**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**КУРСОВАЯ РАБОТА**

**по дисциплине «Алгоритмы и структуры данных».**

Тема: «Случайные БДП – вставка и исключение. Текущий контроль»

Вариант 9.

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. 5383 |  | Допира В. Е. |
| Преподаватель |  | Фирсов М. А. |

Санкт-Петербург

2016

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студентка Допира В. Е. | | |
| Группа 5383 | | |
| Тема работы: «Случайные БДП – вставка и исключение. Текущий контроль» | | |
| Исходные данные: исходная строка с числами типа int | | |
| Содержание пояснительной записки:  «Содержание»  «Введение»  «Спецификация программы»  «Алгоритм и структуры данных»  «Тестирование»  «Заключение»  «Список использованных источников»  «Приложение» | | |
| Предполагаемый объем пояснительной записки:  26 страниц. | | |
| Дата выдачи задания: | | |
| Дата сдачи реферата: | | |
| Дата защиты реферата: | | |
| Студент |  | Допира В.Е. |
| Преподаватель |  | Фирсов М. А. |

**Содержание**

[ВВЕДЕНИЕ](#_Toc467613480) 5

[1. СПЕЦИФИКАЦИЯ ПРОГРАММЫ](#_Toc467613481) 8

[1.1 Входные данные](#_Toc467613482) 8

[1.2 Выходные данные](#_Toc467613484) 9

[1.3 Описание интерфейса пользователя.](#_Toc467613486) 9

[2. АЛГОРИТМ И СТРУКТУРЫ ДАННЫХ](#_Toc467613488) 10

[2.1 Описание алгоритма](#_Toc467613489) 10

[2.2 Функции, использованные в программе](#_Toc467613490) 10

[3. ТЕСТИРОВАНИЕ.](#_Toc467613493) 13

[ЗАКЛЮЧЕНИЕ И ВЫВОД](#_Toc467613497) 17

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ](#_Toc467613497) 18

[ПРИЛОЖЕНИЕ. Текст программы](#_Toc467613497) 19

**Аннотация**

Курсовая работа создает задание для текущего контроля студентов по теме: «Случайные бинарные деревья поиска (БДП)». Для выполнения задач ученику необходимо разобраться с построением деревьев, вставкой и удалением элементов. Результат работы программы позволит проводить удобную проверку. Разработан проект на языке программирования С++ с использованием динамических структур данных и алгоритма быстрого поиска. Составленные задания могут быть использованы в курсе предмета «Алгоритмы и структуры данных».

**Summary**

Course work creates a task to monitor students on the theme: "Random binary search trees (BST)." To perform the tasks the student is necessary to deal with the construction of trees, insert and delete elements. The result of the program will allow for convenient testing. A project was written in the programming language C ++ using dynamic data structures and fast search algorithm. Structured tasks can be used in the subject course "Algorithms and Data Structures."

