Министерство образования и науки Кыргызской Республики

Кыргызский государственный технический университет

им.И.Раззакова

Факультет информационных технологий

Кафедра «Программное обеспечение компьютерных систем»

Направление:710400 «Программная инженерия»

ОТЧЕТ

По дисциплине: «Алгоритмы и структуры данных»

Лабораторная работа №8

Тема: «Работа с деревом»

Выполнил: студент группы

ПИ(б)-2-19 Улан уулу Нурдин

Проверила: Валеева А. А.

Бишкек – 2020

**Практическое задание**

**I. Ответы на контрольные вопросы**

1. **Что такое дерево?**

Дерево является нелинейной динамической структурой хранения данных. Дерево состоит из узлов или вершин, которые содержат поля данных и указатели на другие узлы или вершины.

1. **Какое дерево называется упорядоченным?**

Упорядоченное дерево - структура, упорядочивающая элементы дерева отношением ‘ <,> ’.

1. **Что такое полу-степень исхода вершины дерева?**

Полу-степень исхода вершины-число элементов, являющимися «потомками» данного элемента

1. **Что такое полу-степень захода вершины дерева?**

Полу-степень захода вершины- количество «предков» элемента.

1. **Какие элементы дерева называются узлами?**

Узлами дерева называют элементы дерева, которые имеют хоть 1 потомка.

1. **Какое дерево называется бинарным?**

Бинарное дерево-дерево, узлы которого могут содержать максимум 2 «потомка».

1. **Чем отличаются бинарное дерево от двухсвязных списков?**

Двухсвязный список-линейная структура данных, а дерево-иерархическая.

1. **Опишите организацию связей в двоичном дереве**

Каждый элемент хранит в себе указатели на правый и на левый элемент, которые могут быть равны нулю (элемент является «листом»), кроме того каждый элемент хранит информацию о своём «родителе».

1. **Что такое обход дерева? Какие бывают виды обхода?**

Обход дерева-доступ к элементам дерева по правилу, заданному типом обхода.

Обходы бывают: Прямой (pre-order);

Симметричный (in-order);

В обратном порядке (post-order).

1. **Какие операции осуществляются над двоичными деревьями?**

Инициализация, добавление, удаление, подсчёт высоты, подсчёт листьев, поиск.

**II. Выполнить задания**

**1. Построить бинарное дерево одного из типов данных:**

1. строкового;
2. целочисленного;
3. символьного.

**2. Для бинарного дерева осуществить операции:**

* добавление вершины дерева;
* вывод дерева на экран;
* \*\* удаление вершины дерева;
* поиск вершины дерева по ключу;
* подсчет количества вершин дерева;
* подсчет количества листьев;
* уничтожение дерева.

**3. Выполнить обход дерева рекурсивным способом**

1. Нисходящим способом
2. Восходящим способом
3. Смешанным способом

int main()



void inition(Tree\*& Btree)



void add(Tree\*& Btree)



void output\_menu(Tree\*& Btree)



void output\_inor(Tree\*& Btree)



void output\_preo(Tree\*& Btree)



void output\_post(Tree\*& Btree)



void output(Tree\*& Btree, int j)



void del(Tree\*& Btree)



void Search(Tree\*& Btree)



int Count\_leaves(Tree\*& Btree)



void Count\_tops(Tree\*& Btree, int& count)



void delete\_tree(Tree\*& Btree)



int Search\_By\_El(Tree\* Btree, int x)



int Search\_By\_level(Tree\* Btree, int x)

 int Search\_By\_Parent(Tree\* Btree, int x)



int Search\_By\_Chield(Tree\* Btree, int x)

 int Search\_By\_Pop(Tree\* Btree, int x)



void push(Tree\*& Btree, int x, int lev)



void pop(Tree\*& Btree, int x)



**Текст программы**

#include <iostream>

#include <string>

#include <iomanip>

#include <fstream>

#include "Windows.h"

using namespace std;

struct Tree {

int item;

Tree\* left, \* right, \* parent;

};

int root = 0;

bool \_count = true, checkSearch, checkPop;

void inition(Tree\*& Btree);

void add(Tree\*& Btree);

void output\_menu(Tree\*& Btree);

void del(Tree\*& Btree);

void Search(Tree\*& Btree);

int Count\_leaves(Tree\*& Btree);

void Count\_tops(Tree\*& Btree, int& count);

void delete\_tree(Tree\*& Btree);

void output\_inor(Tree\*& Btree);

void output\_preo(Tree\*& Btree);

void output\_post(Tree\*& Btree);

void output(Tree\*& Btree, int j, int x);

void push(Tree\*& Btree, int x);

Tree\* pop(Tree\* Btree, int x);

int Search\_By\_Pop(Tree\* Btree, int x);

int Search\_By\_El(Tree\* Btree, int x, int j);

int Search\_By\_Parent(Tree\* Btree, int x);

int Search\_By\_Chield(Tree\* Btree, int x);

