Министерство образования Республики Беларусь

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра экономической информатики

**Лабораторная работа №2**

«**Деревья»**

**Вариант 13**

|  |  |
| --- | --- |
| Выполнил:  Группа: | Пчельник М.А.  974001 |
| Проверил: | П.А. Корбит |

**Минск 2020**

**Цель работы:** изучить особенности работы c бинарными деревьями, научиться поиску элементов списка и балансировке.

**Код программы:**

#include <stdio.h>

#include <conio.h>

#include <iostream>

struct Tree {int info;

Tree \*Left, \*Right;} \*Root;

void Make(int);

void Print(Tree\*, int);

void Del(int);

void Del\_All(Tree\*);

Tree\* List(int b);

void Add\_List(Tree \*Root, int m);

void PoiskInfo(Tree \*Root, int key);

int Store\_Tree(Tree \*p, int pos, int \*a);

int KolElementTree(Tree\* tree);

void make\_blns(Tree \*\*p, int n, int k, int \*a);

int poiskMax(Tree \*t);

int poiskMin(Tree \*t);

int Average(Tree \*t);

void PoiskAverage(Tree \*t, int Sr, int &difference, int &number);

void main() {int b, found, key, m, number, difference = 100, Sr;;

bool flag = true;

setlocale(LC\_ALL, "rus");

while (flag) {puts(" Creat - 1\n View - 2\n Find - 3 \nDel Key - 4\n среднее значение -5\n баллансировка\nEXIT - 0");

switch (\_getch()) {

case '1': if (Root == NULL) {

int b;

printf( "Введите корень

дерева:");

scanf\_s("%d", &b);

Root = List(b);}

else {

printf("Введите лист дерева:");

scanf\_s("%d", &m);

Add\_List(Root,m); }

break;

case '2': if (Root == NULL) puts("\t END TREE !");

else Print(Root, 0);

break;

case '3': printf("Введите информацию в ключе, которого пытаетесь найти:");

int k;

scanf\_s("%d", &k);

PoiskInfo(Root, k);

break;

case '4': puts("\n Input Del Info "); scanf\_s("%d", &key);

Del(key);

break;

case '5':if (Root == NULL) {printf("Дерево пусто!");}

else { Sr = Average(Root);

difference = Root->info;

PoiskAverage(Root, Sr, difference, number);

printf("Ближайшее значение к среднему:");

printf ("%d",number);}

break;

case '6':{ if (Root == NULL) {

printf("Создайте дерево");

break; }

else {printf("Баллансировка дерева");

Tree \*t;

int k;

t = Root;

k = KolElementTree(t);

int \*a = new int[k];

int pos = 0;

Store\_Tree(Root, pos, a);

make\_blns(&Root, 0, k, a);

delete[]a;

printf("Дерево сбалансировано!");

break; }}

break;

case '0': Del\_All(Root);

puts("\n Tree Delete!");

return;}}}

//============= Удаление элемента по ключу (не корень) ===============

void Del(int key) {Tree \*Del, \*Priooev\_Del, \*R, \*Prev\_R, \*Prev\_Del;

Del = Root; Prev\_Del = NULL;

//-------- Поиск удаляемого элемента и его предка по ключу key---------------------

while (Del != NULL && Del->info != key) {

Prev\_Del = Del;

if (Del->info >key) Del = Del->Left;

else Del = Del->Right;

}

if (Del == NULL){ // В дереве такого ключа нет

puts("\n NO Key!");

return;}

//--------------------- Поиск элемента для замены R ----------------------------------------

if (Del->Right == NULL) R = Del->Left;

else

if (Del->Left == NULL) R = Del->Right;

else {Prev\_R = Del;

R = Del->Left;

while (R->Right != NULL) {

Prev\_R = R;

R = R->Right;}

if (Prev\_R == Del) R->Right = Del->Right;

else { R->Right = Del->Right;

Prev\_R->Right = R->Left;

