«Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчёт защищён с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель Бубнова Н.Д.

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021 г.

Отчёт

Лабораторной работе №3

**«**Бинарные деревья**»**

Студент группы ПИ 92 В.М. Шульпов

Старший преподаватель Бубнова Н.Д.

Барнаул 2021

**Задание:**

* Ознакомиться с постановкой задачи
* Написать программу для ее реализации
* Разработать тесты
* Исходные данные поместить в файл input.dat
* Результаты вывести на экран.
* Исходные данные и результаты вывести также в выходной файл output.dat
* Предусмотреть вывод исходного и выходного деревьев в табличной форме или в графике на экран. Вывод в табличной форме означает вывод в прямом порядке, т.е. Корень, Левое поддерево, Правое поддерево.
* Вывести деревья в выходной файл.
* Оформить отчет

**Вариант 27**

Реализовать удаление элемента с заданным значением ключевого признака если он расположен в дереве на заданном уровне.

Структура узла

struct tnode {// Узел дерева

int key;// Ключ

int level;//уровень

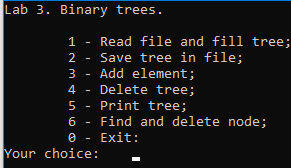
struct tnode\* left;// Левый потомок

struct tnode\* right;// Правый потомок

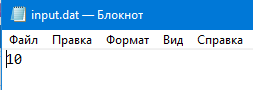
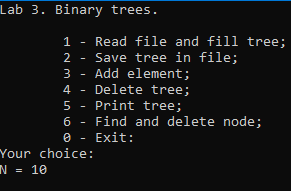
};

Тесты

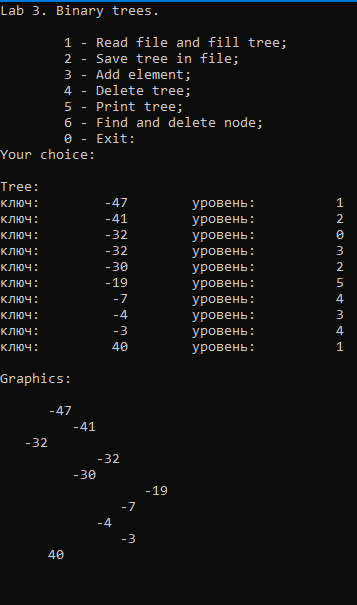
Вход в программу



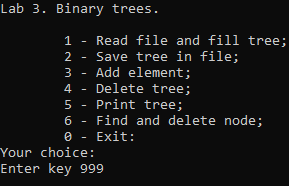
Ввод 1 + enter - Чтение n из файла и заполнение дерева n случайными числами (и вывод сообщения N=… на экран)



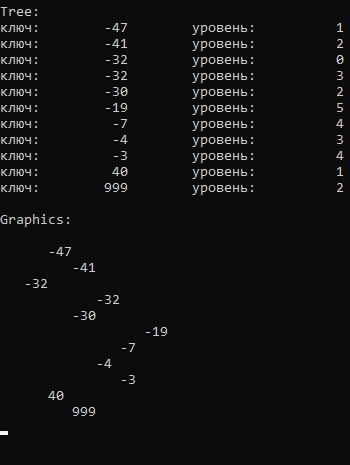
Ввод 5 + enter – вывод дерева на экран в двух вариантах



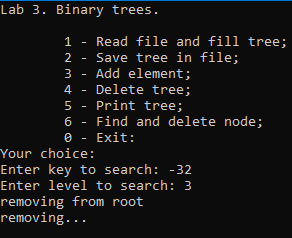
Ввод 3 + enter – добавление элемента (ввёл 999)



Повторно вывожу дерево (добавился элемент)

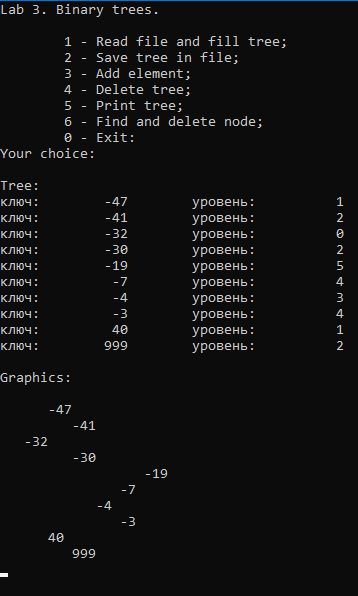


Ввод 6 + enter – удаление элемента из дерева по известному ключу и уровню (ввожу -32, 3)

