break;

}

case 5:

{

printf("Introduceti nodul :\n ");

scanf("%d",&key);

TreeDepth(tree->root,key,0);

break;

}

case 6:

{

printf("Introduceti nodul: \n");

scanf("%d",&key);

NodeHkey(tree->root,key);

break;

}

case 7:

{

printf("\nFrunzele sunt:\n");

frunze(tree->root);

break;

}

case 8:

{

clear\_tree(tree);

printf("Elementele arborelui au fost curatite\n");

break;

}

case 9:

{

DeleteTree(tree);

tree = NULL;

printf("\nMemoria a fost eliberata cu succes!");

printf("\nArborele a fost sters\n");

break;

}

}

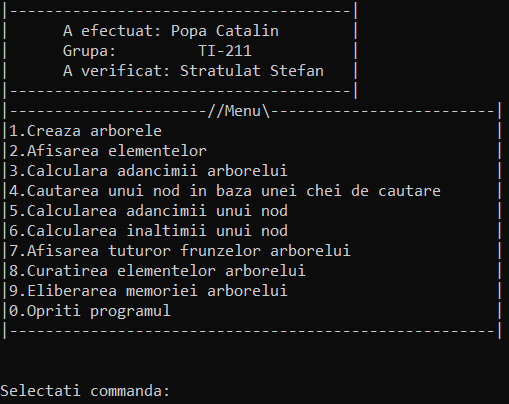
printf("\n\nTastati orice buton pentru continuare!\n");

\_getch();

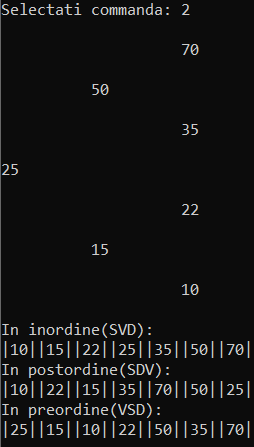
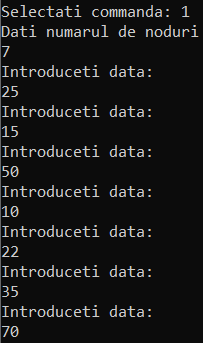
}

return 0;

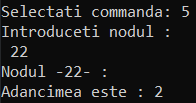
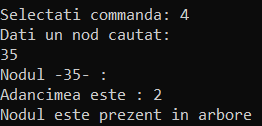
}



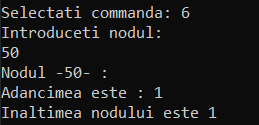
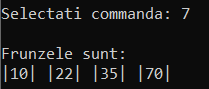
1)Crearea arborelui: 2)Afisarea: 3)Adancimea arborelui:



4)Cautarea unui nod: 5)Adancimea unui nod:



6)Inaltimea unui nod: 7)Afisarea frunzelor:



8)Curatirea elementelor arborelui: 9)Eliberarea memoriei arborelui:

