Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Дискретный анализ»**

Студент: Пивницкий Д.С,

Группа: М8О–206Б–20

Вариант: 02

Преподаватель: А.А.Кухтичев

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

**Задача:**Необходимо создать программную библиотеку, реализующую указанную структуру данных, на основе которой разработать программу-словарь. В словаре каждому ключу, представляющему из себя регистронезависимую последовательность букв английского алфавита длиной не более 256 символов, поставлен в соответствие некоторый номер, от 0 до 264 - 1. Разным словам может быть поставлен в соответствие один и тот же номер.

Программа должна обрабатывать строки входного файла до его окончания. Каждая строка может иметь следующий формат:

+ **word 34** — добавить слово «word» с номером 34 в словарь. Программа должна вывести строку «OK», если операция прошла успешно, «Exist», если слово уже находится в словаре.   
- **word** — удалить слово «word» из словаря. Программа должна вывести «OK», если слово существовало и было удалено, «NoSuchWord», если слово в словаре не было найдено.   
**word** — найти в словаре слово «word». Программа должна вывести «OK: 34», если слово было найдено; число, которое следует за «OK:» — номер, присвоенный слову при добавлении. В случае, если слово в словаре не было обнаружено, нужно вывести строку «NoSuchWord».   
**! Save /path/to/file** — сохранить словарь в бинарном компактном представлении на диск в файл, указанный параметром команды. В случае успеха, программа должна вывести «OK», в случае неудачи выполнения операции, программа должна вывести описание ошибки (см. ниже). ! Load **/path/to/file** — загрузить словарь из файла. Предполагается, что файл был ранее подготовлен при помощи команды Save. В случае успеха, программа должна вывести строку «OK», а загруженный словарь должен заменить текущий (с которым происходит работа); в случае неуспеха, должна быть выведена диагностика, а рабочий словарь должен остаться без изменений. Кроме системных ошибок, программа должна корректно обрабатывать случаи несовпадения формата указанного файла и представления данных словаря во внешнем файле.

Для всех операций, в случае возникновения системной ошибки (нехватка памяти, отсутствие прав записи и т.п.), программа должна вывести строку, начинающуюся с «ERROR:» и описывающую на английском языке возникшую ошибку. Используемая структура данных: Красно-чёрное дерево.

**Описание**

Требуется реализовать программу-словарь, работающую на основе красно-черного дерева. Процесс разработки можно разделить на два этапа: создание и отладка структуры красно-черное дерево и реализация парсера команд (словаря), работающего с этим деревом.

Дерево должно хранить в своих вершинах слова (в формате строки) и соответствующие им числа. Так как слова в словаре должны быть отсортированы в алфавитном порядке, ключом будет считаться именно слово, а соответствующее ему число – значением.

Красно-черное дерево – самобалансирующееся дерево, узлы которого имеют атрибут цвета. При этом, дерево должно удовлетворять условиям: 1. Узел может быть либо красным, либо чёрным и имеет двух потомков; 2. Корень – чёрный. 3. Все листья — чёрные. 4. Оба потомка каждого красного узла — чёрные. 5. Любой простой путь от узла-предка до листового узла-потомка содержит одинаковое число чёрных узлов. Для соблюдения этих условий после каждой операции вставки/удаления узла должна проводиться перебалансировка. Время, затрачиваемое на доступ к элементам, составляет O(lon(n)), где n – число элементов в нем.

Перебаланисировка производится с помощью операций поворота и перекрашивания узлов, которые применяются в зависимости от текущей расцветки и положений узлов дерева. В ходе перебалансировки количество черных узлов на пути из элемента, для которого операция перебалансировки была вызвана до всех его листовых потомков уравниваются, однако так как при этом их суммарное число могло измениться, в некоторых случаях операция рекурсивно вызывается для его потомков, пока не отбалансируется все дерево начиная с корня.

В ходе разработки структуры «красно-черное дерево» также необходимо реализовать операции сохранения его в файл и загрузки из файла. Элементы дерева записываются в файл по очереди их прохождения в процессе обхода в глубину, при этом, вначале записывается текущий элемент, затем – оба его потомка, начиная с левого. Таким образом, во время чтения дерева из заранее сохраненного файла при обнаружении очередного элемента мы можем быть уверены, что следующий за ним элемент является его левым потомком. Если был найден лист (элемент в котором лежит строка длины ноль), то мы понимаем, что текущая ветвь закончилась, и далее записан правый потомок родителя считанного ранее узла.

Для того чтобы не тратить время на попытки расшифровки файлов, которые были созданы не программой словаря, в начало файла добавляется маркер, состоящий из трех символов. При чтении дерева файла структура строится отдельно от текущего дерева, и только если чтение прошло без ошибок, текущее дерево заменяется считанным.

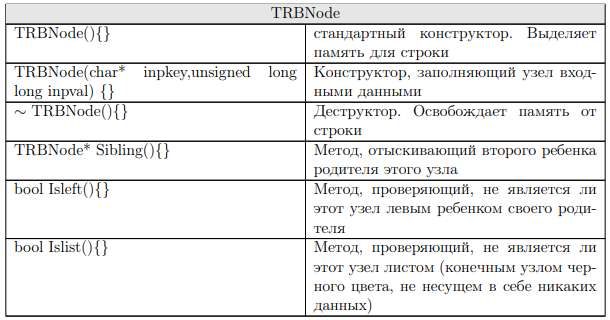
В самом словаре производится непрерывная обработка входных строк, в соответствии с результатом этой обработки для каждой введенной команды вызывается метод заранее созданного дерева. Методы в большинстве своем возвращают код завершения, в соответствии с которым осуществляется вывод на экран информации о результате выполнения команды.

**Исходный код**

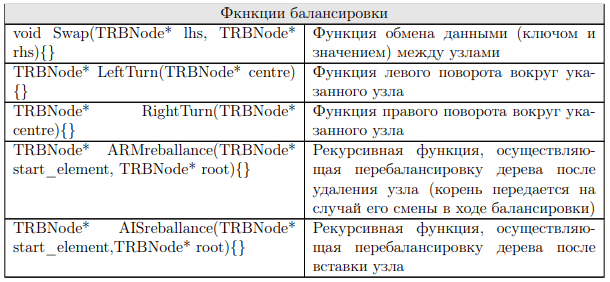
Вначале создаем класс узла дерева. В нем хранится ключ (указатель на строку), значение (переменная типа unsigned long long) и цвет (переменная типа char). Также в узле хранятся указатели на родителя и обоих детей этого узла.