(removing from root, так как в функции удаления элемента вызывается функция удаления корня, где корень – удаляемый элемент)

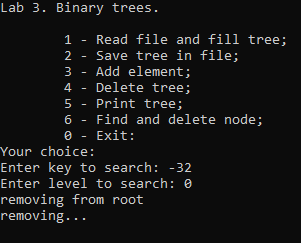
Удалится элемент из конца ветки

Очередной вывод дерева

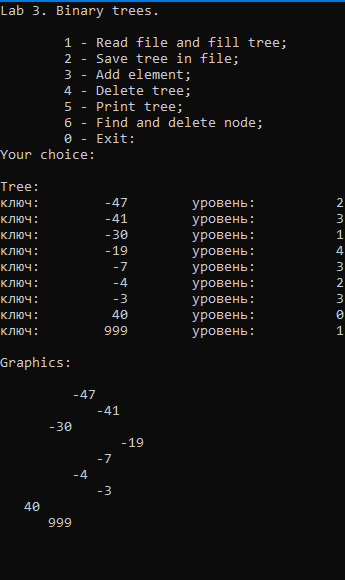


Ввод 6 + enter – удаление элемента из дерева по известному ключу и уровню (ввожу -32, 0)

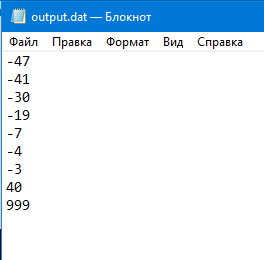
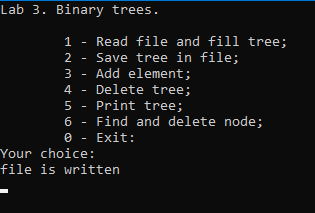
Удаляется элемент из корня



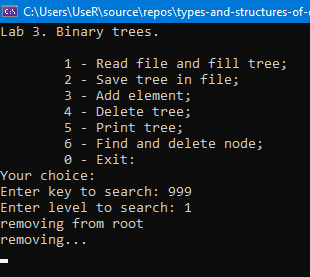
Вывод дерева (новый корень – 40)



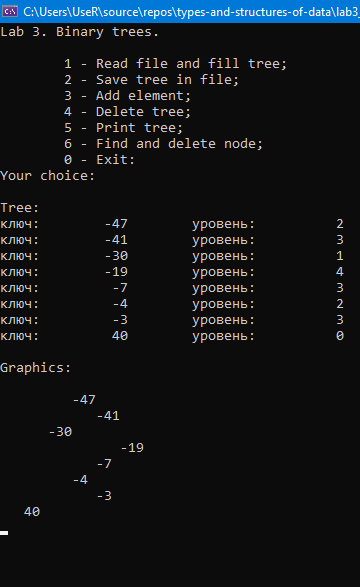
Ввод 2 + enter – сохранения дерева в файл



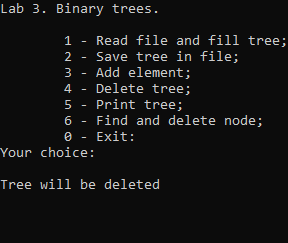
Ещё одно удаление (ввод 999, 1)



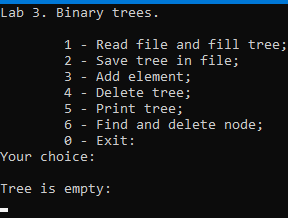
Вывод результата



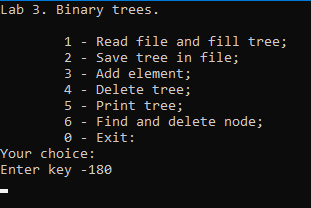
Ввод 4 + enter - Можно полностью почистить дерево (сделать пустым)



