**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Профиль «Автоматизированные системы обработки информации и управления»

**КУРСОВАЯ РАБОТА**

По дисциплине «Программирование»

На тему: «Калькулятор двадцати пятизначных чисел и бинарное дерево»

Выполнил:

студентка 2 курса,

группы АСУ 18-1БЗУ   
Моденова Надежда Владимировна

Научный руководитель:

Полякова О.А.

Пермь, 2020

**ОГЛАВЛЕНИЕ**

[**Постановка задачи** 3](#_Toc29589935)

[**Диаграмма классов** 4](#_Toc29589936)

[**Листинг программы** 13](#_Toc29589938)

[**Примеры экранных форм или диалогов** 36](#_Toc29589940)

[**Примеры результатов работы программы** 38](#_Toc29589941)

[**Программная документация** 40](#_Toc29589942)

[**Список литературы** 41](#_Toc29589943)

# **Постановка задачи**

1. Разработать класс «Калькулятор», выполняющий указанные в варианте операции для заданных исходных данных. Программа должна выполнять ввод данных, проверку правильности введенных данных, выдачу соответствующих сообщений в случае возникновения ошибок.

2)Протокол работы калькулятора записать в файл. Протокол должен включать исходные данные, введенные пользователем, выполняемые операции и результаты их выполнения. В случае возникновения ошибки в файл записывается соответствующее сообщение.

3)Предусмотреть возможность просмотра этого файла из программы калькулятора.

В варианте указаны вид данных, обрабатываемых калькулятором, и операции, выполняемые калькулятором.

**Вариант 1**: Квадратные уравнения (в действительных или комплексных числах). Вычисление корней в форме, заданной пользователем.

2. Разработать класс, реализующий указанную в задании структуру данных.

Разработать интерфейс программы, позволяющий вводить данные и получать результаты в удобной для пользователя форме. Программа должна проверять правильность вводимых данных. В случае возникновения ошибок должны выдаваться сообщения. Должна быть предусмотрена возможность получения данных из файла и запись результатов файл.

**Вариант 25:** Бинарное дерево - Автоматизированная информационная система на железнодорожном вокзале содержит сведения об отправлении поездов дальнего следования. Для каждого поезда указывается:

* номер поезда;
* станция назначения;
* время отправления.

Составить программу, которая

1. обеспечивает первоначальный ввод данных в информационную систему и формирование дерева;
2. производит вывод всего дерева;
3. вводит номер поезда и выводит данные об этом поезде;

вводит название станции назначения и выводит данные о всех поездах, которые следуют до этой станции.