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Tree\* binary\_tree = NULL;

bool f = true;

int choice;

inition(binary\_tree);

while (f) {

int \_count\_tops = 0;

cout << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t\tГлавное Меню:\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t1. Добавление элемента\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t2. Вывод дерева\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t3. Удаление элемента\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t4. Поиск вершины дерева\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t5. Подсчет вершин дерева\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t6. Подсчет листьев дерева\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t7. Удалить дерево\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t8. Выход\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << endl;

cout << "Введите номер операции:";

cin >> choice;

system("cls");

switch (choice) {

case 1: add(binary\_tree); break;

case 2: output\_menu(binary\_tree); break;

case 3: del(binary\_tree); break;

case 4: Search(binary\_tree); break;

case 5: Count\_tops(binary\_tree, \_count\_tops); cout << "Кол - во вершин: " << \_count\_tops << endl; break;

case 6: cout << "Кол - во листьев: " << Count\_leaves(binary\_tree) << endl; break;

case 7: delete\_tree(binary\_tree); \_count = 0; break;

case 8: delete binary\_tree; f = false; break;

default: cout << "Такой операции не существует!!!" << endl << "Введите заново." << endl; break;

}

}

return 0;

}

void inition(Tree\*& Btree) {

\_count = true;

int x;

cout << "Инициализируйте корень:";

cin >> x;

root = x;

system("cls");

cout << "Элемент " << x << " инициализирован" << endl;

Btree = new Tree;

Btree->item = x;

Btree->left = Btree->right = Btree->parent = NULL;

}

void add(Tree\*& Btree) {

if (Btree == NULL) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

} else {

int x;

cout << "Выберите метод ввода: ";

cin >> x;

system("cls");

if (x == 1) {

ifstream sin;

sin.open("qq.txt");

while (!sin.eof()) {

sin >> x;

push(Btree, x);

cout << "Элемент " << x << " добавлен в дерево" << endl;

}

sin.close();

} else {

cout << "Введите новый элемент:";

cin >> x;

system("cls");

push(Btree, x);

cout << "Элемент " << x << " добавлен в дерево" << endl;

}

}

}

void output\_menu(Tree\*& Btree) {

if (Btree == NULL) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

}

else {

bool f = true;

int x;

while (f) {

cout << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t\tМеню вывода:\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t1. Inorder\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t2. Preorder\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t3. Postorder\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t4. В виде дерева\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t5. Назад\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << endl;

cout << "Введите номер операции:";

cin >> x;

system("cls");

switch (x) {

case 1: f = false; output\_inor(Btree); break;

case 2: f = false; output\_preo(Btree); break;

case 3: f = false; output\_post(Btree); break;

case 4: f = false; output(Btree, 0, 1); break;

case 5: f = false; break;

default: cout << "Такой операции не существует!!!" << endl << "Введите заново." << endl; break;

}

}

}

}

void output\_inor(Tree\*& Btree) {

if (Btree != NULL) {

output\_inor(Btree->left);

cout << Btree->item << "\t";

output\_inor(Btree->right);

}

}

void output\_preo(Tree\*& Btree) {

if (Btree != NULL) {

cout << Btree->item << "\t";

output\_inor(Btree->left);

output\_inor(Btree->right);

}

}

void output\_post(Tree\*& Btree) {

if (Btree != NULL) {

output\_inor(Btree->left);

output\_inor(Btree->right);

cout << Btree->item << "\t";

}

}

void output(Tree\*& Btree, int j, int x) {

if (Btree != NULL) {

output(Btree->right, j + 3, x + 1);

for (int i = 0; i < j; i++) {

cout << "\t";

}

cout << "info=(" << Btree->item << ")-lvl(" << x << ")";

if (j != 0) {

cout << "-p(" << Btree->parent->item << ")";

}

else {

cout << "-Корень";

}

cout << endl;

output(Btree->left, j + 3, x + 1);

}

}

void del(Tree\*& Btree) {

if (Btree == NULL) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

}

else {

int x;

cout << "Введите удаляемый элемент:";

cin >> x;

system("cls");

Search\_By\_Pop(Btree, x);

if (checkPop == false) {

cout << "Элемента " << x << " не существует!!!" << endl;

}

else {

cout << "Элемент " << x << " удален из дерева" << endl;

Btree = pop(Btree, x);

}

}

}

void Search(Tree\*& Btree) {

if (Btree == NULL) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

}

else {

int x, Elcheck;

cout << "Введите искоемый элемент:";

cin >> x;

Elcheck = Search\_By\_El(Btree, x, 1);

system("cls");

cout << "Элемент " << x;

if (checkSearch == false) {

cout << " не существует!!!" << endl;

}

else {

cout << " находиться на уровне: " << Search\_By\_El(Btree, x, 1) << endl;

if (Elcheck == 1) {

cout << "Левый потомок: ";

if (Btree->left != nullptr) {

cout << Btree->left->item << "\t";