Для компактности отчета здесь и далее я буду приводить методы класса только в таблице

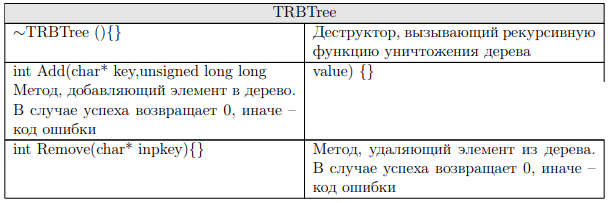
Методы, реализованные для класса узла:

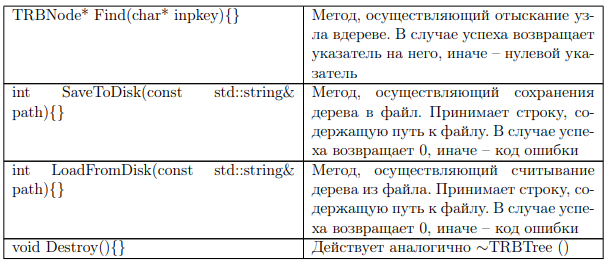


Также реализуем вспомогательную функцию strequal, работающую аналогично strcmp, однако не реагирующую на различный регистр букв сравниваемых строк. Для вставки/удаления узлов, а также последующей перебалансировки нам потребуются следующие функции:

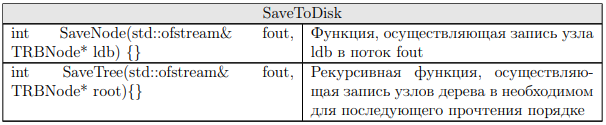


В дереве хранится только корень, работа с деервом осуществляется через его методы:

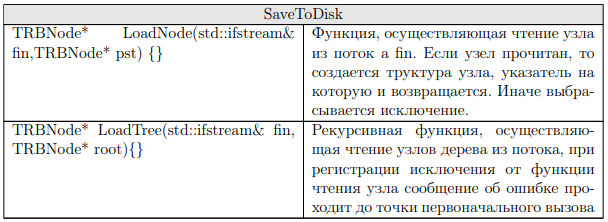




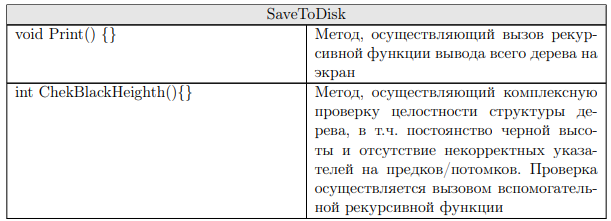
Вспомогательные функции для int SaveToDisk(const std::string path) {}



Вспомогательные функции для int LoadFromDisk(const std::string path) {}



Также для класса дерева реализован ряд дополнительных функций, ненужных при непосредственной работе программы, однако крайне необходимых при отладке:



В функции main будем производить обработку команд, предварительно создав дерево, в которое будем записывать слова.

