1. **Цель работы**

   Целью работы является изучение деревьев поиска и получение практических навыков их использования.

2. **Задание**

Согласно варианту №18:

Циклично удалять каждый нечѐтный узел дерева (Корень – 1, нечетный) и перестраивать дерево, беря элементы по порядку обхода. Данную процедуру выполнять, пока не останется последний элемент. На каждой итерации выводить дерево

Метод обхода – обратный.

3. **Листинг программы**

```
// Вариант 18
//

#include <iostream>
#include <math.h>
using namespace std;

struct Tree
{
    int key;
    int h = 1;
    Tree* left;
    Tree* right;
};

int get_num_int() // Запрос и проверка числа на корректность
{
    int x;

    cin >> x;
    while (cin.fail() || (cin.peek() != '\n')) // Проверка на корректность
    {
        cin.clear(); // Очищение флага ошибки
        cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Очистка буфера
запроса
        cout << "Повторите ввод: ";
        cin >> x;
    }

    return x;
}

int getHeight(Tree* Root)
{
    if (Root == NULL)
        return 0;
    else
        return Root->h;
}

int getHeightDiff(Tree* Root) // right - left
{
    return getHeight(Root->right) - getHeight(Root->left);
}
```

```
void BalanceHeight(Tree** Root)
{
    (*Root)->h = max(getHeight((*Root)->left), getHeight((*Root)->right)) + 1;
}

void RightRotate(Tree** Root)
{
    Tree* left = (*Root)->left;
    (*Root)->left = left->right;
    left->right = (*Root);
    BalanceHeight(Root);
    BalanceHeight(&left);
    *Root = left;
}

void LeftRotate(Tree** Root)
{
    Tree* right = (*Root)->right;
    (*Root)->right = right->left;
    right->left = (*Root);
    BalanceHeight(Root);
    BalanceHeight(&right);
    *Root = right;
}

void Balance(Tree** Root)
{
    BalanceHeight(Root);
    if (getHeightDiff(*Root) == 2)
    {
        if (getHeightDiff((*Root)->right) < 0 )
            RightRotate(&(*Root)->right);
        LeftRotate(Root);
    }
    else if (getHeightDiff(*Root) == -2)
    {
        if (getHeightDiff((*Root)->left) > 0)
            LeftRotate(&(*Root)->left);
        RightRotate(Root);
    }

}


void ShowTree(Tree* Root, int level)
{
    if (Root != NULL) {
        ShowTree(Root->right, level + 1);
        for (int i = 1; i <= level; i++) cout << "    ";
        cout << Root->key << endl;
        ShowTree(Root->left, level + 1);
    }
}

void AddElem(Tree** Root, int value)
{
    if (*Root == NULL)
    {
        Tree* R = new Tree;
        R->key = value;
        R->left = NULL;
        R->right = NULL;
        *Root = R;
        return;
```

```
    }
    if (value < (*Root)->key)
    {
        AddElem(&((*Root)->left), value);
    }
    else if (value > (*Root)->key)
    {
        AddElem(&((*Root)->right), value);
    }

    BalanceHeight(Root);

}

Tree* SearchMin(Tree* Root)
{
    Tree* cur = (Root);
    while (cur->left)
    {
        cur = cur->left;
    }
    Tree* key = cur;
    return key;
}

bool DeleteElem(Tree** Root, int value)
{
    if (Root == NULL) return false;
    if (value < (*Root)->key)
    {
        DeleteElem(&((*Root)->left), value);
    }
    else if (value > (*Root)->key)
    {
        DeleteElem(&((*Root)->right), value);
    }
    else
    {
        if ((*Root)->right == (*Root)->left)
        {
            delete* Root;
            *Root = NULL;
            return true;
        }
        else if ((*Root)->right && not((*Root)->left))
        {
            Tree* right = (*Root)->right;
            *Root = right;
            return true;
        }
        else if ((*Root)->left && not((*Root)->right))
        {
            Tree* left = (*Root)->left;
            *Root = left;
            return true;
        }
        else
        {
            Tree* listmin = SearchMin((*Root)->right);

            if (not((*Root)->right->left))
            {
                (*Root)->key = listmin->key;
                (*Root)->right = listmin->right;
                return true;
```

```
            }
            else
            {
                int key = listmin->key;
                DeleteElem(Root, key);
                (*Root)->key = key;
                return true;
            }
        }

    }
    BalanceHeight(Root);
}


Tree* SearchElem(Tree* Root, int value, int k)
{
    if (Root == NULL) return Root;
    if (value == Root->key) {
        cout << "Кол-во шагов: " << k++ << endl;
        return Root;
    }
