МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Структуры и алгоритмы обработки данных

Отчет по лабораторной работе №1

Бинарные деревья поиска

Выполнили студенты группы M3О-211Б-21

Плоцкий Б.А.

Раужев Ю.М.

Москва 2022 г.

Оглавление

[Задание 3](#_Toc121406728)

[Код программы 4](#_Toc121406729)

[Тестирование программы 19](#_Toc121406730)

[Тест 1 19](#_Toc121406731)

[Вывод 21](#_Toc121406732)

# Задание

1. Реализовать функции вставки, поиска, удаления узла, обхода дерева, вывода дерева на экран, нахождения высоты дерева и количества узлов.
2. Реализовать дополнительно функцию в соответствии с вариантом: *T* – тип ключей, *D* – диапазон изменения значений ключей.

|  |  |  |  |
| --- | --- | --- | --- |
| 110 | **char** | [а..я, А..Я] | Определить число узлов в левом и правом поддеревьях |

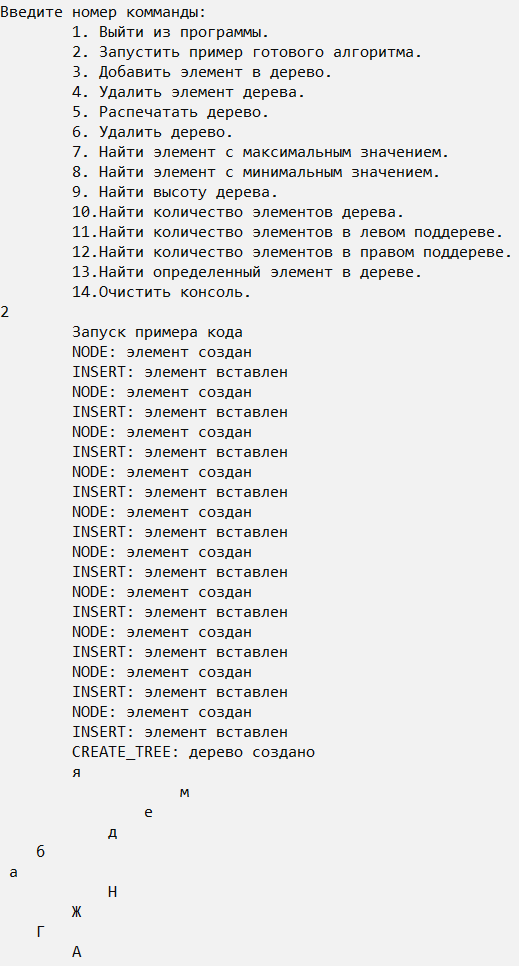
# Код программы

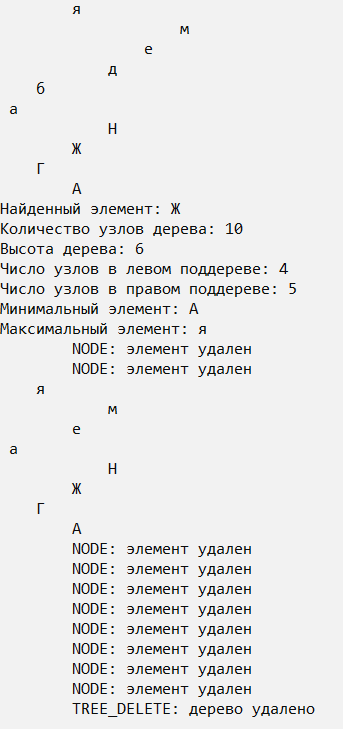
*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\  
\*                    кафедра № 304 2 курс 3 семестр информатика         \*  
\*-----------------------------------------------------------------------\*  
\*   Project type : solution                                             \*  
\*   Project name : LW1                                                  \*  
\*   File name    : main.cpp                                             \*  
\*   Language     : c/c++                                                \*  
\*   Programmers  : Плоцкий Б.А. Раужев Ю. М.                            \*  
\*   Created      : 12/11/22                                             \*  
\*   Last revision: 16/11/22                                             \*  
\*   Comment(s)   :                                                      \*  
\*                                                                       \*  
\*   Лабораторная работа «Бинарные деревья поиска»                       \*  
\*   Задание                                                             \*  
\*   1.  Реализовать функции вставки, поиска, удаления узла,             \*  
\*   обхода дерева, вывода дерева на экран, нахождения высоты            \*  
\*   дерева и количества узлов.                                          \*  
\*   2.  Реализовать дополнительно функцию в соответствии с вариантом:   \*  
\*   T – тип ключей, D – диапазон изменения значений ключей.             \*  
\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
#include <iostream>  
#include <iomanip>  
#include <Windows.h> // для считывания кириллицы****using******namespace****std;  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                      К О Н С Т А Н Т Ы                        \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// если нужна печать по каждому действию  
#define NEED\_PRINT\_DEBUG 1  
   
