Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №6

По дисциплине «Основы алгоритмизации и программирования»

За 2 семестр

Тема: «Динамические структуры. Деревья»

Выполнили:

Студенты 1 курса

Группы ИИ-21(1)

Кирилович А. А.

Проверил:

Гирель Т. Н.

Брест 2022

**Цель работы:** приобретение навыков работы с динамической памятью и указателями на C++. Изучение принципов работы с динамическими структурами данных: деревьями.

**Ход работы:**

**Вариант 3**

**Задание 1**

Напишите программу, содержащую процедуру, которая каждый отрицательный элемент дерева заменяет на положительный, а положительный превращает в ноль.

**Код:**

#include <iostream>

#include <string>

#include <stdlib.h>

using namespace std;

bool is\_left;

int tabs = 0;

struct Node {

    int val;

    Node\* left;

    Node\* right;

    Node(int \_val) : val(\_val), left(nullptr), right(nullptr) {}

};

struct tree {

    Node\* root;

    tree() : root(nullptr) {}

    bool is\_empty() {

        return root == nullptr;

    }

    void push\_back(int \_val, Node\* minor\_root) {

        if (is\_empty()) {

            Node\* p = new Node(\_val);

            root = p;

            return;

        }

        Node\* q = minor\_root;

        if (q->left == nullptr) {

            Node\* p = new Node(\_val);

            q->left = p;

            is\_left = true;

            return;

        }

        if (q->right == nullptr) {

            Node\* p = new Node(\_val);

            q->right = p;

            is\_left = false;

            return;

        }

        if (is\_left) {

            push\_back(\_val, q->right);

        }

        else {

            push\_back(\_val, q->left);

        }

    }

    void print(Node\* minor\_root) {

        if (!minor\_root) return;

        tabs++;

        print(minor\_root->left);

        for (int i = 0; i < tabs; i++) cout << " ";

        cout << minor\_root->val << endl;

        print(minor\_root->right);

        tabs--;

        return;

    }

    void task1(Node\* minor\_root) {

        if (!minor\_root) return;

        task1(minor\_root->left);

        if (minor\_root->val < 0) {

            minor\_root->val = abs(minor\_root->val);

        }

        else {

            minor\_root->val = 0;

        }

        task1(minor\_root->right);

        return;

    }

    void free\_tree(Node\* minor\_root) {

        if (!minor\_root) return;

        free\_tree(minor\_root->left);

        free\_tree(minor\_root->right);

        delete minor\_root;

        return;

    }

};

int main() {

    tree tr;

    string data;

    cout << "Enter the values you need to write to the list. To stop recording write 'q':" << endl;

    while (1) {

        cin >> data;

        if (data == "q") {

            break;

        }

        try {

           tr.push\_back(atoi(data.c\_str()), tr.root);

        }

        catch(const std::exception& e) {

            std::cerr << e.what() << '\n';

        }

    }

    cout << "\n\n";

    tr.print(tr.root);

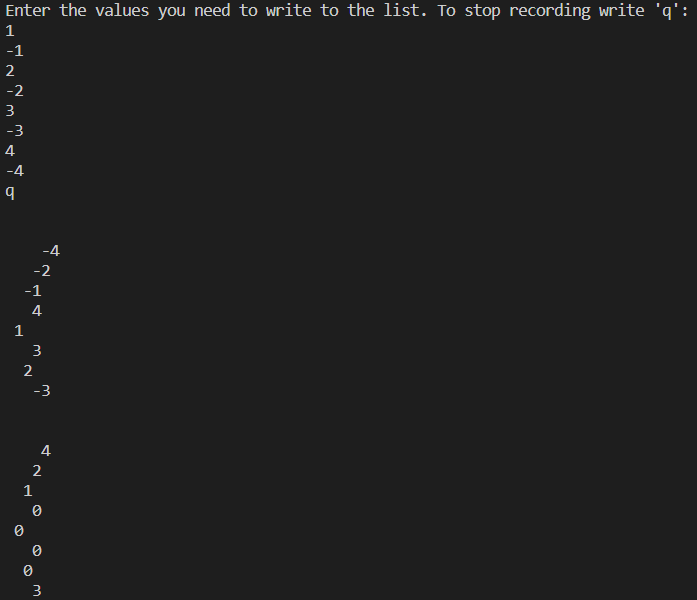
    tr.task1(tr.root);

    cout << "\n\n";

    tr.print(tr.root);

    tr.free\_tree(tr.root);

}



**Задание 2**

Реализовать операцию определения уровня узла с заданным указателем в бинарном дереве, построить дерево с узлами — символами и определить минимальный и максимальный уровни листа.