**введение**

**Вариант 9**

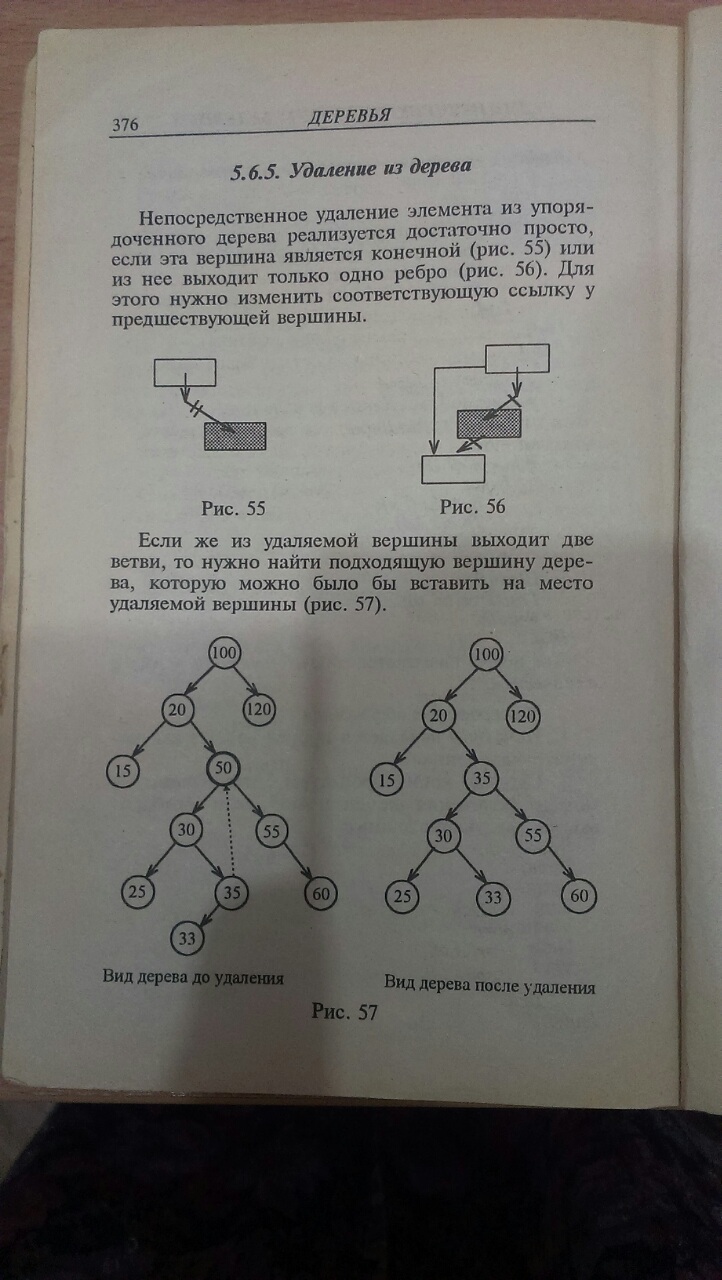
**Цель:** реализация и экспериментальное исследование алгоритма быстрого поиска на основе БДП.

**Задача:** Случайные БДП - вставка и исключение. Текущий контроль.

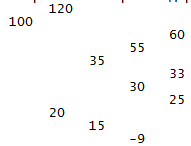
По заданному файлу F (типа file of Elem), все элементы которого различны, построить БДП определённого типа. "Текущий контроль" - создание программы для генерации заданий с ответами к ним для проведения текущего контроля среди студентов. Задания и ответы должны выводиться в файл в удобной форме: тексты заданий должны быть готовы для передачи студентам, проходящим ТК; ответы должны позволять удобную проверку правильности выполнения заданий.

**Формальная постановка задачи**

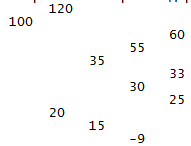
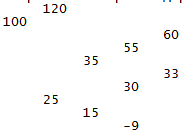
В исходном файле задана строка. Например, 100 120 20 15 35 30 33 25 55 60. Все элементы типа int и неравные. Программа строит по строке случайное БДП, если элемент больше текущего, то записываем его в правую ветвь, если меньше – в левую:



Далее программа генерирует рандомный элемент, который не содержится в дереве, например, -9, и вставляет его в БДП.



Программа также генерирует элемент для удаления, который содержится в дереве и не является листом, то есть имеет поддеревья. Например, 20. На его место встанет самый левый лист (наименьший) в правой ветви:

Создается файл с заданием для студентов по теме случайные бинарные деревья поиска. Задание будет включать построение БДП, вставку и удаление узла. Ответы к заданию сохраняются в отдельный файл.

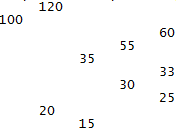
Файл с заданием содержит:

По исходным данным: 100 120 20 15 35 30 33 25 55 60 построить бинарное дерево поиска. Записать элементы дерева по возрастанию.

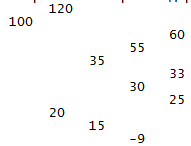
Добавить элемент: -9

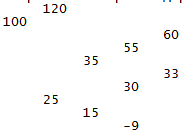
Удалить элемент: 20.

Файл с ответами содержит: построенное БДП

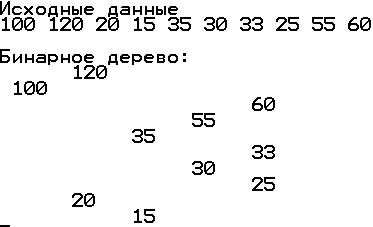
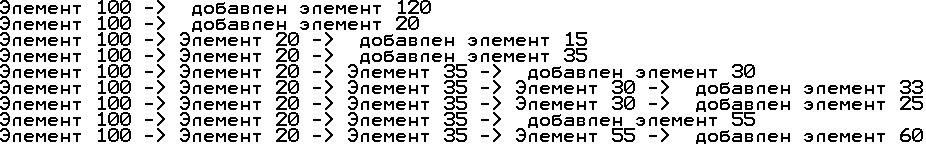