// вывод в консоль сообщения  
#define INFO(str) if(NEED\_PRINT\_DEBUG)cout<<"\t"<<str<<"\n";  
   
// множитель ширины вывода дерева  
#define WIDTH\_MULT\_PRINT 4  
   
// коды для взаимодействия пользователья с программой****enum******class****input\_codes  
{  
    exit = 0,  
    template\_program,  
    tree\_insert,  
    tree\_delete\_node,  
    tree\_print,  
    tree\_delete,  
    tree\_find\_max,  
    tree\_find\_min,  
    tree\_count\_height,  
    tree\_count\_nodes,  
    tree\_count\_nodes\_in\_left\_subtree,  
    tree\_count\_nodes\_in\_right\_subtree,  
    tree\_find\_node,  
    clear\_console  
};  
   
// строка с коммандами****const******char****\* command\_str =  
"\nВведите номер комманды:\n\  
\t1. Выйти из программы.\n\  
\t2. Запустить пример готового алгоритма.\n\  
\t3. Добавить элемент в дерево.\n\  
\t4. Удалить элемент дерева.\n\  
\t5. Распечатать дерево.\n\  
\t6. Удалить дерево.\n\  
\t7. Найти элемент с максимальным значением.\n\  
\t8. Найти элемент с минимальным значением.\n\  
\t9. Найти высоту дерева.\n\  
\t10.Найти количество элементов дерева.\n\  
\t11.Найти количество элементов в левом поддереве.\n\  
\t12.Найти количество элементов в правом поддереве.\n\  
\t13.Найти определенный элемент в дереве.\n\  
\t14.Очистить консоль.";  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                              N O D E                          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// элемент дерева  
template<****class****T>****struct****node  
{  
    T m\_data;           // ключ  
    node\* m\_left;       // указатель на левого потомка  
    node\* m\_right;      // указатель на правого потомка  
   
    // конструкторы  
    node() :m\_left(nullptr), m\_right(nullptr), m\_data(NULL)  
    {  
        INFO("NODE: элемент создан");  
    };  
   
    node(T \_data)  
        :m\_left(nullptr), m\_right(nullptr), m\_data(\_data)  
    {  
        INFO("NODE: элемент создан");  
    };  
   
    // деструктор  
    ~node()  
    {  
        m\_data = T(NULL);  
        m\_left = nullptr;  
        m\_right = nullptr;  
        INFO("NODE: элемент удален");  
    }  
};  
   
// удаление элемента дерева  
template<****class****T>****void****node\_delete(node<T>\* \_node);  
   
// печать элемента  
template<****class****T>****void****node\_print(node<T>\* \_node);  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                              T R E E                          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
// обход дерева симметричный  
template<****class****T>****void****inorder(node<T>\* \_root,****void****(\*\_func)(node<T>\*));  
   
// обход дерева прямой  
template<****class****T>****void****preorder(node<T>\* \_root,****void****(\*\_func)(node<T>\*));  
   
// обход дерева обратный  
template<****class****T>****void****postorder(node<T>\* \_root,****void****(\*\_func)(node<T>\*));  
   
// создание дерева из массива  
template<****class****T>  
node<T>\* tree\_create(T\* arr,****int****size);  
   
// удаление дерева  
template<****class****T>****void****tree\_delete(node<T>\*& \_root);  
   
// печать дерева  
template<****class****T>****void****tree\_print(node<T>\* \_root,****int****\_lvl = 0);  
   
// вставка элемента в дерево  
template<****class****T>  
node<T>\* tree\_node\_insert(node<T>\* \_root, T \_key);  
   
// поиск узла  
template<****class****T>  
node<T>\* tree\_find\_node(node<T>\* \_root, T \_key);  
   