**Листинг программы**

**#include <iostream>**

**#include <vector>**

**#include <algorithm>**

**#include <fstream>**

**#include <cstring>**

**const char BLACK = 1;**

**const char RED = 2;**

**class TRBNode { //структура узла дерева**

**public:**

**char\* Key;**

**unsigned long long Value;**

**char Colour = BLACK;**

**TRBNode\* Parent = nullptr;**

**TRBNode\* Left = nullptr;**

**TRBNode\* Right = nullptr;**

**TRBNode() {**

**Key = (char\*)calloc(260,sizeof(char));**

**if(Key == nullptr) {**

**throw std::bad\_alloc();**

**}**

**Value = 0;**

**Key = strcpy(Key,"");**

**}**

**TRBNode(char\* inpkey,unsigned long long inpval) {**

**Key = (char\*)calloc(260,sizeof(char));**

**if(Key == nullptr) {**

**throw std::bad\_alloc();**

**}**

**Key = strcpy(Key,inpkey);**

**Value = inpval;**

**}**

**TRBNode(unsigned long long inpval, char\* inpkey) {**

**Key = (char\*)calloc(260,sizeof(char));**

**if(Key == nullptr) {**

**throw std::bad\_alloc();**

**}**

**Key = strcpy(Key,inpkey);**

**Value = inpval;**

**}**

**~TRBNode() {**

**//if(Key != nullptr) {**

**free(Key);**

**//}**

**}**

**TRBNode\* Sibling() {**

**if (Parent != nullptr && Parent->Left != nullptr && Parent->Right != nullptr) {**

**if (Parent->Left->Key == Key) {**

**return Parent->Right;**

**}**

**else {**

**return Parent->Left;**

**}**

**}**

**else {**

**return nullptr;**

**}**

**}**

**bool Isleft() {**

**if (Parent != nullptr && Parent->Left != nullptr && Parent->Left->Key != nullptr && Parent->Left->Key == Key) {**

**return true;**

**}**

**else {**

**return false;**

**}**

**}**

**bool Islist() {**

**if (Right == nullptr || Left == nullptr) {**

**return true;**

**}**

**else {**

**return false;**

**}**

**}**

**};**

**int strequal (const char\* lhs, const char\* rhs) { //операторы регистронезависимого сравненияя для std::string**

**//printf("%s|%s\n",lhs,rhs);**

**if(lhs == nullptr || rhs == nullptr) {**

**if(lhs == nullptr) {**

**return -1;**

**}**

**if(rhs == nullptr) {**

**return 1;**

**}**

**}**

**size\_t i=0;**

**while (lhs[i]!=0 || rhs[i]!=0) {**

**if (lhs[i] == rhs[i]) {**

**++i;**

**continue;**

**}**

**else {**

**char cmp\_l = lhs[i];**

**char cmp\_r = rhs[i];**

**if(lhs[i] >= 'A' && lhs[i] <= 'Z') {**

**cmp\_l = lhs[i] + ('a' - 'A');**

**}**

**if(rhs[i] >= 'A' && rhs[i] <= 'Z') {**

**cmp\_r = rhs[i] + ('a' - 'A');**

**}**

**if(cmp\_l < cmp\_r) {**

**return -1;**

**} else**

**if(cmp\_l > cmp\_r) {**

**return 1;**

**} else {**

**++i;**

**continue;**

**}**

**}**

**}**

**if(lhs[i]!=0 || rhs[i]!=0)**

**{**

**if(lhs[i]==0) {**

**return -1;**

**}**

**if(rhs[i]==0) {**

**return 1;**

**}**

**}**

**return 0;**

**}**

**void Tree\_print(TRBNode\* start, int counter) {**

**if (start == nullptr) {**

**return;**

**}**

**Tree\_print(start->Right, counter + 1);**

**for (int i = 0; i < counter; ++i) {**

**printf("\t");**

**}**

**printf("( ");**

**if(start->Parent!=nullptr) {**

**printf("%s<-p|",start->Parent->Key);**

**}**

**else {**

**printf("nullptr <-p|");**

**}**

**printf("%s",start->Key);**

**printf(" | %llu ) - | ", start->Value);**

**if (start->Colour == RED) {**

**printf("RED");**

**}**

**else {**

**printf("BLACK");**

**}**

**printf("|\n");**

**Tree\_print(start->Left, counter + 1);**

**}**

**void Swap(TRBNode\* lhs, TRBNode\* rhs) {**

**char\* tmpkey = (char\*)calloc(300,sizeof(char));**

**if(tmpkey == nullptr) {**

**throw -1;**

**}**

**tmpkey = strcpy(tmpkey,rhs->Key);**

**unsigned long long tmpval = rhs->Value;**

**rhs->Key = strcpy(rhs->Key,lhs->Key);**

**rhs->Value = lhs->Value;**

**lhs->Key = strcpy(lhs->Key,tmpkey);**

**lhs->Value = tmpval;**

**free(tmpkey);**

**}**

**TRBNode\* LeftTurn(TRBNode\* centre) {**

**if (centre->Parent != nullptr) {//сещение указателя сверху на новый центр смещения**

**if (centre->Parent->Left == centre) {**

**centre->Parent->Left = centre->Left;**

**}**

**else**

**if (centre->Parent->Right == centre) {**

**centre->Parent->Right = centre->Left;**

**}**

**}**

**TRBNode\* new\_centre = centre->Left;**

**centre->Left = centre->Left->Right;**

**centre->Left->Parent = centre;**

**//std::cout << centre is: " << new\_centre->Key << std::endl;**

**new\_centre->Parent = centre->Parent;**

**new\_centre->Right = centre;**

**centre->Parent = new\_centre;**

**return new\_centre;**

**}**

**TRBNode\* RightTurn(TRBNode\* centre) {**

**//std::cout << "right turn from " << centre->Key << " " << centre->Value << "\n";**

**if (centre->Parent != nullptr) {//сещение указателя сверху на новый центр смещения**

**//std::cout << "parent\_cons: " << centre->Parent->Left->Key << " " << centre->Parent->Right->Key << std::endl;**

**if (centre->Parent->Left == centre) {**

**//std::cout << "centre is left" << std::endl;**

**centre->Parent->Left = centre->Right;**

**}**

**else**

**if (centre->Parent->Right == centre) {**

**//std::cout << "centre is right" << std::endl;**

**centre->Parent->Right = centre->Right;**

**}**

**}**

**TRBNode\* new\_centre = centre->Right;**

**centre->Right = centre->Right->Left;**

**centre->Right->Parent = centre;**

**//std::cout << "new centre is: " << new\_centre->Key << std::endl;**

**new\_centre->Parent = centre->Parent;**

**new\_centre->Left = centre;**

**centre->Parent = new\_centre;**

**return new\_centre;**

**}**

**TRBNode\* ARMreballance(TRBNode\* start\_element, TRBNode\* root) //перебаллансировка после удаления узла дерева. Начальный элемент - тот, который был удален (сейчас на его месте пустой лист)**

**{**

**if (root == start\_element) {**

**return root;**

**}**

**else {**

**TRBNode\* sibl = start\_element->Sibling();**

**//std::cout << "ballancing... s\_e\_p: " << start\_element->Parent->Key << std::endl;**

**//std::cout << "ballancing... sibl: " << sibl->Key << std::endl;**

**if (sibl->Colour == RED) {**

**if (sibl->Isleft()) {**

**if (start\_element->Parent == root) {**

**root = LeftTurn(start\_element->Parent);**

**}**

**else {**

**LeftTurn(start\_element->Parent);**

**}**

**sibl->Colour = BLACK;**

**sibl->Right->Colour = RED;//**

**root = ARMreballance(start\_element, root);**

**}**

**else {**

**if (start\_element->Parent == root) {**

**root = RightTurn(start\_element->Parent);**

**}**

**else {**

**RightTurn(start\_element->Parent);**

**}**

**sibl->Colour = BLACK;**

**sibl->Left->Colour = RED;**

**root = ARMreballance(start\_element, root);**

**}**

**}**

**else {**

**if (sibl->Isleft()) {**

**if (sibl->Left->Colour == BLACK && sibl->Right->Colour == BLACK && sibl->Parent->Colour == BLACK) {**

**sibl->Colour = RED;**

**sibl->Parent->Colour = BLACK;**

**root = ARMreballance(start\_element->Parent, root);**

**}**

**else**

**if (sibl->Left->Colour == BLACK && sibl->Right->Colour == BLACK && sibl->Parent->Colour == RED) {**

**sibl->Colour = RED;**

**sibl->Parent->Colour = BLACK;**

**}**

**else**

**if (sibl->Right->Colour == RED && sibl->Left->Colour == BLACK) {**

**sibl->Right->Colour = BLACK;**

**sibl->Colour = RED;**

**RightTurn(sibl);**

**root = ARMreballance(start\_element, root);**

**}**

**else**

**if (sibl->Left->Colour == RED) {**

**sibl->Colour = sibl->Parent->Colour;**

**sibl->Parent->Colour = BLACK;**

**//sibl->Right->Colour = BLACK;**

**sibl->Left->Colour = BLACK;**

**if (sibl->Parent == root) {**

**root = LeftTurn(sibl->Parent);**

**}**

**else {**

**LeftTurn(sibl->Parent);**

**}**

**}**

**}**

**else**

**{**

**if (sibl->Left->Colour == BLACK && sibl->Right->Colour == BLACK && sibl->Parent->Colour == BLACK) {**

**sibl->Colour = RED;**

**sibl->Parent->Colour = BLACK;**

**root = ARMreballance(start\_element->Parent, root);**

**}**

**else**

**if (sibl->Left->Colour == BLACK && sibl->Right->Colour == BLACK && sibl->Parent->Colour == RED) {**

**sibl->Colour = RED;**

**sibl->Parent->Colour = BLACK;**

**}**

**else**

**if (sibl->Left->Colour == RED && sibl->Right->Colour == BLACK) {**

**sibl->Left->Colour = BLACK;**

**sibl->Colour = RED;**

**LeftTurn(sibl);**

**root = ARMreballance(start\_element, root);**

**}**

**else**

**if (sibl->Right->Colour == RED) {**

**sibl->Colour = sibl->Parent->Colour;**

**sibl->Parent->Colour = BLACK;**

**sibl->Right->Colour = BLACK;**

**if (sibl->Parent == root) {**

**root = RightTurn(sibl->Parent);**

**}**

**else {**

**RightTurn(sibl->Parent);**

**}**

**}**

**}**

**}**

**return root;**

**}**

**}**

**TRBNode\* AISreballance(TRBNode\* start\_element,TRBNode\* root) //предполагается запуск корректора из только что вставленного элемента => этот элемент красный**