Добавление элемента -9. Построение бинарного дерева поиска:

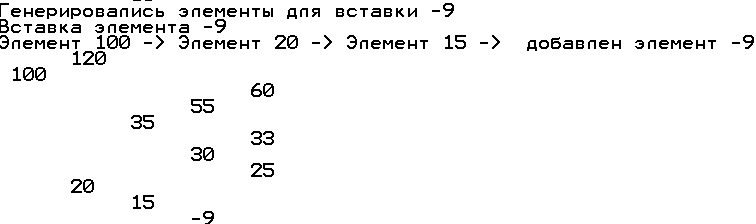


Удаление элемента 20 Построение бинарного дерева поиска: 

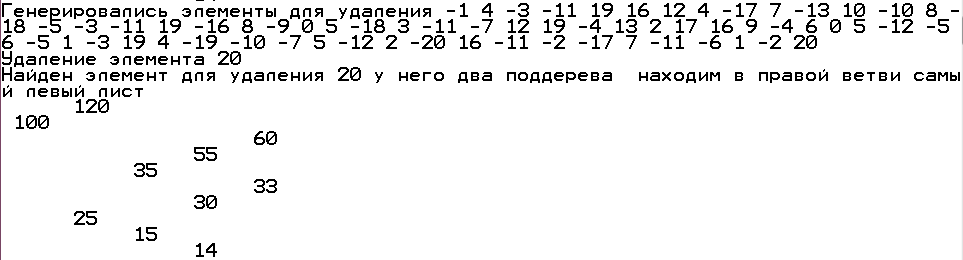
В консоль программа выводит порядок построения дерева:

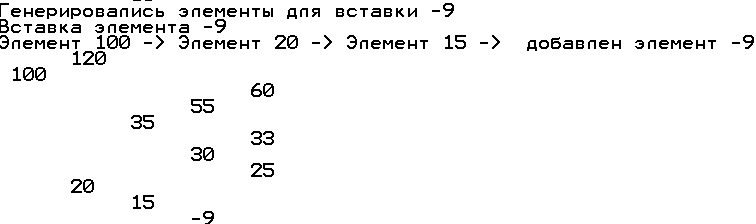


Вставить элемент: порядок его вставки в БДП, новое построенное дерево:



Удалить элемент: новое построенное БДП:





**1. Спецификация программы**

**1.1 Входные данные**

**Структуры данных:** в программе данные хранятся в структуре (динамическая реализация):

struct TreeNode // данные

{

int value; //текущий элемент

TreeNode \*left; //ссылка на левый

TreeNode \*right; //ссылка на правый

TreeNode() { left = NULL; right = NULL; } // constructor

};

Работа с бинарным деревом поиска осуществляется с помощью класса:

class BinaryTree

{

private:

TreeNode \*head;

TreeNode \*link;

public:

TreeNode\* Create();

bool isNull();

void headBT(); //определение вершины дерева

int RootBT(binTree b); // для непустого бин.дерева

TreeNode\* headRootBT( ) ;

binTree Left(binTree b);

binTree Right(binTree b);

void displayBT(binTree, ID );

void displayBT(binTree, ID );

void printTree();

void obhodBT(binTree b); // Обход и вывод по возрастанию

bool findEL(int x , binTree b); //поиск в дереве элемента

TreeNode\* enterBT();

TreeNode\* newElBT(int x);

TreeNode\* enterElBT(int x , binTree b);

void destroyEL(binTree b);

void destroyTree();

TreeNode\* delTree(int x, binTree b);

TreeNode\* delEl(binTree b);

};

Исходный файл содержит строку с элементами типа int.

**1.2 Выходные данные**

Программа создает файл с заданием и ответами к нему. Содержание файлов описано выше.

**1.3 Описание интерфейса пользователя**

Пользователь вводит в файл «in.txt» строку с исходными данными. Программа выводит в консоль результат работы программы с промежуточными выводами. Также создает 2 файла: «task.txt» содержит условие задачи для текущего контроля студентов и «keys.txt» с ответами к нему.