}

else {

cout << "Отсустствует!!!\t";

}

cout << "Правый потомок: ";

if (Btree->right != nullptr) {

cout << Btree->right->item << endl;

}

else {

cout << "Отсустствует!!!\n";

}

cout << "\t\tОн корень!!!" << endl;

}

else {

Search\_By\_Chield(Btree, x);

cout << "\t\tЕго родитель: " << Search\_By\_Parent(Btree, x) << endl;

}

}

}

}

int Count\_leaves(Tree\*& Btree) {

if (\_count == false) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

}

else {

int count;

if ((Btree->left == NULL) && (Btree->right == NULL)) {

count = 1;

}

else {

count = 0;

}

if (Btree->left) {

count += Count\_leaves(Btree->left);

}

if (Btree->right) {

count += Count\_leaves(Btree->right);

}

return count;

}

return 0;

}

void Count\_tops(Tree\*& Btree, int& count) {

if (\_count == false) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

}

else {

if (Btree != NULL) {

Count\_tops(Btree->left, count);

if (Btree->left != NULL || Btree->right != NULL) {

count++;

}

Count\_tops(Btree->right, count);

}

}

}

void delete\_tree(Tree\*& Btree) {

if (\_count == false) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

}

else {

if (Btree != NULL) {

delete\_tree(Btree->left);

delete\_tree(Btree->right);

delete Btree;

Btree = NULL;

}

}

}

int Search\_By\_El(Tree\* Btree, int x, int j) {

if (Btree != NULL) {

if (x == Btree->item) {

checkSearch = true;

return j;

}

else {

checkSearch = false;

if (x < Btree->item) {

return Search\_By\_El(Btree->left, x, j + 1);

}

else if (x > Btree->item) {

return Search\_By\_El(Btree->right, x, j + 1);

}

}

}

return 0;

}

int Search\_By\_Parent(Tree\* Btree, int x) {

if (x == Btree->item) {

return Btree->parent->item;

}

if (x < Btree->item) {

return Search\_By\_Parent(Btree->left, x);

}

else if (x > Btree->item) {

return Search\_By\_Parent(Btree->right, x);

}

return 0;

}

int Search\_By\_Chield(Tree\* Btree, int x) {

if (Btree != NULL) {

if (x == Btree->item) {

cout << "Левый потомок: ";

if (Btree->left != nullptr) {

cout << Btree->left->item << "\t";

}

else {

cout << "Отсустствует!!!\t";

}

cout << "Правый потомок: ";

if (Btree->right != nullptr) {

cout << Btree->right->item << "\n";

}

else {

cout << "Отсустствует!!!\n";

}

return Btree->item;

}

else {

if (x < Btree->item) {

return Search\_By\_Chield(Btree->left, x);

}

else if (x > Btree->item) {

return Search\_By\_Chield(Btree->right, x);

}

}

}

return 0;

}

int Search\_By\_Pop(Tree\* Btree, int x) {

if (Btree != NULL) {

if (x == Btree->item) {

checkPop = true;

return Btree->item;

}

else {

checkPop = false;

if (x < Btree->item) {

return Search\_By\_Pop(Btree->left, x);

}

else if (x > Btree->item) {

return Search\_By\_Pop(Btree->right, x);

}

}

}

return 0;

}

void push(Tree\*& Btree, int x) {

if (x == Btree->item) {

cout << "Такой элемент уже существует!!!" << endl;

}

else if (x < Btree->item) {

if (Btree->left != NULL) {

push(Btree->left, x);

}

else {

Btree->left = new Tree;

Btree->left->parent = Btree;

Btree->left->left = Btree->left->right = NULL;

Btree->left->item = x;

}

}

else {

if (Btree->right != NULL) {

push(Btree->right, x);

}

else {

Btree->right = new Tree;

Btree->right->parent = Btree;

Btree->right->left = Btree->right->right = NULL;

Btree->right->item = x;

}

}

}

Tree\* pop(Tree\* Btree, int x) {

if (Btree == NULL) {

return Btree;

}

if (x == Btree->item) {

Tree\* tree = new Tree;

if (x == Btree->item && Btree->left == nullptr && Btree->right == nullptr) {

delete Btree;

Btree = NULL;

return 0;

} else if (x == Btree->item && Btree->left != nullptr && Btree->right == nullptr) {

tree = Btree->left;

tree->parent = Btree->parent;

} else if (x == Btree->item && Btree->right != nullptr && Btree->left == nullptr) {

tree = Btree->right;

tree->parent = Btree->parent;

} else if (x == Btree->item && Btree->right != nullptr && Btree->left != nullptr) {

if (Btree->item < root) {

tree = Btree->right;

tree->right->left = Btree->right->left;

tree->left = Btree->left;

tree->parent = Btree->parent;

tree->left->parent = tree;

} else {

tree = Btree->left;

tree->left->right = Btree->left->right;

tree->right = Btree->right;

tree->parent = Btree->parent;

tree->right->parent = tree;

}

}

delete Btree;

return tree;

} else if (x < Btree->item) {

Btree->left = pop(Btree->left, x);

} else if (x > Btree->item) {

Btree->right = pop(Btree->right, x);

}

return Btree;

}

**Самостоятельная работа**

1. Построить дерево из N (N>0) вершин, в котором каждая внутренняя вершина имеет только одну дочернюю вершину, причем если значение вершины является нечетным, то она имеет левую дочернюю вершину, а если четным, то правую. Каждой создаваемой вершине присвоить очередное значение из исходного набора данных.