**{**

**//std::cout << "print started | uncorrected\n";**

**//Tree\_print(root, 0);**

**//std::cout << "print finished | uncorrected\n";**

**if (start\_element->Parent == nullptr) {**

**start\_element->Colour = BLACK;**

**}**

**// std::cout << "starting correct at elem: " << start\_element->Key << std::endl;**

**/\* if (start\_element->Parent != nullptr && start\_element->Parent->Sibling() != nullptr)**

**{**

**std::cout << "sibl: " << start\_element->Parent->Sibling()->Key << " "; printf("%d\n", start\_element->Parent->Sibling()->Colour);**

**}\*/**

**if (start\_element->Colour == RED) {**

**if (start\_element->Parent != nullptr && start\_element->Parent->Colour == RED) {**

**if (start\_element->Parent->Sibling() != nullptr) {**

**if (start\_element->Parent->Sibling()->Colour == BLACK) {**

**int rcm = 0;**

**TRBNode\* new\_centre;**

**if (!start\_element->Parent->Isleft()) {**

**if (start\_element->Isleft()) {**

**//std::cout << "left long turn initiated from element: " << start\_element->Key << std::endl;**

**if (start\_element->Parent->Parent == root) {**

**rcm = 1;**

**}**

**//std::cout << "left turn initiated from element: " << start\_element->Key << std::endl;**

**new\_centre = LeftTurn(start\_element->Parent);**

**new\_centre = RightTurn(start\_element->Parent);**

**if (rcm == 1) {**

**// std::cout << "rcm" << std::endl;**

**root = new\_centre;**

**rcm = 0;**

**}**

**}**

**else {**

**if (start\_element->Parent->Parent == root) {**

**rcm = 1;**

**}**

**//std::cout << "right turn initiated from element: " << start\_element->Key << std::endl;**

**new\_centre = RightTurn(start\_element->Parent->Parent);**

**if (rcm == 1) {**

**// std::cout << "rcm" << std::endl;**

**root = new\_centre;**

**rcm = 0;**

**}**

**}**

**new\_centre->Colour = BLACK;**

**new\_centre->Left->Colour = RED;**

**new\_centre->Right->Colour = RED;**

**}**

**else {**

**if (!start\_element->Isleft()) {**

**//std::cout << "right long turn initiated from element: " << start\_element->Key << std::endl;**

**if (start\_element->Parent->Parent == root) {**

**rcm = 1;**

**}**

**new\_centre = RightTurn(start\_element->Parent);**

**new\_centre = LeftTurn(start\_element->Parent);**

**if (rcm == 1) {**

**// std::cout << "rcm" << std::endl;**

**root = new\_centre;**

**rcm = 0;**

**}**

**}**

**else {**

**if (start\_element->Parent->Parent == root) {**

**rcm = 1;**

**}**

**//std::cout << "left turn initiated from element: " << start\_element->Key << std::endl;**

**new\_centre = LeftTurn(start\_element->Parent->Parent);**

**if (rcm == 1) {**

**// std::cout << "rcm" << std::endl;**

**root = new\_centre;**

**rcm = 0;**

**}**

**}**

**new\_centre->Colour = BLACK;**

**new\_centre->Left->Colour = RED;**

**new\_centre->Right->Colour = RED;**

**}**

**}**

**else {**

**start\_element->Parent->Parent->Colour = RED;**

**start\_element->Parent->Parent->Left->Colour = BLACK;**

**start\_element->Parent->Parent->Right->Colour = BLACK;**

**root = AISreballance(start\_element->Parent->Parent,root);**

**}**

**}**

**else {**

**if (start\_element->Parent->Parent == nullptr) {**

**start\_element->Parent->Colour = BLACK;**

**}**

**else {**

**start\_element->Parent->Parent->Colour = RED;**

**start\_element->Parent->Colour = BLACK;**

**}**

**}**

**}**

**else {**

**//std::cout << "root: " << root->Key << std::endl;**

**return root;**

**}**

**}**

**else {**

**//std::cout << "root: " << root->Key << std::endl;**

**return root;**

**}**

**//std::cout << "root: " << root->Key << std::endl;**

**return root;**

**}**

**int HeigthCounter(TRBNode\* start) {**

**int counter = 0;**

**while (!start->Islist()) {**

**if (start->Colour == BLACK) {**

**++counter;**

**}**

**start = start->Right;**

**}**

**return counter;**

**}**

**int HeigthChecker(TRBNode\* start,int maxheigth,int heigth) {**

**if (start->Islist() && heigth!=maxheigth) {**

**return -1;**

**}**

**else**

**if (!start->Islist() && ((start->Parent != nullptr && !(start->Parent->Right == start || start->Parent->Left== start))||(start->Left->Parent!=start)||(start->Right->Parent!=start))) {**

**return -1;**

**}**

**else {**

**int nextheigth = heigth;**

**if (start->Colour == BLACK) {**

**nextheigth++;**

**}**

**if (!start->Islist()) {**

**if (HeigthChecker(start->Right, maxheigth, nextheigth) != -1 && HeigthChecker(start->Left, maxheigth, nextheigth) != -1) {**