**2. Алгоритм и структуры данных**

**2.1 Описание алгоритма**

Программа выполняет следующую последовательность действий:

Программа создает случайное БДП. Если дерево не пустое, программа рекурсивно строит его. Если элемент больше текущего, то записываем его в правую ветвь, если меньше – в левую.

Далее генерируется случайное число от -20 до 20, которого еще нет в дереве, для вставки элемента в БДП. Программа строит новое БДП.

Генерируется элемент для удаления от -20 до 20, который содержится в дереве и не является листом, то есть имеет поддеревья. У БДП может быть 1 поддерево (правое или левое), может быть 2 поддерева, может являться вершиной (тогда меняем ссылку на голову дерева). На место удаляемого элемента встанет самый левый лист (наименьший) в правой ветви. Если у правой ветви нет элементов слева, то место удаляемого элемента займет следующий справа. Программа строит новое БДП. Далее программа удаляет дерево.

**2.2 Функции, используемые в программе**

TreeNode\* BinaryTree::Create()

Назначение: создание дерева

Возвращаемое значение: head - голова.

bool BinaryTree::isNull()

Назначение: проверка пустое ли дерево

Возвращаемое значение: пустой текущий узел.

void BinaryTree::headBT( )

Назначение: определение вершины дерева

int BinaryTree::RootBT(binTree b )

Назначение: возвращение символа в текущей вершине для непустого бинарного дерева

Параметры: binTree b – бинарное дерево

Возвращаемое значение символ в текущей вершине.

TreeNode\* BinaryTree::headRootBT( )

Назначение: возвращение головы для непустого бинарного дерева

Возвращаемое значение голова на дерево.

binTree BinaryTree::Left(binTree b)

Назначение: переход к левому поддереву для непустого бинарного дерева

Параметры: binTree b – бинарное дерево

Возвращаемое значение переход к левой вершине.

binTree BinaryTree::Right(binTree b)

Назначение: переход к правому поддереву для непустого бинарного дерева

Параметры: binTree b – бинарное дерево

Возвращаемое значение переход к правой вершине.

TreeNode\* BinaryTree::newElBT( int x)

Назначение: ввод листа

Параметры: int x – корень дерева

Возвращаемое значение: лист.

TreeNode\* BinaryTree::enterElBT(int x , binTree b)

Назначение: ввод вершины

Параметры: int x – корень дерева

binTree b– бинарное дерево

Возвращаемое значение дерево.

TreeNode\* BinaryTree::enterBT()

Назначение: ввод дерева из файла

void BinaryTree::displayBT(binTree b, ID n)

Назначение: вывод бинарного дерева на экран

Параметры: binTree b – бинарное дерево

ID n – уровень, на котором находится узел

void BinaryTree::printT(binTree b, unsigned k )

Назначение: вывод бинарного дерева на экран

Параметры: binTree b – бинарное дерево

unsigned k – уровень, на котором находится узел

void BinaryTree::displayData()

Назначение: печать исходной строки

void BinaryTree::printTree()

Назначение: печать бинарного дерева

void BinaryTree::destroyEL(binTree b)

Назначение: удаление элемента в бинарном дереве

Параметры: binTree b – бинарное дерево

void BinaryTree::destroyTree()

Назначение: удаление дерева

bool BinaryTree::sheetCheck(int x, binTree b)

Назначение: проверка на лист

Параметры: int x – корень дерева

binTree b– бинарное дерево

Возвращаемое значение правда или ложь.

TreeNode\* BinaryTree::delTree(int x, binTree b)

Назначение: удаление элемента из БДП

Параметры: int x – корень дерева

binTree b– бинарное дерево

Возвращаемое значение новое дерево.

bool BinaryTree::findEL(int x , binTree b)