**Код:**

#include <iostream>

#include <string>

#include <stdlib.h>

using namespace std;

bool is\_left;

int tabs = 0;

int MAX = 0;

int MIN = 100;

string s = "Way: ";

struct Node {

    int val;

    Node\* left;

    Node\* right;

    Node(int \_val) : val(\_val), left(nullptr), right(nullptr) {}

};

struct tree {

    Node\* root;

    tree() : root(nullptr) {}

    bool is\_empty() {

        return root == nullptr;

    }

    void push\_back(int \_val, Node\* minor\_root) {

        if (is\_empty()) {

            Node\* p = new Node(\_val);

            root = p;

            return;

        }

        Node\* q = minor\_root;

        if (\_val < q->val) {

            if (q->left == nullptr) {

                Node\* p = new Node(\_val);

                q->left = p;

                return;

            }

            q = q->left;

        }

        else {

            if (q->right == nullptr) {

                Node\* p = new Node(\_val);

                q->right = p;

                return;

            }

            q = q->right;

        }

        push\_back(\_val, q);

    }

    void search(int \_val, Node\* minor\_root) {

        if (is\_empty()) {

            cout << "Tree is empty.";

            return;

        }

        if (!minor\_root) {

            cout << "This element doesn't exist.";

            return;

        }

        Node\* q = minor\_root;

        if (\_val < q->val) {

            q = q->left;

            s += "Left. ";

        }

        else if (\_val > q->val){

            q = q->right;

            s += "Right. ";

        }

        else {

            s += "There.";

            cout << s;

            return;

        }

        search(\_val, q);

    }

    void print(Node\* minor\_root) {

        if (!minor\_root) return;

        tabs++;

        print(minor\_root->left);

        for (int i = 0; i < tabs; i++) cout << " ";

        cout << minor\_root->val << endl;

        print(minor\_root->right);

        tabs--;

        return;

    }

    void depth(Node\* minor\_root, int k) {

        if (!minor\_root) {

            if (k < MIN) MIN = k - 1;

            return;

        }

        depth(minor\_root->left, k + 1);

        if (k > MAX) MAX = k;

        depth(minor\_root->right, k + 1);

        return;

    }

    void free\_tree(Node\* minor\_root) {

        if (!minor\_root) return;

        free\_tree(minor\_root->left);

        free\_tree(minor\_root->right);

        delete minor\_root;

        return;

    }

};

int main() {

    tree tr;

    string data;

    cout << "Enter the values you need to write to the list. To stop recording write 'q':" << endl;

    while (1) {

        cin >> data;

        if (data == "q") {

            break;

        }

        try {

           tr.push\_back(atoi(data.c\_str()), tr.root);

        }

        catch(const std::exception& e) {

            std::cerr << e.what() << '\n';

        }

    }

    cout << "\n\n";

    tr.print(tr.root);

    cout << "\n\n";

    cout << "Enter the values you need to search:" << endl;

    cin >> data;

    tr.search(atoi(data.c\_str()), tr.root);

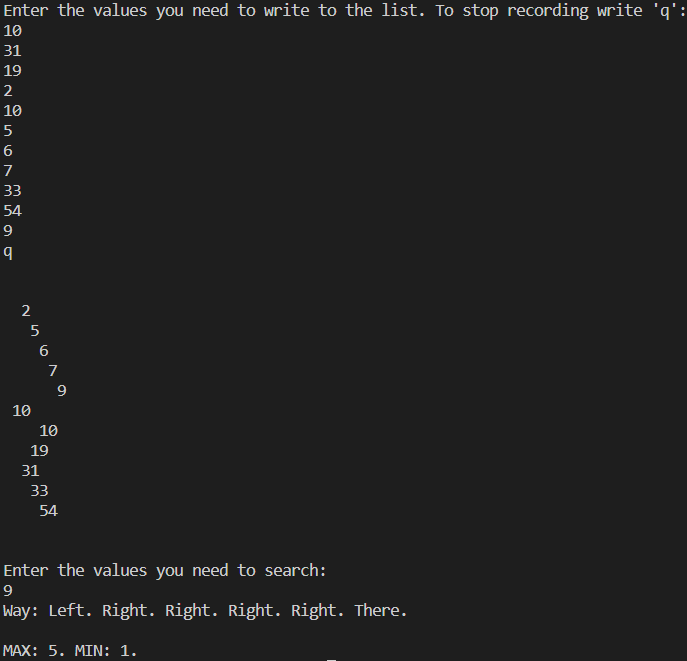
    cout << "\n\n";

    tr.depth(tr.root, 0);

    cout << "MAX: " << MAX << ".\tMIN: " << MIN << ".";

    tr.free\_tree(tr.root);

}



**Вывод:** приобрел навыки работы с динамической памятью и указателями на C. Изучил принципы работы с динамическими структурами данных: деревьями.