**return 0;**

**}**

**else {**

**return -1;**

**}**

**}**

**else {**

**return 0;**

**}**

**}**

**}**

**//функции загрузки дерева из файла**

**TRBNode\* LoadNode(std::ifstream& fin,TRBNode\* pst) { //загрузка узла**

**int siz[1]; siz[0] = 0;**

**unsigned long long val[1]; val[0] = 0;**

**char ch[1]; ch[0] = 0;**

**try {**

**fin.read((char\*)&siz[0], sizeof(int) / sizeof(char));**

**if(siz[0]<0||siz[0]>257) {**

**throw -1;**

**}**

**char\* stri = (char\*)calloc(siz[0]+1,sizeof(char));**

**if(stri == nullptr) {**

**throw -1;**

**}**

**fin.read(&stri[0], siz[0]);**

**//std::cout << siz[0] << std::endl;**

**stri[siz[0]] = 0;**

**fin.read((char\*)&val[0], sizeof(unsigned long long) / sizeof(char));**

**fin.read(&ch[0], 1);**

**TRBNode \*ND = new(TRBNode);**

**if(ND->Key == nullptr) { //ошибка аллокации**

**free(stri);**

**delete ND;**

**throw -1;**

**}**

**ND->Key = strcpy(ND->Key,stri);**

**free(stri);**

**ND->Value = val[0];**

**ND->Colour = ch[0];**

**ND->Parent = pst;**

**if (siz[0] == 0) {**

**delete ND;**

**return nullptr;**

**}**

**return ND;**

**}**

**catch (std::ios\_base::failure& fail) {**

**throw -1;**

**}**

**catch (std::bad\_alloc()) {**

**throw -1;**

**}**

**}**

**//функции уничтожения дерева**

**void DestroyNode(TRBNode\* dyb) {**

**delete dyb;**

**}**

**void DestroyTree(TRBNode\* root) {**

**if (root != nullptr) {**

**//printf(" || %s - %llu\n",root->Key,root->Value);**

**if (root->Left != nullptr) {**

**DestroyTree(root->Left);**

**root->Left = nullptr;**

**}**

**if (root->Right != nullptr) {**

**DestroyTree(root->Right);**

**root->Right = nullptr;**

**}**

**DestroyNode(root);**

**}**

**}**

**TRBNode\* LoadTree(std::ifstream& fin, TRBNode\* root) { //загрузка всего дерева**

**TRBNode\* croot;**

**try {**

**croot = LoadNode(fin, root);**

**}**

**catch (int err) {**

**return nullptr;**

**}**

**if (croot == nullptr) {**

**//std::cout << "nullptr detected\n";**

**try {**

**croot = new(TRBNode);**

**}**

**catch (std::bad\_alloc()) {//ошибка аллокации**

**return nullptr;**

**}**

**if(croot->Key == nullptr) {**

**delete croot;**

**return nullptr;**

**}**

**croot->Key = strcpy(croot->Key,"");**

**croot->Value = 0;**

**croot->Colour = BLACK;**

**croot->Parent = root;**

**croot->Left = nullptr;**

**croot->Right = nullptr;**

**return croot;**

**}**

**croot->Left = LoadTree(fin, croot);**

**croot->Right = LoadTree(fin, croot);**

**if(croot->Left == nullptr || croot->Right == nullptr) { //если произошла ошибка, удаляем все дерево и возвращаем nullptr**