Назначение: поиск элемента в БДП

Параметры: int x – корень дерева

binTree b– бинарное дерево

**3. Тестирование**

|  |  |  |
| --- | --- | --- |
| **Входные данные** | | **Результат** |
| **Тесты** | **Задание** | **Полученные ответы** |
| 1 |  | Файл не может быть открыт! |
| 2 | По исходным данным:  6 3 5 0 -8 4 2 9 1 -20 19 -16 построить бинарное дерево поиска.  Добавить элемент: -7  Удалить элемент: 5 | Построение бинарного дерева поиска:    Добавление элемента -7  Построение бинарного дерева поиска:    Удаление элемента 5  Построение бинарного дерева поиска: |
| 3 | По исходным данным:  1 7 9 0 2 построить бинарное дерево поиска.  Добавить элемент: -5  Удалить элемент: 7 | Построение бинарного дерева поиска:    Добавление элемента -5  Построение бинарного дерева поиска:    Удаление элемента 7  Построение бинарного дерева поиска: |
| 4 | По исходным данным: 9 0 2 -6 19 -5 -3 13 -12 -2 3 4 5 -10 11 -11 14 построить бинарное дерево поиска.  Добавить элемент: 18  Удалить элемент: 19 | Построение бинарного дерева поиска:    Добавление элемента 18  Построение бинарного дерева поиска:    Удаление элемента 19  Построение бинарного дерева поиска: |
| 5 | По исходным данным:  1 -7 5 4 -13 построить бинарное дерево поиска.  Добавить элемент: 11  Удалить элемент: 1 | Построение бинарного дерева поиска:    Добавление элемента 11  Построение бинарного дерева поиска:    Удаление элемента 1  Построение бинарного дерева поиска: |
| 6 | По исходным данным: -3 1 -2 19 13 0 56 построить бинарное дерево поиска.  Добавить элемент: -19  Удалить элемент: -2 | Построение бинарного дерева поиска:    Добавление элемента -19  Построение бинарного дерева поиска:    Удаление элемента -2  Построение бинарного дерева поиска: |
| 7 | По исходным данным: 6 3 5 0 -8 4 2 9 1 -20 19 -16 -4 7 построить бинарное дерево поиска.  Добавить элемент: 12  Удалить элемент: -20 | Построение бинарного дерева поиска:    Добавление элемента 12  Построение бинарного дерева поиска:    Удаление элемента -20  Построение бинарного дерева поиска: |

**ЗАКЛЮЧЕНИЕ**

**Вывод**

В ходе данной работы было создано задание для текущего контроля студентов с ответами на языке программирования С++. При выполнении курсовой работы использовались знания, полученные во время лекций и практических занятий по работе с случайными бинарными деревьями поиска и реализации алгоритма быстрого поиска, по работе с динамическими структурами данных, классами.

**список использованных источников**

1. [Кнут, Д. Э.](https://ru.wikipedia.org/wiki/%D0%9A%D0%BD%D1%83%D1%82,_%D0%94%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%B4_%D0%AD%D1%80%D0%B2%D0%B8%D0%BD) [Искусство программирования. Том 3. Сортировка и поиск](https://books.google.ru/books?id=92rW-nktlbgC&printsec=frontcover&dq=editions:spjjKVwoQ3QC&hl=ru&sa=X&ved=0ahUKEwiUjLbc88rQAhVCfiwKHUJlBQoQuwUIIjAB#v=onepage&q&f=false). The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. Ю. В. Козаченко. Москва: Вильямс, 2007. Т. 3.— 832 с.

2. Роман Акопов. [Двоичные деревья поиска](http://www.rsdn.ru/article/alg/binstree.xml). RSDN Magazine #5-2003.

3. Кормен Т., Лейзерсон Ч., Ривест Р.: «Алгоритмы: построение и анализ». Вильямс, 2-е издание, 2005.

4. Роберт Седжвик. Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск = Algorithms in C. Fundamentals/Data Structures/Sorting/Searching. СПб.: ДиаСофтЮП, 2003. С. 672.

5. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ = INTRODUCTION TO ALGORITHMS. 2-е изд. М.: [«Вильямс»](https://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2006. С. 1296.

**приложение**

**Текст программы**

#include<iostream>

#include <fstream>

#include <string.h>

#include <cstdlib>

#include <ctime> // time()

using namespace std;

ifstream in("in.txt"); //файл с исходной строкой

ofstream task("task.txt"); //файл с заданием

ofstream keys("keys.txt"); //файл с ключами

typedef int ID;

int data[255];

int count;

struct TreeNode // данные

{

int value;

TreeNode \*left;

TreeNode \*right;

TreeNode() { left = NULL; right = NULL; } // constructor

};

typedef TreeNode \*binTree; // "представитель" бинарного дерева

binTree q;

class BinaryTree

{

private:

TreeNode \*head;

TreeNode \*link;

public:

TreeNode\* Create();

bool isNull();

void headBT(); //определение вершины дерева

int RootBT(binTree b); // для непустого бин.дерева

TreeNode\* headRootBT( ) ;

binTree Left(binTree b);

binTree Right(binTree b);

void displayData();

void displayBT(binTree, ID );

void printTree();

void printT(binTree b, unsigned k); //Печать дерева

bool findEL(int x , binTree b); //поиск в дереве элемента

TreeNode\* enterBT();