R->Left = Prev\_R;}}

if (Prev\_Del == NULL) { Root = Prev\_Del = R; }

else

if (Del->info < Prev\_Del->info) Prev\_Del->Left = R;

else Prev\_Del->Right = R;

printf("\n Delete %d element ", Del->info);

delete Del;}

//============= Формирование (создание) элемента - листа =============

Tree\* List(int b) {

Tree \*t = new Tree; // Захват памяти

t->info = b;

t->Left = t->Right = NULL;

return t;}

void Add\_List(Tree \*root, int m) {

Tree \*prev, \*t;

bool find = true;

t = root;

prev = t;

while (t && find) {

prev = t;

if (m== t->info) {find = false;

printf("Ключ должен быть уникален!" );}

else

if (m< t->info) t = t->Left;

else t = t->Right;}

if (find) {t = List(m);

if (m < prev->info) prev->Left = t;

else prev->Right = t;}}

//=============Поиск==========================/

void PoiskInfo(Tree \*t, int k) {Tree \*c;

c = NULL;

while (t != NULL && t->info != k) { c= t;

if (t->info > k) t = t->Left;

else t = t->Right;}

if (t == NULL) { printf("Элемент не найден");}

else {printf("По ключу:%d, найдено следующее:%d ", k, t->info);}}

//=============Балансировка==========================/

int KolElementTree(Tree\* tree) {if (tree->Left == NULL && tree->Right == NULL) return 1;

int Left, Right;

if (tree->Left != NULL) {Left = KolElementTree(tree->Left);}

else Left = 0;

if (tree->Right != NULL) {Right = KolElementTree(tree->Right);}

else Right = 0;

return Left + Right + 1;}

int Store\_Tree(Tree \*p, int pos, int \*a) {//занесение данных в массив

if (p) {pos = Store\_Tree(p->Left, pos, a);

a[pos++] = p->info;

pos = Store\_Tree(p->Right, pos, a);}

return pos;}

void make\_blns(Tree \*\*p, int n, int k, int \*a) {if (n == k)

{\*p = NULL;//при обращении n=0,k=размер массива

return;}

else {int m = (n + k) / 2;

\*p = new Tree;

(\*p)->info = a[m];

make\_blns(&(\*p)->Left, n, m, a);

make\_blns(&(\*p)->Right, m + 1, k, a);}}

//=========== Функция вывода на экран ========================

void Print(Tree \*p, int level) {if (p) {

Print(p->Right, level + 1);

for (int i = 0; i<level; i++) printf(" ");

printf("%d \n", p->info);

Print(p->Left, level + 1); // }}

//=====================Найти ключ ближайший к среднему значению==================

int poiskMax(Tree \*t) {int MaxElement;

while (t->Right != NULL) {t = t->Right;}

MaxElement = t->info;

return MaxElement;}

int poiskMin(Tree \*t) {int MinElement;

while (t->Left != NULL) {

t = t->Left;}

MinElement = t->info;

return MinElement;}

int Average(Tree \*t) {int Average;

Average = (poiskMax(Root) + poiskMin(Root)) / 2;

printf("Среднее значение минимального и максимального элемента:%d", Average);

return Average;}

void PoiskAverage(Tree \*t, int Sr, int &difference, int &number) {

int TempRoot;

if (t != NULL) {TempRoot = t->info;

TempRoot = abs(t->info - Sr);

if (TempRoot < difference) {

difference = TempRoot;

number = t->info;}

PoiskAverage(t->Left, Sr, difference, number);