**if(croot->Left != nullptr) {**

**DestroyTree(croot->Left);**

**croot->Left = nullptr;**

**}**

**if(croot->Right != nullptr) {**

**DestroyTree(croot->Right);**

**croot->Right = nullptr;**

**}**

**delete croot;**

**return nullptr;**

**}**

**return croot;**

**}**

**//функции сохраниения дерева**

**int SaveNode(std::ofstream& fout, TRBNode\* ldb) { //сохранение одного узла**

**try {**

**int size = strlen(ldb->Key);**

**fout.write((char\*)&size, sizeof(size));**

**fout.write(ldb->Key, size);**

**fout.write((char\*)&ldb->Value, sizeof(unsigned long long));**

**fout.write(&ldb->Colour, 1);**

**}**

**catch (std::ios\_base::failure& fail) {**

**return -1;**

**}**

**return 0;**

**}**

**int SaveTree(std::ofstream& fout, TRBNode\* root) { //рекурсивная ф-я по дереву**

**if (root == nullptr) {**

**return 0;**

**}**

**if (SaveNode(fout, root) != 0) {**

**return -1;**

**}**

**if (root->Left != nullptr) {**

**if (SaveTree(fout, root->Left) != 0) {**

**return -1;**

**}**

**}**

**if (root->Right != nullptr) {**

**if (SaveTree(fout, root->Right) != 0) {**

**return -1;**

**}**

**}**

**return 0; //в случчае успех возвращает 0**

**}**

**class TRBTree {**

**public:**

**~TRBTree () {**

**DestroyTree(TreeRoot);**

**}**

**int Add(char\* key,unsigned long long& value){**

**if(key == nullptr ) {**

**return -3;**

**}**

**if (TreeRoot == nullptr) {**

**try {**

**TreeRoot = new(TRBNode);**

**}**

**catch(std::bad\_alloc()) {**

**TreeRoot = nullptr;**

**return -1;**

**}**

**if(TreeRoot->Key == nullptr) {**

**TreeRoot = nullptr;**

**return -1;**

**}**

**TreeRoot->Colour = BLACK;**

**TreeRoot->Parent = nullptr;**

**TreeRoot->Left = nullptr;**

**TreeRoot->Right = nullptr;**

**TreeRoot->Key = strcpy(TreeRoot->Key,key);**

**TreeRoot->Value = value;**

**try {**

**TreeRoot->Right = new(TRBNode);**

**}**

**catch(std::bad\_alloc()) {**

**delete TreeRoot;**

**TreeRoot = nullptr;**

**return -1;**

**}**

**if (TreeRoot->Right->Key == nullptr) {**

**delete TreeRoot->Right;**

**delete TreeRoot;**

**TreeRoot = nullptr;**

**return -1;**

**}**

**TreeRoot->Right->Parent = TreeRoot;**

**try {**

**TreeRoot->Left = new(TRBNode);**

**}**

**catch(std::bad\_alloc()) { //зааллочился ли узел**

**delete TreeRoot->Right;**

**delete TreeRoot;**

**TreeRoot = nullptr;**

**return -1;**

**}**

**if (TreeRoot->Left->Key == nullptr) {//зааллочилась ли строка в узле**

**delete TreeRoot->Right;**

**delete TreeRoot->Left;**

**delete TreeRoot;**

**TreeRoot = nullptr;**

**return -1;**

**}**

**TreeRoot->Left->Parent = TreeRoot;**

**}**

**else {**

**TRBNode\* last = nullptr;**

**TRBNode\* cur = TreeRoot;**

**//поиск места для вставки**

**while (!cur->Islist())**

**{**

**if(cur->Key == nullptr) {**

**return -4;**

**}**

**if (strequal(key, cur->Key) == 0) {**

**return -7;**

**}**

**if (strequal(key, cur->Key) > 0) {**

**last = cur;**

**cur = cur->Right;**

**}**

**else**

**if (strequal(key, cur->Key) < 0) {**

**last = cur;**

**cur = cur->Left;**

**}**

**}**

**//вставка**

**cur->Parent = last;**

**cur->Key = strcpy(cur->Key,key);**

**cur->Value = value;**

**cur->Colour = RED;**

**//std::cout << "cur: " << cur->Key << " par: " << cur->Parent->Key <<std::endl;**

**//std::cout << "parent\_cons: " << cur->Parent->Left->Key << " " << cur->Parent->Right->Key << std::endl;**

**try {**

**cur->Right = new TRBNode;**

**}**

**catch (std::bad\_alloc()) {**

**cur->Key = strcpy(cur->Key,"");**

**cur->Value = 0;**

**cur->Colour = BLACK;**

**cur->Right = nullptr;**

**cur->Left = nullptr;**

**return -1;**

**}**

**try {**

**cur->Left = new TRBNode;**

**}**

**catch (std::bad\_alloc()) {**

**delete cur->Right;**

**cur->Key = strcpy(cur->Key,"");**

**cur->Value = 0;**

**cur->Colour = BLACK;**

**cur->Right = nullptr;**

**cur->Left = nullptr;**

**return -1;**

**}**

**if(cur->Right->Key == nullptr || cur->Left->Key == nullptr) { //ошибка аллокации строк**

**if(cur->Left != nullptr) {**

**delete cur->Left;**

**}**

**if(cur->Right != nullptr) {**

**delete cur->Right;**

**}**

**cur->Key = strcpy(cur->Key,"");**

**cur->Value = 0;**

**cur->Colour = BLACK;**

**cur->Right = nullptr;**

**cur->Left = nullptr;**

**return -1;**

**}**

**cur->Right->Parent = cur;**

**cur->Left->Parent = cur;**

**//балансировка**

**//std::cout << "root was: " << TreeRoot->Key << std::endl;**

**TreeRoot = AISreballance(cur,TreeRoot);**

**//std::cout << "root became: " << TreeRoot->Key << std::endl;**

**}**

**return 0;**

**}**

**int Remove(char\* inpkey) {**

**//std::cout << "removing..." << std::endl;**

**//TRBNode\* last = nullptr;**

**if(inpkey == nullptr) {**

**return -8;**

**}**

**TRBNode\* cur = TreeRoot;**

**if (cur == nullptr) {**

**return -8;**

**}**

**while (true) {**

**//std::cout << "searching... " <<cur->Key<< std::endl;**

**if (cur->Key == nullptr) {**

**return -1;**

**}**

**if (cur->Islist()) {**

**return -8;**

**}**

**if (strequal(inpkey, cur->Key) == 0) {**

**break;**

**}**

**else**

**if (strequal(inpkey, cur->Key) > 0) {**

**//last = cur;**

**cur = cur->Right;**

**}**

**else**

**if (strequal(inpkey, cur->Key) < 0) {**

**//last = cur;**

**cur = cur->Left;**

**}**

**}**

**//std::cout << "element found " << cur->Key<< std::endl;**

**if (cur->Left->Islist() && cur->Right->Islist()) {**

**//std::cout << "removing both-ended" << std::endl;**

**if (cur == TreeRoot) {**

**TreeRoot = nullptr;**

**delete cur->Left;**

**delete cur->Right;**

**delete cur;**

**return 0;**

**}**

**else**

**if (cur->Colour == RED) {**

**if (cur->Isleft()) {**

**cur->Left->Parent = cur->Parent;**

**cur->Parent->Left = cur->Left;**

**}**

**else {**

**cur->Left->Parent = cur->Parent;**

**cur->Parent->Right = cur->Left;**

**}**

**delete cur->Right;**

**delete cur;**

**}**

**else {**

**if (cur->Isleft()) {**

**if(cur->Parent!