TreeNode\* newElBT(int x);

TreeNode\* enterElBT(int x , binTree b);

void destroyEL(binTree b);

void destroyTree();

bool sheetCheck(int x, binTree b);

TreeNode\* delTree(int x, binTree b);

TreeNode\* delEl(binTree b);

};

//функции

TreeNode\* BinaryTree::Create() //создание дерева

{

link=head=NULL;

return head;

}

bool BinaryTree::isNull()

{

return (link == NULL);

}

void BinaryTree::headBT( ) //определение вершины дерева

{

head=link;

}

int BinaryTree::RootBT(binTree b ) // для непустого бин.дерева

{

if (b == NULL) { cerr << "Error: RootBT(null) \n"; exit(1); } //сообщение об ошибке

else { return b->value; }

}

TreeNode\* BinaryTree::headRootBT( ) // для непустого бин.дерева

{

if (head == NULL) { cerr << "Error: headRootBT(null) \n"; exit(1); } //сообщение об ошибке

else {

return head;

}

}

binTree BinaryTree::Left(binTree b) // для непустого бин.дерева

{

if (b == NULL) { cerr << "Error: Left(null) \n"; exit(1); } //сообщение об ошибке

else return b->left;

}

binTree BinaryTree::Right(binTree b) // для непустого бин.дерева

{

if (b == NULL) { cerr << "Error: Right(null) \n"; exit(1); } //сообщение об ошибке

else return b->right;

}

TreeNode\* BinaryTree::newElBT(int x) //ввод листа

{

binTree p = new TreeNode; //выделение памяти

if (p != NULL) { //для непустого дерева

p->value = x;

p->left = NULL;

p->right = NULL;

return p;

}

else { cerr << "Memory not enough\n"; exit(1); } //сообщение об ошибке

}

TreeNode\* BinaryTree::enterElBT(int x , binTree b) //ввод вершины

{

if (b!=NULL) { cout<<"Элемент "<<b->value<<" -> "; //вывод порядка построения БДП

if (x <= b->value) { b->left=enterElBT(x,b->left); } //влево

else { b->right=enterElBT(x,b->right); } //вправо

}

else {

b = newElBT(x);

cout<<" добавлен элемент "<<b->value << endl;

}

return b;

}

TreeNode\* BinaryTree::enterBT() //ввод дерева из файла

{

int ch;

count=0;

if (!in.is\_open()) // если файл не открыт

cout << "Файл не может быть открыт!\n"; // сообщить об этом

else

{

data[count]=ch; //создание массива для вывода исходной строки

count++;

while(in >> ch )

{

if (!isNull()){ link=headRootBT();

link = enterElBT(ch,headRootBT());}

else {

link = newElBT(ch);

headBT(); }

}

in.close(); //закрытие файла

}

}

void BinaryTree::displayBT(binTree b, ID n)

{ // n - уровень узла

if (b != NULL) { //вывод и построение БДП

cout << ' ' << RootBT(b);

keys << " " << RootBT(b);

if (Right(b)!= NULL) { displayBT(Right(b), n + 1); }

else cout << endl; // вниз

if (Left(b)!= NULL) {

for (int i = 1; i <= n; i++) cout << " "; // вправо

displayBT(Left(b), n + 1);

}

}

else {};

}

void BinaryTree::printT(binTree b, unsigned k )

{

if (b != NULL)

{ printT(b->right, k + 5);

for (int i = 0; i < k; i++) {

cout << " "; keys<<" ";

}

cout << RootBT(b)<< endl;;

keys << RootBT(b)<< endl;

printT(b->left, k + 5);

}

}

void BinaryTree::displayData() //вывод строки исходных данных

{

if (count!= 0) {

cout <<endl<< "Исходные данные " << endl;

task << "По исходным данным: " ;

for (int i = 0; i < count; i++)

{

cout<<data[i]<<" ";

task<<data[i]<<" ";

}

cout<<endl;

task<<" построить бинарное дерево поиска."<<endl;

}

else{};

}

void BinaryTree::printTree() //печать

{

binTree b=headRootBT();

ID n=1;

keys<<endl<<"Построение бинарного дерева поиска: "<<endl;

//displayBT( b, n);

printT(b,n);