**Текст программы**

#include <iostream>

#include <string>

#include <iomanip>

#include "Windows.h"

using namespace std;

struct Tree {

int item, level;

Tree\* left, \* right, \* parent;

};

bool \_count = true, checkSearch, checkPop;

void inition(Tree\*& Btree);

void add(Tree\*& Btree);

void output\_menu(Tree\*& Btree);

void del(Tree\*& Btree);

void Search(Tree\*& Btree);

int Count\_leaves(Tree\*& Btree);

void Count\_tops(Tree\*& Btree, int& count);

void delete\_tree(Tree\*& Btree);

void output\_inor(Tree\*& Btree);

void output\_preo(Tree\*& Btree);

void output\_post(Tree\*& Btree);

void output(Tree\*& Btree, int j);

void push(Tree\*& Btree, int x, int lev);

void pop(Tree\*& Btree, int x);

void POP(Tree\*& Btree);

int Search\_By\_Pop(Tree\* Btree, int x);

int Search\_By\_level(Tree\* Btree, int x);

int Search\_By\_El(Tree\* Btree, int x);

int Search\_By\_Parent(Tree\* Btree, int x);

int Search\_By\_Chield(Tree\* Btree, int x);

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Tree\* binary\_tree = NULL;

bool f = true;

int choice;

inition(binary\_tree);

while (f) {

int \_count\_tops = 0;

cout << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t\tГлавное Меню:\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t1. Добавление элемента\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t2. Вывод дерева\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t3. Удаление элемента\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t4. Поиск вершины дерева\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t5. Подсчет вершин дерева\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t6. Подсчет листьев дерева\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t7. Удалить дерево\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t8. Выход\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << endl;

cout << "Введите номер операции:";

cin >> choice;

system("cls");

switch (choice) {

case 1: add(binary\_tree); break;

case 2: output\_menu(binary\_tree); break;

case 3: del(binary\_tree); break;

case 4: Search(binary\_tree); break;

case 5: Count\_tops(binary\_tree, \_count\_tops); cout << "Кол - во вершин: " << \_count\_tops << endl; break;

case 6: cout << "Кол - во листьев: " << Count\_leaves(binary\_tree) << endl; break;

case 7: delete\_tree(binary\_tree); \_count = 0; break;

case 8: f = false; break;

default: cout << "Такой операции не существует!!!" << endl << "Введите заново." << endl; break;

}

}

return 0;

}

void inition(Tree\*& Btree) {

\_count = true;

int x;

cout << "Инициализируйте корень:";

cin >> x;

system("cls");

cout << "Элемент " << x << " инициализирован" << endl;

Btree = new Tree;

Btree->item = x;

Btree->level = 1;

Btree->left = Btree->right = Btree->parent = NULL;

}

void add(Tree\*& Btree) {

if (Btree == NULL) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

} else {

int x;

cout << "Введите новый элемент:";

cin >> x;

system("cls");

push(Btree, x, 2);

cout << "Элемент " << x << " добавлен в дерево" << endl;

}

}

void output\_menu(Tree\*& Btree) {

if (Btree == NULL) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

} else {

bool f = true;

int x;

while (f) {

cout << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t\tМеню вывода:\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t1. Inorder\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t2. Preorder\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t3. Postorder\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t4. В виде дерева\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t5. Назад\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << endl;

cout << "Введите номер операции:";

cin >> x;

system("cls");

switch (x) {

case 1: f = false; output\_inor(Btree); break;

case 2: f = false; output\_preo(Btree); break;

case 3: f = false; output\_post(Btree); break;

case 4: f = false; output(Btree, 0); break;

case 5: f = false; break;

default: cout << "Такой операции не существует!!!" << endl << "Введите заново." << endl; break;

}

}

}

}

void output\_inor(Tree\*& Btree) {

if (Btree != NULL) {

output\_inor(Btree->left);

cout << Btree->item << "\t";

output\_inor(Btree->right);

}

}

void output\_preo(Tree\*& Btree) {

if (Btree != NULL) {

cout << Btree->item << "\t";

output\_inor(Btree->left);

output\_inor(Btree->right);

}

}

void output\_post(Tree\*& Btree) {

if (Btree != NULL) {

output\_inor(Btree->left);

output\_inor(Btree->right);

cout << Btree->item << "\t";