=nullptr)**

**cur->Parent->Left = cur->Left;**

**cur->Left->Parent = cur->Parent;**

**TRBNode\* start = cur->Left;**

**delete cur->Right;**

**delete cur;**

**TreeRoot = ARMreballance(start, TreeRoot);**

**}**

**else {**

**if (cur->Parent != nullptr)**

**cur->Parent->Right = cur->Left;**

**cur->Left->Parent = cur->Parent;**

**TRBNode\* start = cur->Left;**

**delete cur->Right;**

**delete cur;**

**TreeRoot = ARMreballance(start, TreeRoot);**

**}**

**}**

**}**

**else**

**if (cur->Left->Islist()) {**

**//std::cout << "removing Left-ended" << std::endl;**

**if (cur == TreeRoot) {**

**//std::cout << "removing root" << std::endl;**

**TreeRoot = cur->Right;**

**TreeRoot->Colour = BLACK;**

**TreeRoot->Parent = nullptr;**

**delete cur->Left;**

**delete cur;**

**return 0;**

**}**

**if (cur->Colour == RED) {**

**if (cur->Isleft()) {**

**cur->Right->Parent = cur->Parent;**

**cur->Parent->Left = cur->Right;**

**}**

**else {**

**cur->Right->Parent = cur->Parent;**

**cur->Parent->Right = cur->Right;**

**}**

**delete cur->Left;**

**delete cur;**

**}**

**else {**

**if (cur->Right->Colour == RED) {**

**if (cur->Isleft()) {**

**cur->Right->Parent = cur->Parent;**

**cur->Right->Colour = BLACK;**

**cur->Parent->Left = cur->Right;**

**}**

**else {**

**cur->Right->Parent = cur->Parent;**

**cur->Right->Colour = BLACK;**

**cur->Parent->Right = cur->Right;**

**}**

**delete cur->Left;**

**delete cur;**

**}**

**else {**

**//такого не может быть**

**return -2;**

**}**

**}**

**}**

**else**

**if (cur->Right->Islist()) {**

**//std::cout << "removing Right-ended" << std::endl;**

**if (cur == TreeRoot) {**

**//std::cout << "removing root" << std::endl;**

**TreeRoot = cur->Left;**

**TreeRoot->Colour = BLACK;**

**TreeRoot->Parent = nullptr;**

**delete cur->Right;**

**delete cur;**

**return 0;**

**}**

**if (cur->Colour == RED) {**

**//значит потомки черные**

**if (cur->Isleft()) {**

**cur->Left->Parent = cur->Parent;**

**cur->Parent->Left = cur->Left;**

**}**

**else {**

**cur->Left->Parent = cur->Parent;**

**cur->Parent->Right = cur->Left;**

**}**

**delete cur->Right;**

**delete cur;**

**}**

**else {**

**if (cur->Left->Colour == RED) {**

**if (cur->Isleft()) {**

**cur->Left->Parent = cur->Parent;**

**cur->Left->Colour = BLACK;**

**cur->Parent->Left = cur->Left;**

**}**

**else {**

**cur->Left->Parent = cur->Parent;**

**cur->Left->Colour = BLACK;**

**cur->Parent->Right = cur->Left;**

**}**

**delete cur->Right;**

**delete cur;**

**}**

**else {**

**//такого не может быть**

**return -2;**

**}**

**}**

**}**

**else {**

**// либо оба потомка красные либо оба - черные**

**//std::cout << "removing mud" << std::endl;**

**TRBNode\* cnb = cur->Right;**

**while (!cnb->Left->Islist()) {**

**cnb = cnb->Left;**

**}**

**try {**

**Swap(cur, cnb);**

**}**

**catch (int err) {**

**return -1;**

**}**

**//std::cout << "swap complete. removing... " <<cnb->Key<< std::endl;**

**cur = cnb;**

**//повторенный из начала код**

**if (cur->Left->Islist() && cur->Right->Islist()) {**

**//std::cout << "removing both-ended" << std::endl;**

**if (cur == TreeRoot) {**

**TreeRoot = nullptr;**

**delete cur->Left;**

**delete cur->Right;**

**delete cur;**

**return 0;**

**}**

**else**

**if (cur->Colour == RED) {**

**if (cur->Isleft()) {**

**cur->Left->Parent = cur->Parent;**

**cur->Parent->Left = cur->Left;**

**}**

**else {**

**cur->Left->Parent = cur->Parent;**

**cur->Parent->Right = cur->Left;**

**}**

**delete cur->Right;**

**delete cur;**

**}**

**else {**

**if (cur->Isleft()) {**

**if(cur->Parent!=nullptr)**

**cur->Parent->Left = cur->Left;**

**cur->Left->Parent = cur->Parent;**

**TRBNode\* start = cur->Left;**

**delete cur->Right;**

**delete cur;**

**TreeRoot = ARMreballance(start, TreeRoot);**

**}**

**else {**

**if (cur->Parent != nullptr)**

**cur->Parent->Right = cur->Left;**

**cur->Left->Parent = cur->Parent;**

**TRBNode\* start = cur->Left;**

**delete cur->Right;**

**delete cur;**

**TreeRoot = ARMreballance(start, TreeRoot);**

**}**

**}**

**}**

**else**

**if (cur->Left->Islist()) {**

**//std::cout << "removing Left-ended" << std::endl;**

**if (cur == TreeRoot) {**

**//std::cout << "removing root" << std::endl;**

**TreeRoot = cur->Right;**

**TreeRoot->Colour = BLACK;**

**TreeRoot->Parent = nullptr;**

**delete cur->Left;**

**delete cur;**

**return 0;**

**}**

**if (cur->Colour == RED) {**

**if (cur->Isleft()) {**

**cur->Right->Parent = cur->Parent;**

**cur->Parent->Left = cur->Right;**

**}**

**else {**

**cur->Right->Parent = cur->Parent;**

**cur->Parent->Right = cur->Right;**

**}**

**delete cur->Left;**

**delete cur;**

**}**

**else {**

**if (cur->Right->Colour == RED) {**

**if (cur->Isleft()) {**

**cur->Right->Parent = cur->Parent;**

**cur->Right->Colour = BLACK;**

**cur->Parent->Left = cur->Right;**

**}**

**else {**

**cur->Right->Parent = cur->Parent;**

**cur->Right->Colour = BLACK;**

**cur->Parent->Right = cur->Right;**

**}**

**delete cur->Left;**

**delete cur;**

**}**

**else {**

**//такого не может быть**

**return -2;**

**}**

**}**

**}**

**else**

**if (cur->Right->Islist()) {**

**//std::cout << "removing Right-ended" << std::endl;**

**if (cur == TreeRoot) {**

**//std::cout << "removing root" << std::endl;**

**TreeRoot = cur->Left;**

**TreeRoot->Colour = BLACK;**

**TreeRoot->Parent = nullptr;**

**delete cur->Right;**

**delete cur;**

**return 0;**

**}**

**if (cur->Colour == RED) {**

**//значит потомки черные**

**if (cur->Isleft()) {**

**cur->Left->Parent = cur->Parent;**

**cur->Parent->Left = cur->Left;**

**}**

**else {**

**cur->Left->Parent = cur->Parent;**

**cur->Parent->Right = cur->Left;**

**}**

**delete cur->Right;**

**delete cur;**

**}**

**else {**

**if (cur->Left->Colour == RED) {**

**if (cur->Isleft()) {**

**cur->Left->Parent = cur->Parent;**

**cur->Left->Colour = BLACK;**

**cur->Parent->Left = cur->Left;**

**}**

**else {**

**cur->Left->Parent = cur->Parent;**

**cur->Left->Colour = BLACK;**

**cur->Parent->Right = cur->Left;**

**}**

**delete cur->Right;**

**delete cur;**

**}**

**else {**

**//такого не может быть**

**return -2;**

**}**

**}**

**}**

**}**

**return 0;**

**}**

**TRBNode\* Max() {**

**TRBNode\* scb = TreeRoot;**

**if (scb == nullptr) {**

**return nullptr;**

**}**

**while (!scb->Right->Islist()) {**

**scb = scb->Right;**

**}**

**return scb;**

**}**

**TRBNode\* Min() {**

**TRBNode\* scb = TreeRoot;**

**if (scb == nullptr) {**

**return nullptr;**

**}**

**while (!scb->Left->Islist()) {**

**scb = scb->Left;**

**}**

**return scb;**

**}**

**TRBNode\* Root() {**

**return TreeRoot;**

**}**

**void Print() {**

**std::cout << "print started\n";**

**Tree\_print(TreeRoot,0);**

**std::cout << "print finished\n";**

**}**

**int ChekBlackHeighth()**

**{**

**if (TreeRoot == nullptr) {**

**return 0;**

**}**

**if (HeigthChecker(TreeRoot, HeigthCounter(TreeRoot), 0) == -1) {**