    k++;
    if (value < Root->key)
    {
        if (Root->left == NULL) return NULL;
        SearchElem(Root->left, value, k);
    }
    else if (value > Root->key)
    {
        if (Root->right == NULL) return NULL;
        SearchElem(Root->right, value, k);
    }
}

void ShowBack(Tree* Root)
{
    if (Root == NULL) return;
    ShowBack(Root->left);
    ShowBack(Root->right);
    cout << Root->key << " ";
}

void DeleteOdd(Tree** Root, int& k)
{
    if (Root == NULL) return;


    if ((*Root)->left != NULL)    DeleteOdd(&((*Root)->left), k);;
    if ((*Root)->right != NULL)    DeleteOdd(&((*Root)->right), k);;
    k++;
    if (k % 2 != 0)
    {
        DeleteElem(Root, (*Root)->key);
    }
}

void CycleDelete(Tree ** Root)
{
    int t = 1;

    cout << "Изначальное дерево: "<< endl;
    ShowTree(*Root, 0);
    while ((*Root)->right != (*Root)->left)
    {
```

```
            int k = 0;
            cout << "Итерация№ " << t++ << endl;
            DeleteOdd(Root, k);
            Balance(Root);
            ShowTree(*Root, 0);
        }
    }

    int main()
    {
        setlocale(LC_ALL, "RUS");
        Tree* Root = NULL;

        cout << "Добро пожаловать в программу АВЛ деревьев поиска." << endl;
        /*
        AddElem(&Root, 100);
        Balance(&Root);
        AddElem(&Root, 200);
        Balance(&Root);
        AddElem(&Root, 70);
        Balance(&Root);
        AddElem(&Root, 69);
        Balance(&Root);
        AddElem(&Root, 300);
        Balance(&Root);
        AddElem(&Root, 80);
        Balance(&Root);
        AddElem(&Root, 79);
        Balance(&Root);
        AddElem(&Root, 90);
        Balance(&Root);
        */
        while (true)
        {

            cout << "1. Добавить элемент" << endl;
            cout << "2. Удалить элемент" << endl;
            cout << "3. Найти элемент" << endl;
            cout << "4. Вывод дерева" << endl;
            cout << "5. Обход вершин" << endl;
            cout << "6. Удаление каждого нечётного узла" << endl;
            cout << "7. Выход" << endl;

            cout << "Выберите дальнейшее действие: ";
            int choice = get_num_int();

            if (choice == 1)
            {
                system("cls");
                cout << "Введите элемент для добавления: ";
                int value = get_num_int();
                if (SearchElem(Root, value, 0) != NULL)
                {
                    cout << "Элемент уже существует" << endl;
                }
                else
                {
                    AddElem(&Root, value);
                    Balance(&Root);
                }

            }
            else if (choice == 2)
            {
                system("cls");
```

```
            cout << "Введите элемент для удаления: ";
            int value = get_num_int();
            if (SearchElem(Root, value, 0) == NULL)
            {
                cout << "Элемента не существует" << endl;
            }
            else
            {
                DeleteElem(&Root, value);

                if (Root == NULL) {
                    cout << "Удалено всё дерево\n"; break;
                }
                Balance(&Root);
            }
        }
        else if (choice == 3)
        {
            system("cls");
            cout << "Введите элемент для поиска: ";
            int value = get_num_int();

            if (SearchElem(Root, value, 0) != NULL)
            {
                cout << "Элемент найден" << endl;
            }
            else
            {
                cout << "Элемент не найден" << endl;
            }
        }
        else if (choice == 4)
        {
            system("cls");
            cout << "Дерево: " << endl;

            ShowTree(Root, 0);
        }
        else if (choice == 5)
        {
            system("cls");
            cout << "Список, соответствующий обратному обходу: " << endl;
            ShowBack(Root);
            cout << endl;
        }
        else if (choice == 6)
        {
            system("cls");
            cout << "Начинаем обратный обход.." << endl;
            CycleDelete(&Root);
        }
        else if (choice == 7)
        {
            system("cls");
            exit(0);
        }
        else
        {
            system("cls");
        }
    }
}
```

4. Выводы

В процессе лабораторной работы были изучены деревья поиска и способы их практической реализации.