PoiskAverage(t->Right, Sr, difference, number);}}

//===================== Освобождение памяти ==================

void Del\_All(Tree \*t) {

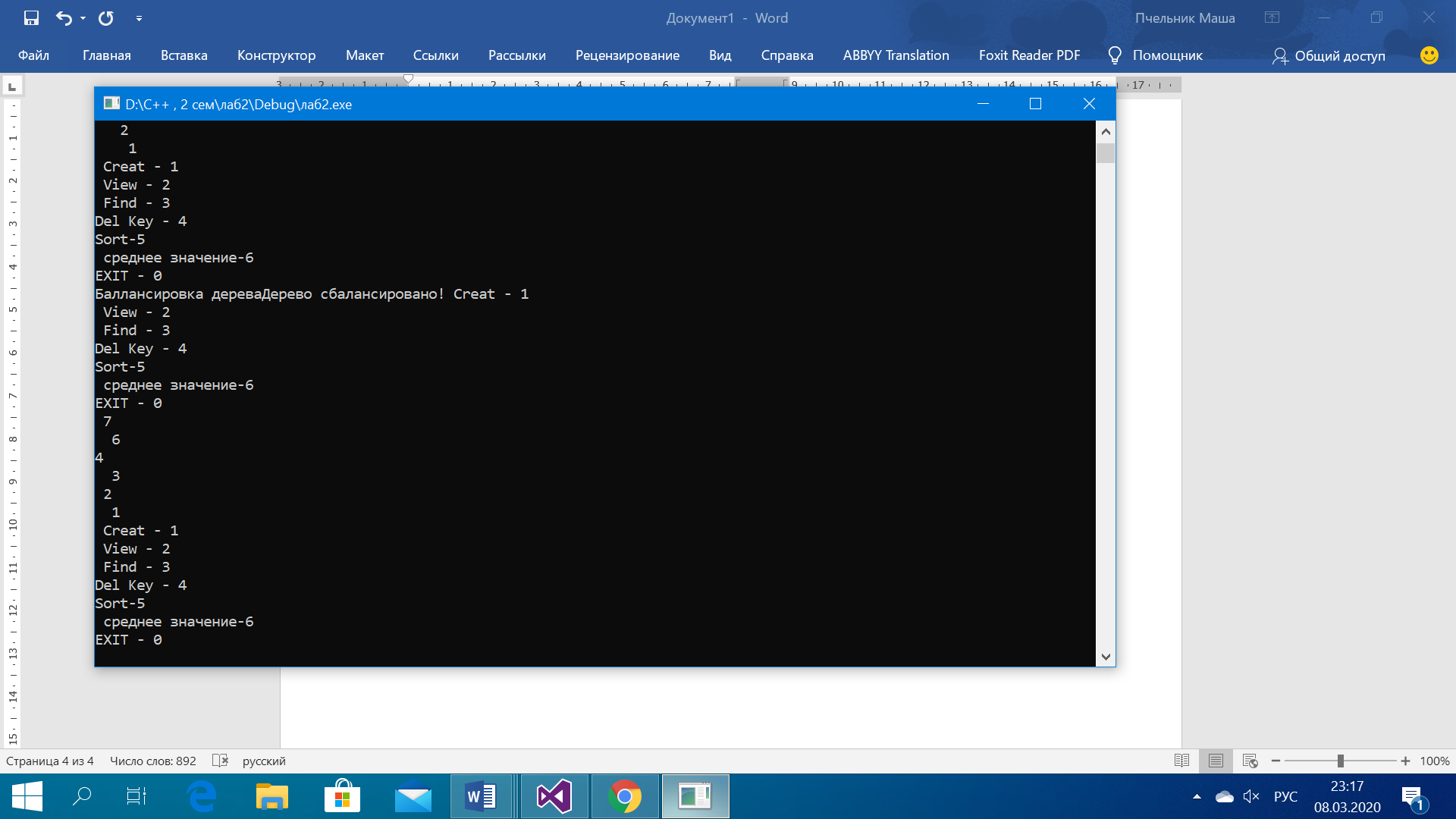
if (t != NULL) {