// нахождение высоты дерева  
template<****class****T>****int****tree\_count\_height(node<T>\* \_root);  
   
// нахождение количества узлов  
template<****class****T>****int****tree\_count\_nodes(node<T>\* \_root);  
   
// нахождение минимального элемента в дереве  
template<****class****T>  
node<T>\* tree\_find\_min(node<T>\* \_root);  
   
// нахождение максимального элемента в дереве  
template<****class****T>  
node<T>\* tree\_find\_max(node<T>\* \_root);  
   
// удаление элемента дерева  
template<****class****T>  
node<T>\* tree\_node\_delete(node<T>\* \_root, T \_key);  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*        В С П О М О Г А Т Е Л Ь Н Ы Е   Ф У Н К Ц И И          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// проверка на номер команды****bool****check\_number(****int****\_num);  
   
// проверка на корректность введенного символа****bool****check\_cyrillic\_symbol(****char****\_symb);  
   
// пример готовой программы****void****example\_program();  
   
// ввод и проверка значений  
template<typename T>  
T input\_and\_check(****bool****(\*\_comp)(T),****const******char****\* welcome\_str,****const******char****\* err\_str  
    = "Было введено некорректное значение");  
   
// функция ведения диалога с пользователем****void****dialog();  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                Г Л А В Н А Я   Ф У Н К Ц И Я                  \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/****int****main()  
{  
    // для считывания кириллицы  
    SetConsoleCP(1251);  
    SetConsoleOutputCP(1251);  
   
    // запуск диалога  
    dialog();****return****0;  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*              Р Е А Л И З А Ц И Я   Ф У Н К Ц И Й              \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                              N O D E                          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// удаление элемента дерева  
template<****class****T>****void****node\_delete(node<T>\* \_node)  
{  
    delete \_node;  
}  
   
// печать элемента  
template<****class****T>****void****node\_print(node<T>\* \_node)  
{  
    cout << \_node->m\_data << " ";  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                              T R E E                          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                              T R E E                          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
// обход дерева симметричный  
template<****class****T>****void****inorder(node<T>\* \_root,****void****(\*\_func)(node<T>\*)) {****if****(\_root)  
    {  
        // идем налево  
        inorder(\_root->m\_left, \_func);  
   
        // обрабатываем элемент  
        \_func(\_root);  
   
        // идем направо  
        inorder(\_root->m\_right, \_func);  
    }  
}  
   
// обход дерева прямой  
template<****class****T>****void****preorder(node<T>\* \_root,****void****(\*\_func)(node<T>\*))  
{****if****(\_root)  
    {  
        // обрабатываем элемент  
        \_func(\_root);  
   
        // идем налево  
        preorder(\_root->m\_left, \_func);  
   
        // идем направо  
        preorder(\_root->m\_right, \_func);  
    }  
}  
   
// обход дерева обратный  
template<****class****T>****void****postorder(node<T>\* \_root,****void****(\*\_func)(node<T>\*))  
{****if****(\_root)  
    {  
        // идем налево  
        postorder(\_root->m\_left, \_func);  
   
        // идем направо  
        postorder(\_root->m\_right, \_func);  
   
        // обрабатываем элемент  
        \_func(\_root);  
    }  
}  
   
// создание дерева из массива  
template<****class****T>  
node<T>\* tree\_create(T\* arr,****int****size)  
{  
    // объявление переменной списка  
    node<T>\* root = nullptr;  
   
    // заполнение дерева значениями****for****(****int****i = 0; i < size; i++)  
    {  
        root = tree\_node\_insert(root, arr[i]);  
    }  
   
    INFO("CREATE\_TREE: дерево создано");  
   
    // возвращение созданного списка****return****root;  
}  
   
// удаление дерева  
template<****class****T>****void****tree\_delete(node<T>\*& \_root)  
{  
    // если дерево существует****if****(\_root)  
    {  
        // удаляем его  
        postorder(\_root, node\_delete);  
        \_root = nullptr;  
        INFO("TREE\_DELETE: дерево удалено");  
    }****else*** *{  
        INFO("TREE\_DELETE: дерево не существует");  
    }  
}  
   
// печать дерева  
template<****class****T>****void****tree\_print(node<T>\* \_root,****int****\_lvl)  
{****if****(\_root)  
    {  
        // вывод правого поддерева  
        tree\_print(\_root->m\_right, \_lvl + 1);  
   