}

}

void output(Tree\*& Btree, int j) {

if (Btree != NULL) {

output(Btree->right, j + 3);

for (int i = 0; i < j; i++) {

cout << "\t";

}

cout << "info=(" << Btree->item << ")-lvl(" << Btree->level << ")";

if (j != 0) {

cout << "-p(" << Btree->parent->item << ")";

} else {

cout << "-p(none)";

}

cout << endl;

output(Btree->left, j + 3);

}

}

void del(Tree\*& Btree) {

if (Btree == NULL) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

} else {

int x;

cout << "Введите удаляемый элемент:";

cin >> x;

system("cls");

Search\_By\_Pop(Btree, x);

if (checkPop == false) {

cout << "Элемента " << x << " не существует!!!" << endl;

} else {

cout << "Элемент " << x << " удален из дерева" << endl;

pop(Btree, x);

}

}

}

void Search(Tree\*& Btree) {

if (Btree == NULL) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

} else {

int x;

cout << "Введите искоемый элемент:";

cin >> x;

Search\_By\_El(Btree, x);

system("cls");

if (checkSearch == false) {

cout << "Элемента " << x << " не существует!!!" << endl;

} else {

cout << "Элемент " << Search\_By\_El(Btree, x) << " находиться на уровне: " << Search\_By\_level(Btree, x) << endl;

if (Search\_By\_level(Btree, x) == 1) {

cout << "Левый потомок: ";

if (Btree->left != nullptr) {

cout << Btree->left->item << "\t";

} else {

cout << "Отсустствует!!!\t";

}

cout << "Правый потомок: ";

if (Btree->right != nullptr) {

cout << Btree->right->item << endl;

} else {

cout << "Отсустствует!!!\n";

}

cout << "Его родитель: отсутствует!!!" << endl;

} else {

Search\_By\_Chield(Btree, x);

cout << "Его родитель: " << Search\_By\_Parent(Btree, x) << endl;

}

}

}

}

int Count\_leaves(Tree\*& Btree) {

if (\_count == false) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

} else {

int count;

if ((Btree->left == NULL) && (Btree->right == NULL)) {

count = 1;

} else {

count = 0;

}

if (Btree->left) {

count += Count\_leaves(Btree->left);

}

if (Btree->right) {

count += Count\_leaves(Btree->right);

}

return count;

}

return 0;

}

void Count\_tops(Tree\*& Btree, int& count) {

if (\_count == false) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

} else {

if (Btree != NULL) {

Count\_tops(Btree->left, count);

if (Btree->left != NULL || Btree->right != NULL) {

count++;

}

Count\_tops(Btree->right, count);

}

}

}

void delete\_tree(Tree\*& Btree) {

if (\_count == false) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

} else {

if (Btree != NULL) {

delete\_tree(Btree->left);

delete\_tree(Btree->right);

delete Btree;

Btree = NULL;

}

}

}

int Search\_By\_El(Tree\* Btree, int x) {

if (x == Btree->item) {

checkSearch = true;

return Btree->item;

} else {

checkSearch = false;

if ((Btree->item % 2) != 0) {

return Search\_By\_El(Btree->left, x);

} else if ((Btree->item % 2) == 0) {

return Search\_By\_El(Btree->right, x);

}

}

return 0;

}

int Search\_By\_level(Tree\* Btree, int x) {

if (x == Btree->item) {

return Btree->level;

}

if ((Btree->item % 2) != 0) {

return Search\_By\_level(Btree->left, x);

} else if ((Btree->item % 2) == 0) {

return Search\_By\_level(Btree->right, x);

}

return 0;

}

int Search\_By\_Parent(Tree\* Btree, int x) {

if (x == Btree->item) {

return Btree->parent->item;

}

if ((Btree->item % 2) != 0) {

return Search\_By\_Parent(Btree->left, x);

} else if ((Btree->item % 2) == 0) {

return Search\_By\_Parent(Btree->right, x);

}

return 0;

}

int Search\_By\_Chield(Tree\* Btree, int x) {

if (x == Btree->item) {

cout << "Левый потомок: ";

if (Btree->left != nullptr) {

cout << Btree->left->item << "\t";

} else {

cout << "Отсустствует!!!\t";

}

cout << "Правый потомок: ";

if (Btree->right != nullptr) {

cout << Btree->right->item << "\n";

} else {

cout << "Отсустствует!!!\n";

}

return Btree->item;

} else {

if ((Btree->item % 2) != 0) {

return Search\_By\_Chield(Btree->left, x);

} else if ((Btree->item % 2) == 0) {

return Search\_By\_Chield(Btree->right, x);

}

}

return 0;

}

int Search\_By\_Pop(Tree\* Btree, int x) {

if (x == Btree->item) {

checkPop = true;

return Btree->item;

} else {

checkPop = false;

if ((Btree->item % 2) != 0) {

return Search\_By\_Pop(Btree->left, x);

} else {

return Search\_By\_Pop(Btree->right, x);