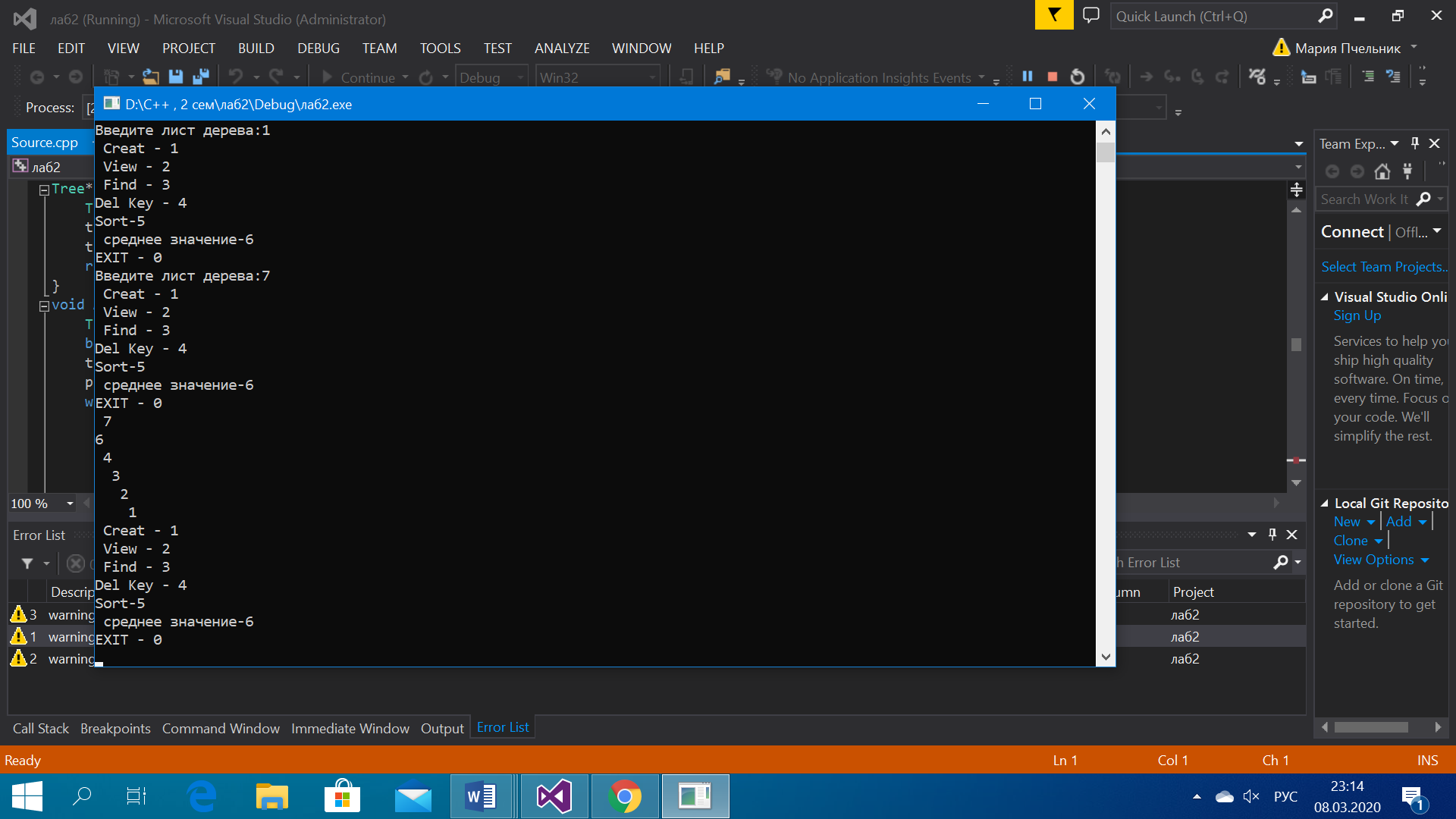
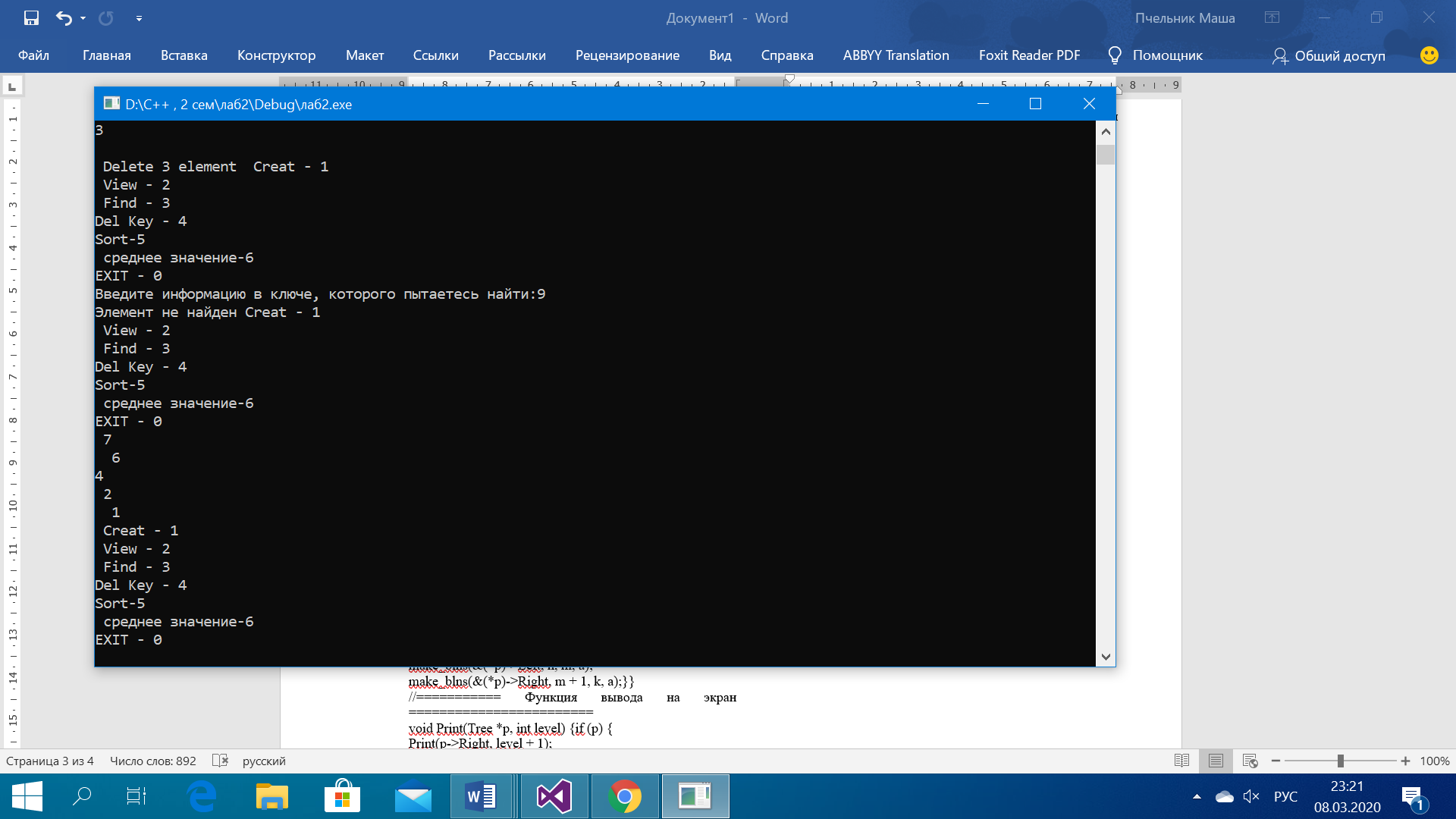
Del\_All(t->Left);