        // вывод количества сдвигов  
        cout <<****fixed****<< setw(\_lvl \* WIDTH\_MULT\_PRINT) << setfill(' ') << ' ';  
   
        // вывод корня  
        cout << \_root->m\_data << endl;  
   
        // вывод левого поддерева  
        tree\_print(\_root->m\_left, \_lvl + 1);  
    }  
}  
   
// вставка элемента в дерево  
template<****class****T>  
node<T>\* tree\_node\_insert(node<T>\* \_root, T \_key)  
{  
    // если текущий элемент пуст****if****(!\_root)  
    {  
        // создаем новый элемент  
        node<T>\* new\_node =****new****node<T>(\_key);  
   
        INFO("INSERT: элемент вставлен");  
   
        // возвращаем его****return****new\_node;  
    }  
    // если ключ меньше ключа текущего элемента****if****(\_key < \_root->m\_data)  
    {  
        // идем влево  
        \_root->m\_left = tree\_node\_insert(\_root->m\_left, \_key);  
    }  
    // если ключ больше ключа текущего элемента****else******if****(\_key > \_root->m\_data)  
    {  
        // идем вправо  
        \_root->m\_right = tree\_node\_insert(\_root->m\_right, \_key);  
    }****return****\_root;  
}  
   
// поиск узла  
template<****class****T>  
node<T>\* tree\_find\_node(node<T>\* \_root, T \_key)  
{  
    // если мы не нашли элемент  
    // возвращаем nullptr****if****(!\_root)  
    {****return****nullptr;  
    }  
    // если мы нашли элемент, возвращаем \_root****else******if****(\_key == \_root->m\_data)  
    {****return****\_root;  
    }  
    // если элемент для поиска меньше текущего****else******if****(\_key < \_root->m\_data)  
    {  
        // переходим к левой ветви****return****tree\_find\_node(\_root->m\_left, \_key);  
    }  
    // если элемент для поиска больше текущего****else*** *{  
        // переходим к правой ветви****return****tree\_find\_node(\_root->m\_right, \_key);  
    }  
}  
   
// нахождение высоты дерева  
template<****class****T>****int****tree\_count\_height(node<T>\* \_root)  
{  
    // если элемент не существует****if****(!\_root)****return****0;  
   
    // если это самый низкий элемент в данной ветке****if****(!\_root->m\_left && !\_root->m\_right)****return****1;  
   
    // возвращаем максимальное значение из двух ветвей + 1****return****max(  
        tree\_count\_height(\_root->m\_left),  
        tree\_count\_height(\_root->m\_right)  
    ) + 1;  
}  
   
// нахождение количества узлов  
template<****class****T>****int****tree\_count\_nodes(node<T>\* \_root)  
{  
    // если элемент не существует****if****(!\_root)****return****0;  
   
    // если это самый низкий элемент в данной ветке****if****(!\_root->m\_left && !\_root->m\_right)****return****1;  
   
    // возвращаем сумму значений из двух ветвей + 1****return****(  
        tree\_count\_nodes(\_root->m\_left) +  
        tree\_count\_nodes(\_root->m\_right)  
        ) + 1;  
}  
   
// нахождение минимального элемента в дереве  
template<****class****T>  
node<T>\* tree\_find\_min(node<T>\* \_root)  
{  
    node<T>\* current = \_root;  
   
    // идем по левой ветке, пока не дойдем до конца****while****(current && current->m\_left)  
    {  
        current = current->m\_left;  
    }****return****current;  
}  
   
// нахождение максимального элемента в дереве  
template<****class****T>  
node<T>\* tree\_find\_max(node<T>\* \_root)  
{  
    node<T>\* current = \_root;  
   
    // идем по правой ветке, пока не дойдем до конца****while****(current && current->m\_right)  
    {  
        current = current->m\_right;  
    }****return****current;  
}  
   
// удаление элемента дерева  
template<****class****T>  
node<T>\* tree\_node\_delete(node<T>\* \_root, T \_key)  
{  
    // Возвращаем, если дерево пустое****if****(!\_root)****return****\_root;  
   