}

}

return 0;

}

void push(Tree\*& Btree, int x, int lev) {

if (x == Btree->item) {

cout << "Такой элемент уже существует!!!" << endl;

} else if (Btree->left != NULL) {

push(Btree->left, x, lev + 1);

} else if (Btree->right != NULL) {

push(Btree->right, x, lev + 1);

} else if ((Btree->item % 2) != 0) {

Btree->left = new Tree;

Btree->left->parent = Btree;

Btree->left->left = Btree->left->right = NULL;

Btree->left->item = x;

Btree->left->level = lev;

} else if ((Btree->item % 2) == 0) {

Btree->right = new Tree;

Btree->right->parent = Btree;

Btree->right->left = Btree->right->right = NULL;

Btree->right->item = x;

Btree->right->level = lev;

}

}

void pop(Tree\*& Btree, int x) {

if (Btree == NULL) {

return;

} else {

if (x == Btree->item && Btree->left == nullptr && Btree->right == nullptr) {

POP(Btree);

}else if (x == Btree->item && Btree->left != nullptr && Btree->right == nullptr) {

Btree->left->parent = Btree->parent;

Btree->left->level = Btree->level;

Btree = Btree->left;

POP(Btree->left);

}else if (x == Btree->item && Btree->right != nullptr && Btree->left == nullptr) {

Btree->right->parent = Btree->parent;

Btree->right->level = Btree->level;

Btree = Btree->right;

POP(Btree->right);

}else if (x == Btree->item && Btree->right != nullptr && Btree->left != nullptr) {

Btree->right->parent = Btree->parent;

Btree->right->level = Btree->level;

Btree->right->left = Btree->left;

Btree->right->left->parent = Btree->right;

Btree->right->left->level = Btree->right->left->level;

Btree = Btree->right;

POP(Btree->right);

} else if ((Btree->item % 2) != 0) {

pop(Btree->left, x);

} else if ((Btree->item % 2) == 0) {

pop(Btree->right, x);

}

}

}

void POP(Tree\*& Btree) {

delete Btree;

Btree = NULL;

}

1. Построить дерево из N (N>0) вершин, со следующей структурой – если вершина дерева является внутренней, то в случае, если она имеет нечетное значение, то ее левая дочерняя вершина должна стать листом, а если имеет четное значение, тогда листом должна стать ее правая вершина. Для каждой внутренней вершины, сначала создать дочернюю вершину – лист, а затем внутреннюю дочернюю вершину (если данная вершина существует); каждой создаваемой вершине присвоить очередное значение из исходного набора данных.

**Текст программы**

#include <iostream>

#include <string>

#include <iomanip>

#include "Windows.h"

using namespace std;

struct Tree {

int item, level;

Tree\* left, \* right, \* parent;

};

bool \_count = true, checkSearch, checkPop;

void inition(Tree\*& Btree);

void add(Tree\*& Btree);

void output\_menu(Tree\*& Btree);

void del(Tree\*& Btree);

void Search(Tree\*& Btree);

int Count\_leaves(Tree\*& Btree);

void Count\_tops(Tree\*& Btree, int& count);

void delete\_tree(Tree\*& Btree);

void output\_inor(Tree\*& Btree);

void output\_preo(Tree\*& Btree);

void output\_post(Tree\*& Btree);

void output(Tree\*& Btree, int j);

void push(Tree\*& Btree, int x, int lev);

void pop(Tree\*& Btree, int x);

void POP(Tree\*& Btree);

int Search\_By\_Pop(Tree\* Btree, int x);

int Search\_By\_level(Tree\* Btree, int x);

int Search\_By\_El(Tree\* Btree, int x);

int Search\_By\_Parent(Tree\* Btree, int x);

int Search\_By\_Chield(Tree\* Btree, int x);

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Tree\* binary\_tree = NULL;

bool f = true;

int choice;

inition(binary\_tree);

while (f) {

int \_count\_tops = 0;

cout << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t\tГлавное Меню:\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t1. Добавление элемента\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t2. Вывод дерева\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t3. Удаление элемента\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t4. Поиск вершины дерева\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t5. Подсчет вершин дерева\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t6. Подсчет листьев дерева\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t7. Удалить дерево\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t8. Выход\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << endl;

cout << "Введите номер операции:";

cin >> choice;

system("cls");

switch (choice) {

case 1: add(binary\_tree); break;

case 2: output\_menu(binary\_tree); break;

case 3: del(binary\_tree); break;

case 4: Search(binary\_tree); break;

case 5: Count\_tops(binary\_tree, \_count\_tops); cout << "Кол - во вершин: " << \_count\_tops << endl; break;

case 6: cout << "Кол - во листьев: " << Count\_leaves(binary\_tree) << endl; break;

case 7: delete\_tree(binary\_tree); \_count = 0; break;

case 8: f = false; break;

default: cout << "Такой операции не существует!!!" << endl << "Введите заново." << endl; break;