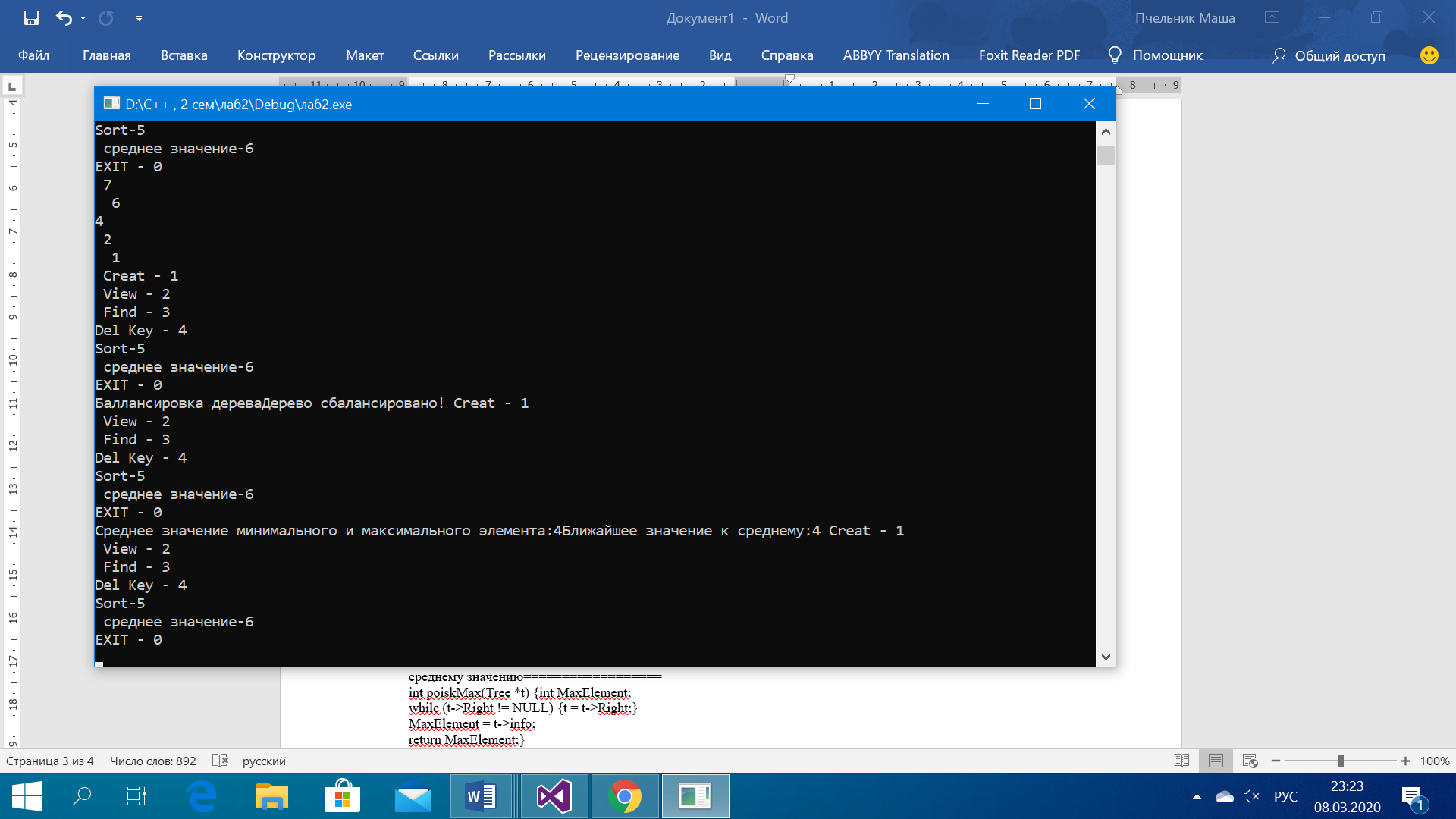
Del\_All(t->Right);

delete t;}}

**Результат выполнения программы:**





**Вывод:** изучили особенности работы с деревьями, научились сортировать, искать и удалять элементы дерева.