    // Ищем узел, который хотим удалить****if****(\_key < \_root->m\_data)  
    {  
        // идем на левую ветвь  
        \_root->m\_left = tree\_node\_delete(\_root->m\_left, \_key);  
    }****else******if****(\_key > \_root->m\_data)  
    {  
        // идем на правую ветвь  
        \_root->m\_right = tree\_node\_delete(\_root->m\_right, \_key);  
    }  
    // если был найден нужный элемент****else****{  
        // Если у узла один дочерний элемент или их нет  
        // если нет левой ветви****if****(!\_root->m\_left) {  
            node<T>\* temp = \_root->m\_right;  
            delete \_root;****return****temp;  
        }  
        // если нет правой ветви****else******if****(!\_root->m\_right) {  
            node<T>\* temp = \_root->m\_left;  
            delete \_root;****return****temp;  
        }  
   
        // Если у узла два дочерних элемента  
        // находим минимальный элемент в правой ветви  
        node<T>\* temp = tree\_find\_min(\_root->m\_right);  
   
        // помещаем найденный элемент в тот, который хотим удалить  
        \_root->m\_data = temp->m\_data;  
   
        // удаляем скопированный элемент из правой ветви дерева  
        \_root->m\_right = tree\_node\_delete(\_root->m\_right, temp->m\_data);  
    }****return****\_root;  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*        В С П О М О Г А Т Е Л Ь Н Ы Е   Ф У Н К Ц И И          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// проверка на номер команды****bool****check\_number(****int****\_num)  
{****return****0 <= \_num && \_num <= 14;  
}  
   
// проверка на корректность введенного символа****bool****check\_cyrillic\_symbol(****char****\_symb)  
{****return****'А' <= \_symb && \_symb <= 'я'  
        || \_symb == 'ё' || \_symb == 'Ё';  
}  
   
// пример готовой программы****void****example\_program()  
{  
    // элементы, которые окажутся в дереве****char****\* mass =****new******char****[10] { 'а', 'б', 'Г', 'Ж', 'я', 'д', 'е', 'А', 'м', 'Н' };  
   
    // заполнение дерева элементами  
    node<****char****>\* root = tree\_create(mass, 10);  
   
    // печать дерева  
    tree\_print(root);  
   
    // поиск элемента в дереве****char****to\_find = 'Ж';  
    node<****char****>\* finded\_node = tree\_find\_node(root, to\_find);  
    // если элемент был найден****if****(finded\_node)  
    {  
        cout << "Найденный элемент: " << finded\_node->m\_data << endl;  
    }****else*** *{  
        cout << "Элемент" << to\_find << " не был найден\n";  
    }  
   
    // печать характеристик дерева  
    cout << "Количество узлов дерева: " << tree\_count\_nodes(root) << endl;  
    cout << "Высота дерева: " << tree\_count\_height(root) << endl;  
    cout << "Число узлов в левом поддереве: " << tree\_count\_nodes(root->m\_left) << endl;  
    cout << "Число узлов в правом поддереве: " << tree\_count\_nodes(root->m\_right) << endl;  
    cout << "Минимальный элемент: " << tree\_find\_min(root)->m\_data << endl;  
    cout << "Максимальный элемент: " << tree\_find\_max(root)->m\_data << endl;  
   
    // удаление из дерева  
    root = tree\_node\_delete(root, 'в');  
    root = tree\_node\_delete(root, 'б');  
    root = tree\_node\_delete(root, 'д');  
   
    // вывод дерева  
    tree\_print(root);  
   
    // удаление дерева  
    tree\_delete(root);  
}  
   
// ввод и проверка значений  
template<typename T>  
T input\_and\_check(****bool****(\*\_comp)(T),****const******char****\* welcome\_str,****const******char****\* err\_str)  
{  
    // размер массива  
    T symb;  
   
    // вывод сообщения  
    cout << welcome\_str << "\n";  
    cin >> symb;  
   
    // если было введено некорректное значение****if****(!\_comp(symb)) {  
        // если была введено не то, что нужно было****if****(cin.fail())  
        {  
            cin.clear();  
            cin.ignore(INT\_MAX, '\n');  
        }  
        cout << err\_str << "\n";  
   
        // рекурсивное обращение  
        symb = input\_and\_check(\_comp, welcome\_str, err\_str);  
    }****return****symb;  
}  
   
// функция ведения диалога с пользователем****void****dialog()  
{  
    // переменная содержащая коды действий  
    input\_codes in\_code;  
   
    // элемент для вставки****char****symb;  
   