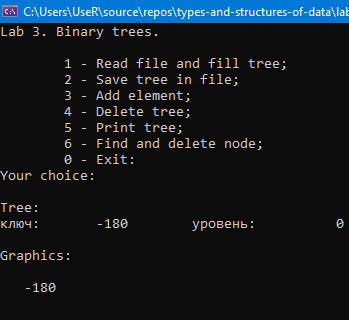
Попытка вывода



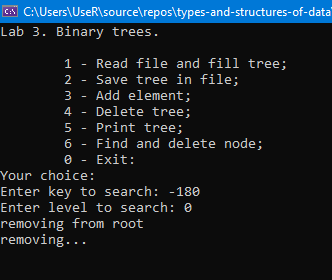
Ввод 3 + enter – ввод в пустое дерево



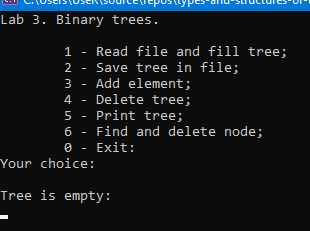
Вывод



Удаление единственного элемента из дерева



Вывод



Код

binaryTree.cpp

#include "binaryTree1.h"

int main(int argc, char\* argv[])

{

srand(time(NULL));

setlocale(LC\_ALL, "Rus");

string in\_filneame = "input.dat";

string out\_filneame = "output.dat";

int n = 0;// Кол-во узлов для записи из файла

int key = 0; // Ключ

int level = 0; // Уровень

char choice; // Выбор

struct tnode\* root;

root = NULL;

do

{

cout << "Lab 3. Binary trees." << endl;

printf("\n\t1 - Read file and fill tree;\n\t2 - Save tree in file;\n\t3 - Add element;\n\t4 - Delete tree;\n\t5 - Print tree;\n\t6 - Find and delete node;\n\t0 - Exit:\nYour choice:\t");

fflush(stdin);

choice = \_getch();

printf("\n");

switch (choice)

{

case '1':

n = readArrayN(in\_filneame);

cout << "N = " << n << endl;

randomAdd\_N\_Nodes(&root, n);

\_getch();

break;

case '2':

clearFile(out\_filneame);

SaveTreeInFile(root, out\_filneame);

cout << "file is written" << endl;

\_getch();

break;

case '3':

while (!(fflush(stdin)) && !(printf("Enter key ") && scanf\_s("%d", &key)));//ввод ключа

root = addNode(root, key);

\_getch();

break;

case '4':

cout << "\nTree will be deleted" << endl;

deleteSubTree(root);

root = NULL;

\_getch();

break;

case '5':

if (root) {

cout << "\nTree:\n";//вывод в прямом порядке

printTree(root);

cout << "\nGraphics:\n\n";//вывод в прямом порядке

printTreeGraphics(root,1);

}

else cout << "\nTree is empty:\n";

\_getch();

break;

case '6':

while (!(fflush(stdin)) && !(printf("Enter key to search: ") && scanf\_s("%d", &key)));//ввод ключа

while (!(fflush(stdin)) && !(printf("Enter level to search: ") && scanf\_s("%d", &level)));//ввод уровня

root = deleteNode(root, key, level);

cout << "removing..." << endl;

\_getch();

break;

}

system("cls");

} while (choice != '0');

system("pause");

return 0;

}

binaryTree1.h

#pragma once

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <conio.h>

#include <fstream>

#include <iostream>

#include <string>

#include <Windows.h>

#include <ctime>

using namespace std;

// Структура узла

struct tnode {// Узел дерева

int key;// Ключ

int level;//уровень

struct tnode\* left;// Левый потомок

struct tnode\* right;// Правый потомок

};

// Сравнение чисел

int numcmp(int a, int b)

{

if (a > b) return 1;

if (a < b) return -1;

if (a == b) return 0;

}

// Функция добавления узла к дереву

struct tnode\* addNode(struct tnode\* node, int key) {

int cond;

if (node == NULL) {

node = new tnode();

node->key = key;

node->level = 0;

node->left = node->right = NULL;

}

else

{

cond = numcmp(key, node->key);

if (cond < 0) {

node->left = addNode(node->left, key);

node->left->level = node->level + 1;

}

else {

node->right = addNode(node->right, key);

node->right->level = node->level + 1;

}

}

return node;

}

// Добавление N узлов к дереву с рандомным значением

void randomAdd\_N\_Nodes(struct tnode\*\* node, int quantity) {

int number;

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

number = rand() % 100 - 50;

\*node=addNode(\*node, number);

}

}

// Функция удаления поддерева

void deleteSubTree(tnode\* node) {

if (node != NULL) {

deleteSubTree(node->left);

deleteSubTree(node->right);

delete node;