}

}

return 0;

}

void inition(Tree\*& Btree) {

\_count = true;

int x;

cout << "Инициализируйте корень:";

cin >> x;

system("cls");

cout << "Элемент " << x << " инициализирован" << endl;

Btree = new Tree;

Btree->item = x;

Btree->level = 1;

Btree->left = Btree->right = Btree->parent = NULL;

}

void add(Tree\*& Btree) {

if (Btree == NULL) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

}

else {

int x;

cout << "Введите новый элемент:";

cin >> x;

system("cls");

push(Btree, x, 2);

cout << "Элемент " << x << " добавлен в дерево" << endl;

}

}

void output\_menu(Tree\*& Btree) {

if (Btree == NULL) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

}

else {

bool f = true;

int x;

while (f) {

cout << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t\tМеню вывода:\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t1. Inorder\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t2. Preorder\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t3. Postorder\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t4. В виде дерева\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << "|\t5. Назад\t\t\t|" << endl;

cout << "-----------------------------------------" << endl;

cout << endl;

cout << "Введите номер операции:";

cin >> x;

system("cls");

switch (x) {

case 1: f = false; output\_inor(Btree); break;

case 2: f = false; output\_preo(Btree); break;

case 3: f = false; output\_post(Btree); break;

case 4: f = false; output(Btree, 0); break;

case 5: f = false; break;

default: cout << "Такой операции не существует!!!" << endl << "Введите заново." << endl; break;

}

}

}

}

void output\_inor(Tree\*& Btree) {

if (Btree != NULL) {

output\_inor(Btree->left);

cout << Btree->item << "\t";

output\_inor(Btree->right);

}

}

void output\_preo(Tree\*& Btree) {

if (Btree != NULL) {

cout << Btree->item << "\t";

output\_inor(Btree->left);

output\_inor(Btree->right);

}

}

void output\_post(Tree\*& Btree) {

if (Btree != NULL) {

output\_inor(Btree->left);

output\_inor(Btree->right);

cout << Btree->item << "\t";

}

}

void output(Tree\*& Btree, int j) {

if (Btree == nullptr) {

return;

}

else {

output(Btree->right, j + 3);

for (int i = 0; i < j; i++) {

cout << "\t";

}

cout << "info=(";

if (Btree->item == -9999) {

cout << "list)-lvl(" << Btree->level << ")";

}

else {

cout << Btree->item << ")-lvl(" << Btree->level << ")";

}

if (j != 0) {

cout << "-p(" << Btree->parent->item << ")";

}

else {

cout << "-p(none)";

}

cout << endl;

output(Btree->left, j + 3);

}

}

void del(Tree\*& Btree) {

if (Btree == NULL) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

}

else {

int x;

cout << "Введите удаляемый элемент:";

cin >> x;

system("cls");

Search\_By\_Pop(Btree, x);

if (checkPop == false) {

cout << "Элемента " << x << " не существует!!!" << endl;

}

else {

cout << "Элемент " << x << " удален из дерева" << endl;

pop(Btree, x);

}

}

}

void Search(Tree\*& Btree) {

if (Btree == NULL) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

}

else {

int x;

cout << "Введите искоемый элемент:";

cin >> x;

Search\_By\_El(Btree, x);

system("cls");

if (checkSearch == false) {

cout << "Элемента " << x << " не существует!!!" << endl;

}

else {

cout << "Элемент " << Search\_By\_El(Btree, x) << " находиться на уровне: " << Search\_By\_level(Btree, x) << endl;

if (Search\_By\_level(Btree, x) == 1) {

cout << "Левый потомок: ";

if (Btree->left != nullptr) {

cout << Btree->left->item << "\t";

}

else {

cout << "Отсустствует!!!\t";

}

cout << "Правый потомок: ";

if (Btree->right != nullptr) {

cout << Btree->right->item << endl;

}

else {

cout << "Отсустствует!!!\n";

}

cout << "Его родитель: отсутствует!!!" << endl;

}

else {

Search\_By\_Chield(Btree, x);

cout << "Его родитель: " << Search\_By\_Parent(Btree, x) << endl;

}

}

}

}

int Count\_leaves(Tree\*& Btree) {

if (\_count == false) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

}

else {

int count;

if ((Btree->left == NULL) && (Btree->right == NULL)) {

count = 1;

}

else {

count = 0;

}

if (Btree->left) {

count += Count\_leaves(Btree->left);

}

if (Btree->right) {

count += Count\_leaves(Btree->right);

}

return count;

}

return 0;

}

void Count\_tops(Tree\*& Btree, int& count) {

if (\_count == false) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

}

else {

if (Btree != NULL) {

Count\_tops(Btree->left, count);

if (Btree->left != NULL || Btree->right != NULL) {

count++;

}

Count\_tops(Btree->right, count);

}

}

}

void delete\_tree(Tree\*& Btree) {

if (\_count == false) {

cout << "Дерева не существует" << endl << "ИНИЦИАЛИЗИРУЕМ...\n" << endl;

inition(Btree);