}

void BinaryTree::destroyEL(binTree b) //удаление элемента

{

if (b != NULL) {//для непустого дерева

destroyEL(b->left);

destroyEL(b->right);

delete b;

b = NULL;

}

}

void BinaryTree::destroyTree() //удаление дерева

{

destroyEL(headRootBT()); //обращение к функции удаления элемента

}

bool BinaryTree::sheetCheck(int x, binTree b) // проверка на ЛИСТ

{ if (b!=NULL)

{

if (x!=b->value)

if (x < b->value) { sheetCheck(x,b->left); }

else { sheetCheck(x,b->right); }

else

{

if ((Right(b)== NULL) && (Left(b)== NULL)) //нет поддеревьев

return false;

else return true; // одно поддерево

}

}

}

TreeNode\* BinaryTree::delTree(int x, binTree b)

{

if (b!=NULL){ //для непустого дерева

if (x!=b->value) //элемент не корень

if (x < b->value) { b->left=delTree(x,b->left); }//переход влево

else { b->right=delTree(x,b->right); }//переход вправо

else

{

cout<<"Найден элемент для удаления "<<b->value;

q=b;

if (Right(q)== NULL) {

cout<<" у него нет правого поддерева"<<endl;

b=q->left;

}

else

if (Left(q)== NULL) {

cout<<" у него нет левого поддерева"<<endl;

b=q->right;

}

else // два поддерева

{

cout<<" у него два поддерева ";

binTree y=q->right;

if(Left(y) == NULL) // нет у правой ветви левого листа

{

cout<<" но у правой ветви нет левого листа"<<endl;

y->left = q->left;

b=y;

}

else

{

binTree z=y->left; // есть у правой ветви левый лист

while(Left(z) != NULL)

{

y = z;

z = y->left;

}

cout<<" находим в правой ветви самый левый лист"<<endl;

y->left = z->right;

z->left = q->left;

z->right =q->right;

b=z;

}

}

}

}

if (q==head) head=b;//если удаляем вершину, то нужно поменять ссылку на голову дерева

delete q;

return b;

}

bool BinaryTree::findEL(int x , binTree b) //поиск элемента

{

if (b==NULL) { return false;} //дерево пустое

else

if (x==b->value) return true; //элемент найден

else

if (x < b->value){ findEL(x,b->left);}//переход к левому узлу

else { findEL(x,b->right); }//переход к правому узлу

}

int main()

{

setlocale(LC\_ALL, "rus"); // корректное отображение Кириллицы

BinaryTree Bin;

Bin.Create(); //создание дерева

Bin.enterBT(); //чтение из файла

ID depth = 1;

if (Bin.isNull()) cout << "Пустое БД" <<endl;//сообщить, если дерево пустое

else

{

Bin.displayData(); //рекурсивный порядок построения дерева

cout << endl<<"Бинарное дерево: " << endl;

Bin.printTree();//вывод исходной строки

srand(time(NULL));

int addValue;

cout<<"Генерировались элементы для вставки ";

while(true)

{

addValue=rand() % 41 - 20; // Случайное число oт —20 дo 20

cout<<addValue<<" ";

if (!Bin.findEL(addValue,Bin.headRootBT()))

{ cout<< endl<<"Вставка элемента "<<addValue<< endl;

task <<endl<<"Добавить элемент: "<<addValue<< endl;

keys<<endl<<endl<<"Добавление элемента "<<addValue<< endl;

Bin.enterElBT(addValue,Bin.headRootBT());//вставка элемента

Bin.printTree();//вывод нового БДП

break;

}

}

int delValue;

cout<<"Генерировались элементы для удаления ";

while(true)

{ delValue=rand() % 41 - 20; // Случайное число oт —20 дo 20

cout<<delValue<< " ";

if (Bin.findEL(delValue,Bin.headRootBT())) // есть элемент

{

if (Bin.sheetCheck(delValue,Bin.headRootBT())) // есть поддеревья

{ cout<< endl<<"Удаление элемента "<<delValue<<endl;

task<< endl<<"Удалить элемент: "<<delValue<<endl;

keys<<endl<<endl<<"Удаление элемента "<<delValue<<endl;

Bin.delTree(delValue,Bin.headRootBT());//удаление элемента

Bin.printTree();//вывод нового БДП

break;

}

}

}

Bin.destroyTree();//удаление дерева

}

return 0;

}