}

}

// Функция вывода дерева

void printTree(tnode\* node) {

if (node != NULL) {

printTree(node->left);

printf("ключ: %10d\tуровень: %10d\n", node->key, node->level);

printTree(node->right);

}

}

// Функция графического вывода дерева

void printTreeGraphics(tnode\* node, int level)

{

if (node)

{

printTreeGraphics(node->left, level + 1);

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

cout << node->key << endl;

printTreeGraphics(node->right, level + 1);

}

}

// Изменить уровень (node-level) каждого узла

void changeLevel(tnode\* node, int x) {

if (node != NULL) {

node->level += x;

changeLevel(node->left, x);

changeLevel(node->right, x);

}

}

// Удаление корня поддерева

tnode\* deleteRootSubTree(tnode\* node) {

if (node != NULL) {

cout << "removing from root" << endl;

if (node->left == NULL && node->right == NULL) {

return NULL;

}

if (node->left != NULL && node->right == NULL) {

node = node->left;// Присваиваем корню левую часть дерева (от корня)

changeLevel(node, -1);

}

if (node->left == NULL && node->right != NULL) {

node = node->right;// Присваиваем корню правую часть дерева (от корня)

changeLevel(node, -1);

}

if (node->left != NULL && node->right != NULL) {

int height = 0; // Высота, на которую опустится левое поддерево

tnode\* left\_node = node->left;// Сохраняем левую часть дерева (от корня)

node = node->right;// Присваиваем корню правую часть дерева (от корня)

//нужно левую часть дерева вставить после наименьшего элемента в дереве

tnode\* tmp = node;// Временный узел для прохода по ветке

while (tmp->left != NULL) {

tmp = (tmp)->left;

height++;

}

changeLevel(node, -1);

changeLevel(left\_node, height);

tmp->left = left\_node;// <последний наименьший элмент в правом дереве>->left = бывшее левое дерево

}

return node;

}

else node;

}

// Поиск в бинарном дереве и удаление узла

tnode\* deleteNode(tnode\* node, int key, int level) {

tnode \*ptr;// Указатель для прохода по дереву

tnode\* ptr2;// Указатель для хранения поддерева, в котором удалена вершина

tnode\* prev\_ptr = node;// Указатель на предыдущий элемент (инициализация для компилятора, позже затрётся)

if (node != NULL) {// Если дерево имеет хотя бы 1 элемент

if (node->key == key && node->level==level) return deleteRootSubTree(node);// Если вершина дерева - то, что мы ищем,

else {// Иначе ищём этот элемент, двигаясь по узлам по единственному пути

ptr = node;

while (ptr != NULL)

if (ptr->level == level && ptr->key == key) {

ptr2 = deleteRootSubTree(ptr);

if (prev\_ptr->key > key) prev\_ptr->left = ptr2;

else prev\_ptr->right = ptr2;

return node;

}

else {

prev\_ptr = ptr;

if (ptr->key > key) ptr = ptr->left;

else ptr = ptr->right;

if (ptr == NULL) return node;// Если узел пуст

if (ptr->level > level) return node;// Если прошли нужный уровень

}

}

}

return node;

}

// Чтения количества элементов массива из файла

int readArrayN(const std::string& filename)// Чтения количества элементов массива из файла

{

ifstream in;

string str\_n;

int n = 0;

// Open the file.

in.open(filename, std::ios::in | std::ios::binary);

if (in.is\_open()) {

getline(in, str\_n);

}

else

{

throw exception("Exception: file did't open");

}

try {

n = stoi(str\_n);

}

catch (exception exc) {

throw exception("Exception: convert failure");

}

return n;

}

// Сохранения числа в файл

bool SaveNumberInFile(const std::string& filename, int key)

{

ofstream out;

// Open the file.

out.open(filename, ios::app | std::ios::binary);

// Validate that the file is open.

if (!out.is\_open())

return false;

out << key << endl;

out.close();

return true;

}

// Сохранить дерево в файл

void SaveTreeInFile(struct tnode\* node, const std::string& filename)

{

if (node != NULL) {

SaveTreeInFile(node->left, filename);

SaveNumberInFile(filename, node->key);// Сохранить 1 значение в файл

SaveTreeInFile(node->right, filename);

}

}

// Очистить файл

void clearFile(const std::string& filename) {

ofstream out;

// Open the file.

out.open(filename, ios::trunc);// Пересоздать файл (сделать пустым)

out.close();

}