}

else {

if (Btree != NULL) {

delete\_tree(Btree->left);

delete\_tree(Btree->right);

delete Btree;

Btree = NULL;

}

}

}

int Search\_By\_El(Tree\* Btree, int x) {

if (Btree != NULL) {

if (x == Btree->item) {

checkSearch = true;

return Btree->item;

}

else {

checkSearch = false;

if ((Btree->item % 2) == 0) {

return Search\_By\_El(Btree->left, x);

}

else if ((Btree->item % 2) != 0) {

return Search\_By\_El(Btree->right, x);

}

}

}

return 0;

}

int Search\_By\_level(Tree\* Btree, int x) {

if (x == Btree->item) {

return Btree->level;

}

if ((Btree->item % 2) == 0) {

return Search\_By\_level(Btree->left, x);

}

else if ((Btree->item % 2) != 0) {

return Search\_By\_level(Btree->right, x);

}

return 0;

}

int Search\_By\_Parent(Tree\* Btree, int x) {

if (x == Btree->item) {

return Btree->parent->item;

}

if ((Btree->item % 2) == 0) {

return Search\_By\_Parent(Btree->left, x);

}

else if ((Btree->item % 2) != 0) {

return Search\_By\_Parent(Btree->right, x);

}

return 0;

}

int Search\_By\_Chield(Tree\* Btree, int x) {

if (Btree != NULL) {

if (x == Btree->item) {

cout << "Левый потомок: ";

if (Btree->left != nullptr) {

cout << Btree->left->item << "\t";

}

else {

cout << "Отсустствует!!!\t";

}

cout << "Правый потомок: ";

if (Btree->right != nullptr) {

cout << Btree->right->item << "\n";

}

else {

cout << "Отсустствует!!!\n";

}

return Btree->item;

}

else {

if ((Btree->item % 2) == 0) {

return Search\_By\_Chield(Btree->left, x);

}

else if ((Btree->item % 2) != 0) {

return Search\_By\_Chield(Btree->right, x);

}

}

}

return 0;

}

int Search\_By\_Pop(Tree\* Btree, int x) {

if (Btree != NULL) {

if (x == Btree->item) {

checkPop = true;

return Btree->item;

}

else {

checkPop = false;

if ((Btree->item % 2) == 0) {

return Search\_By\_Pop(Btree->left, x);

}

else if ((Btree->item % 2) != 0) {

return Search\_By\_Pop(Btree->right, x);

}

}

}

return 0;

}

void push(Tree\*& Btree, int x, int lev) {

if (x == Btree->item) {

cout << "Такой элемент уже существует!!!" << endl;

}

else if ((Btree->item % 2) == 0) {

if (Btree->left != NULL) {

push(Btree->left, x, lev + 1);

}

else {

Btree->left = new Tree;

Btree->left->parent = Btree;

Btree->left->left = Btree->left->right = NULL;

Btree->left->item = x;

Btree->left->level = lev;

Btree->right = new Tree;

Btree->right->item = -9999;

Btree->right->parent = Btree;

Btree->right->level = lev;

Btree->right->left = Btree->right->right = NULL;

}

}

else {

if (Btree->right != NULL) {

push(Btree->right, x, lev + 1);

}

else {

Btree->right = new Tree;

Btree->right->parent = Btree;

Btree->right->left = Btree->right->right = NULL;

Btree->right->item = x;

Btree->right->level = lev;

Btree->left = new Tree;

Btree->left->item = -9999;

Btree->left->parent = Btree;

Btree->left->level = lev;

Btree->left->left = Btree->left->right = NULL;

}

}

}

void pop(Tree\*& Btree, int x) {

if (Btree == NULL) {

return;

}

else {

if (x == Btree->item && Btree->left == nullptr && Btree->right == nullptr) {

POP(Btree);

}

else if (x == Btree->item && Btree->left != nullptr && Btree->right == nullptr) {

Btree->left->parent = Btree->parent;

Btree->left->level = Btree->level;

Btree = Btree->left;

POP(Btree->left);

}

else if (x == Btree->item && Btree->right != nullptr && Btree->left == nullptr) {

Btree->right->parent = Btree->parent;

Btree->right->level = Btree->level;

Btree = Btree->right;

POP(Btree->right);

}

else if (x == Btree->item && Btree->right != nullptr && Btree->left != nullptr) {

Btree->right->parent = Btree->parent;

Btree->right->level = Btree->level;

Btree->right->left = Btree->left;

Btree->right->left->parent = Btree->right;

Btree->right->left->level = Btree->right->left->level;

Btree = Btree->right;

POP(Btree->right);

}

else if (x < Btree->item) {

pop(Btree->left, x);

}

else if (x > Btree->item) {

pop(Btree->right, x);

}

}

}

void POP(Tree\*& Btree) {

delete Btree;

Btree = NULL;

}