**return -1;**

**}**

**else {**

**return 0;**

**}**

**}**

**TRBNode\* Find(char\* inpkey) {**

**if (inpkey == nullptr) {**

**return nullptr;**

**}**

**TRBNode\* cur = TreeRoot;**

**if(cur == nullptr) {**

**return nullptr;**

**}**

**while (true) {**

**//std::cout << "searching... " <<cur->Key<< std::endl;**

**if (cur->Right == nullptr || cur->Left == nullptr) {**

**return nullptr;**

**}**

**if (cur->Key == nullptr) {**

**return nullptr;**

**}**

**if (strequal(inpkey, cur->Key) == 0) {**

**break;**

**}**

**else**

**if (strequal(inpkey, cur->Key) > 0) {**

**//last = cur;**

**cur = cur->Right;**

**}**

**else**

**if (strequal(inpkey, cur->Key) < 0) {**

**//last = cur;**

**cur = cur->Left;**

**}**

**}**

**return cur;**

**}**

**int SaveToDisk(const std::string& path) {**

**try {**

**std::ofstream fout;**

**fout.open(path, std::ios::binary | std::ios::out);**

**if(!fout.is\_open()) {**

**return -1;**

**}**

**char mark[3] = { 'r','b','t' };**

**fout.write((char\*)&mark[0], sizeof(char) \* 3);**

**if (SaveTree(fout, TreeRoot) != 0) {**

**fout.close();**

**//ошибка при сохранении**

**return -2;**

**}**

**fout.close();**

**}**

**catch (std::ios\_base::failure& fail) {**

**//ошибка с доступом к файлу**

**return -3;**

**}**

**return 0;**

**}**

**int LoadFromDisk(const std::string& path) {**

**char mark[4];**

**std::ifstream fin;**

**try {**

**fin = std::ifstream(path, std::ios::binary | std::ios::in);**

**if(!fin.is\_open()) {**

**return -3;**

**}**

**fin.read((char\*)mark, sizeof(char) \* 3);**

**}**

**catch (std::ios\_base::failure& fail) {**

**//ошибка с доступом к файлу**

**fin.close();**

**return -3;**

**}**

**if (!(mark[0] == 'r' && mark[1] == 'b' && mark[2] == 't'))**

**{**

**//в файле нет маркировки формата**

**fin.close();**

**return -2;**

**}**

**TRBNode \*NewRoot = nullptr;**

**NewRoot = LoadTree(fin, NewRoot);**

**if (NewRoot != nullptr) {**

**//дерево успешно загрузилось**

**fin.close();**

**DestroyTree(TreeRoot);**

**TreeRoot = NewRoot;**

**return 0;**

**}**

**//дерево не загрузилось**

**DestroyTree(NewRoot);**

**fin.close();**

**return -1;**

**}**

**void Destroy() {**

**DestroyTree(TreeRoot);**

**TreeRoot = nullptr;**

**}**

**private:**

**TRBNode\* TreeRoot = nullptr;**

**};**

**int main() {**

**//std::ios\_base::sync\_with\_stdio(false);**

**// //std::cin.tie(nullptr);**

**// long long counter = 0;**

**TRBTree maintree;**

**char \*inpstr = (char\*)calloc(260,sizeof(char));**

**char \*word = (char\*)calloc(260,sizeof(char));**

**unsigned long long inpval;**

**if(inpstr == NULL || inpstr == nullptr || word == NULL || word == nullptr) {**

**printf("ERROR: allocation error\n");**

**return -1;**

**}**

**while (scanf("%s", inpstr) != EOF) {**

**//maintree.Print();**

**// ++counter;**

**// if (counter > 9) {**

**// std::cout<<"break here\n";**

**// }**

**if(strlen(inpstr)>256) {**

**printf("ERROR: uncorrect input\n");**

**continue;**

**}**

**if (inpstr[0] == '+') {**

**if(scanf("%s %llu", word, &inpval) == EOF) {**

**break;**

**}**

**if(strlen(word)>256) {**

**printf("ERROR: uncorrect input\n");**

**continue;**

**}**

**int mrk = maintree.Add(word, inpval);**

**if (mrk == 0) {**

**printf("OK\n");**

**}**

**else {**

**if (mrk == -7) {**

**printf("Exist\n");**

**}**

**else**

**if (mrk == -3) {**

**printf("ERROR: empty input\n");**

**}**

**else**

**if (mrk == -4) {**

**printf("ERROR: untraced allocation error\n");**

**}**

**else**

**if (mrk == -1) {**

**printf("ERROR: out of memory\n");**

**}**

**else {**

**printf("ERROR: unknown error\n");**

**}**

**}**

**}**

**else**

**if (inpstr[0] == '-') {**

**if(scanf("%s", word) == EOF) {**

**break;**

**}**

**if(strlen(word)>256) {**

**printf("ERROR: uncorrect input\n");**

**continue;**

**}**

**int mrk = maintree.Remove(word);**

**if (mrk == 0) {**

**printf("OK\n");**

**}**

**else**

**if (mrk == -8) {**

**printf("NoSuchWord\n");**

**}**

**else**

**if (mrk == -1) {**

**printf("ERROR: out of memory\n");**

**}**

**else {**

**printf("ERROR: unknown error\n");**

**}**

**}**

**else**

**if (inpstr[0] == '!') {**

**std::string path;**

**if(scanf("%s", word) == EOF) {**

**break;**

**}**

**if (strcmp(word, "Save") == 0) {**

**std::cin >> path;**

**int mrk = maintree.SaveToDisk(path);**

**if (mrk == 0) {**

**printf("OK\n");**

**continue;**

**}**

**else**

**if (mrk == 1) {//сохранили пустое дерево**

**printf("OK\n");**

**continue;**

**}**

**else**

**if (mrk == -1) {**

**printf("ERROR: unable to open file\n");**

**continue;**

**}**

**else**

**if (mrk == -2) {**

**printf("ERROR: unable to write file\n");**

**continue;**

**}**

**else**

**if (mrk == -3) {**

**printf("ERROR: file acsess error\n");**

**continue;**

**}**

**else**

**{**

**printf("ERROR: something gone wrong\n");**

**continue;**

**}**

**}**

**else**

**if (strcmp(word, "Load") == 0) {**

**std::cin >> path;**

**int mrk = maintree.LoadFromDisk(path);**

**if (mrk == 0) {**

**printf("OK\n");**

**continue;**

**}**

**else**

**if (mrk == -1) {**

**printf("ERROR: file is damaged\n");**

**continue;**

**}**

**else**

**if (mrk == -2) {**

**printf("ERROR: wrong format of file\n");**

**continue;**

**}**

**else**

**if (mrk == -3) {**

**printf("ERROR: file acsess error\n");**

**continue;**

**}**

**else**

**{**

**printf("ERROR: something gone wrong\n");**

**continue;**

**}**

**}**

**}**

**else {**

**TRBNode\* res = maintree.Find(inpstr);**

**if (res != nullptr) {**

**printf("OK: %llu\n",res->Value);**

**}**

**else {**

**printf("NoSuchWord\n");**

**}**

**}**

**}**

**maintree.Destroy();**

**free(inpstr);**

**free(word);**

**return 0;**

**}**

**Вывод**

Выполнив эту лабораторную работу я поближе познакомился с различными сбалансированными деревьями. Работа с красно-черным деревом позволила мне хоть немного погрузиться в волшебный процесс балансировки, а также испытать неподдельное восхищение при виде самостоятельно отсортировавшегося по алфавитному порядку словаря. Однозначно полезный и весьма интересный опыт продолжает поступать ко мне от изощренных тестов чекера. Поиск ошибок без возможности наблюдать хотя бы вывод программы хоть и приносит уйму негативных эмоций, однако позволяет лучше понять, где и какого рода ошибки чаще всего допускаются, а каких ошибок искать не надо, дабы не уйти по ложному пути в бесконечный зацикленный поиск.