    // переменная корня  
    node<****char****>\* root = nullptr;  
   
    // элемент для поиска max и min в дереве  
    node<****char****>\* elem = nullptr;  
   
    /\* коды комманд  
    exit,                               1  
    template\_program,                   2  
    tree\_insert,                        3  
    tree\_delete\_node,                   4  
    tree\_print,                         5  
    tree\_delete,                        6  
    tree\_find\_max,                      7  
    tree\_find\_min,                      8  
    tree\_count\_height,                  9  
    tree\_count\_nodes,                   10  
    tree\_count\_nodes\_in\_left\_subtree,   11  
    tree\_count\_nodes\_in\_right\_subtree,  12  
    tree\_find\_node,                     13  
    clear\_console                       14  
    \*/****do*** *{  
        // запрос у пользователя следующих действий  
        in\_code = input\_codes(  
            input\_and\_check(check\_number, command\_str)  
            - 1);  
   
        // запуск соответствующих функций****switch****(in\_code)  
        {****case****input\_codes::exit:  
            INFO("Произведен выход");****break****;****case****input\_codes::template\_program:  
            INFO("Запуск примера кода");  
            example\_program();****break****;****case****input\_codes::tree\_insert:  
            symb = input\_and\_check(check\_cyrillic\_symbol,  
                "Введите символ для вставки в дерево: ");  
            root = tree\_node\_insert(root, symb);****break****;****case****input\_codes::tree\_delete\_node:  
            symb = input\_and\_check(check\_cyrillic\_symbol,  
                "Введите символ для удаления в дереве: ");  
            root = tree\_node\_delete(root, symb);****break****;****case****input\_codes::tree\_print:  
            cout << endl;  
            tree\_print(root);****break****;****case****input\_codes::tree\_delete:  
            tree\_delete(root);****break****;****case****input\_codes::tree\_find\_max:  
            elem = tree\_find\_max(root);****if****(elem)  
                cout << "Максимальный элемент в дереве: "  
                << elem->m\_data << endl;****else*** *INFO("FIND\_MAX: дерево пусто");****break****;****case****input\_codes::tree\_find\_min:  
            elem = tree\_find\_min(root);****if****(elem)  
                cout << "Минимальный элемент в дереве: "  
                << elem->m\_data << endl;****else*** *INFO("FIND\_MIN: дерево пусто");****break****;****case****input\_codes::tree\_count\_height:  
            cout << "Высота дерева: " <<  
                tree\_count\_height(root) << endl;****break****;****case****input\_codes::tree\_count\_nodes:  
            cout << "Количество элементов в дереве: " <<  
                tree\_count\_nodes(root) << endl;****break****;****case****input\_codes::tree\_count\_nodes\_in\_left\_subtree:****if****(root)  
                cout << "Количество элементов в левом поддереве: " <<  
                tree\_count\_nodes(root->m\_left) << endl;****else*** *INFO("TREE\_COUNT\_NODES\_IN\_LEFT\_SUBTREE: дерево пусто");****break****;****case****input\_codes::tree\_count\_nodes\_in\_right\_subtree:****if****(root)  
                cout << "Количество элементов в правом поддереве: " <<  
                tree\_count\_nodes(root->m\_right) << endl;****else*** *INFO("TREE\_COUNT\_NODES\_IN\_RIGHT\_SUBTREE: дерево пусто");****break****;****case****input\_codes::tree\_find\_node:  
            symb = input\_and\_check(check\_cyrillic\_symbol,  
                "Введите символ для поиска в дереве: ");  
            elem = tree\_find\_node(root, symb);****if****(elem)  
                cout << "Элемент [" << symb << "] был найден\n";****else*** *cout << "Элемент [" << symb << "] НЕ был найден\n";****break****;****case****input\_codes::clear\_console:  
            system("cls");****break****;****default****:  
            INFO("Неизвестный код")****break****;  
        }  
   
    }****while****(  
        // пока пользователь не захотел выйти из программы  
        // или пока не запустил пример программыЫ  
        in\_code != input\_codes::exit &&  
        in\_code != input\_codes::template\_program  
        );  
}  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End Of main.cpp File \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

# Тестирование программы

### Тест 1

Выполнение заранее созданного алгоритма.





# Вывод

Работа программы завершена на основании:

1) Полученные результаты совпали с ожидаемыми;

2) Считаем набор тестов полным.