# **Диаграмма классов**

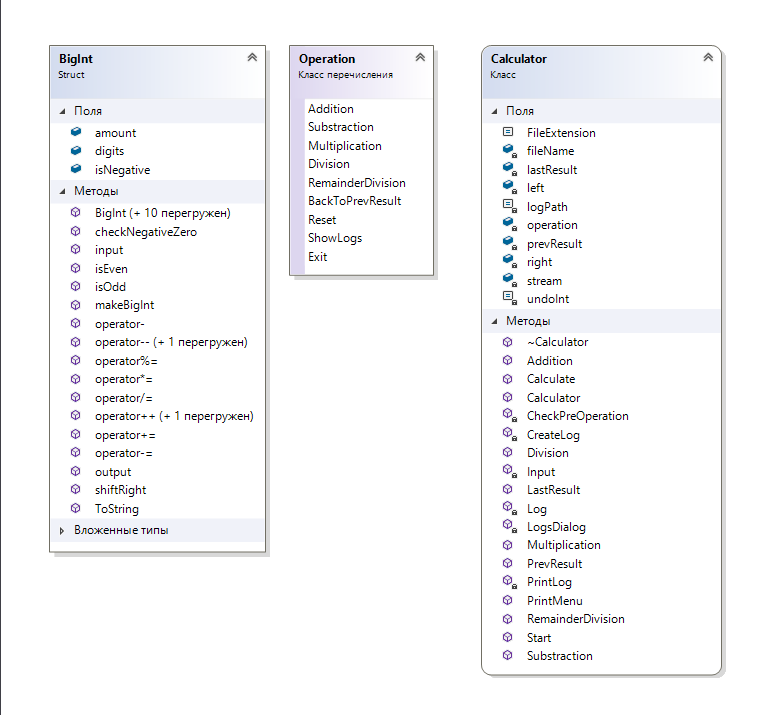


Рисунок 1 Диаграмма классов ч.1

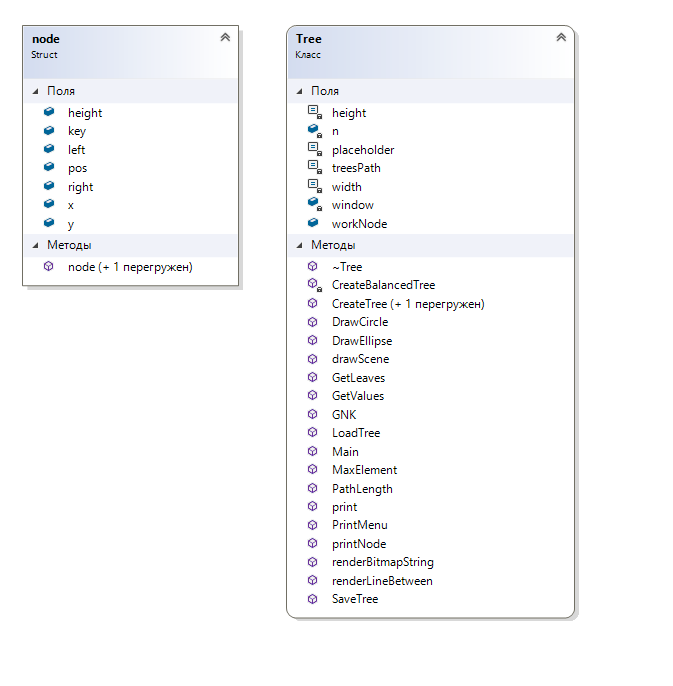


Рисунок 1 Диаграмма классов ч.2

# **Листинг программы**

1. **TrainBinTree**

#include "stdafx.h"

#include <GL/glut.h>

#include "Dialog.h" //инклюд диалога

#include "BinaryTree.h"

using namespace std;

BinaryTree::Node\* BinaryTree::workNode;

//точка входа в программу

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

{

glutInit(&argc, argv);

//вызываем конструктор без параметров

Dialog dialog;

//вызываем функцию execute

dialog.execute();

return 0;

}

#include "stdafx.h"

#include "BinaryTree.h" //заголовочный файл дерева

#include <iostream> //ввод вывод

#include <SDL.h>

#include <gl/glut.h>

#include <string>

//пространство имен, чтобы постояно его не указывать

using namespace std;

//Конструктор без параметров

BinaryTree::BinaryTree()

{

root = NULL;

}

//Деструктор

BinaryTree::~BinaryTree()

{

destroyTree();

}

//Функция вставляет новый поезд в цепочку

void BinaryTree::insert(Train key)

{

//Рекурсивно ищет конец дерева и вставляет туда элемент

if (root != NULL) //если элемент есть

insert(key, root, 'n'); //спускаемся

else //конец

{

root = new Node(); //создаем новый элемент

root->key = key; //указываем ему переданный поезд

root->left = NULL;

root->right = NULL;

root->pos = 'n';

}

}

//Функция поиска поезда

vector<BinaryTree::Node\*> BinaryTree::search(Train key)

{

Node \*res = root; //стартуем с рута

vector<Node\*> vec; //вектор для найденных поездов

while ((res = search(key, res)) != NULL) //рекурсивно ищем поезд

{

vec.push\_back(res); //если нашли, добавляем в наш вектор

res = res->right; //идем в право

}

return vec;

}

//Функция поиска поездов, идущих к указанной станции

vector<BinaryTree::Node\*> BinaryTree::searchByDestination(const string & key)

{

Node \*res = root; //стартуем с рута

vector<Node\*> vec; //вектор для найденных поездов

while ((res = searchByDestination(key, res)) != NULL) //рекурсивно ищем поезда

{

vec.push\_back(res); //найденный добавляем в вектор

res = res->right; //идем в право

}

return vec;

}

//Функция для деструктора

void BinaryTree::destroyTree()

{

destroyTree(root);

}

//Вывод листья

void BinaryTree::travesal(BinaryTree& tree)

{

workNode = tree.root;

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowPosition(100, 100);

glutInitWindowSize(640, 480);

glutCreateWindow("Tree");

glClearColor(1, 1, 1, 1);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, 640, 480, 0);

glutDisplayFunc(drawScene);

glutMainLoop();

//travesal(root);

}

void BinaryTree::drawScene() {

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

int x = 320;

int y = 30;

GenerateCoords(workNode, NULL, 320, 30, 0);

printNode(workNode);

glutSwapBuffers();

}

void BinaryTree::printNode(Node\*& p)

{

if (p != NULL)

{

DrawCircle(p->x, p->y, 10, 15);

renderBitmapString(p->x - 5, p->y + 5, 0, GLUT\_BITMAP\_HELVETICA\_18, to\_string(p->key.getID()));

if (p->right)

{

renderLineBetween(p->x, p->y + 10, p->right->x + 2, p->right->y - 9);

printNode(p->right);

}

if (p->left)

{

renderLineBetween(p->x, p->y + 10, p->left->x, p->left->y - 10);

printNode(p->left);

}

}

}

void BinaryTree::GenerateCoords(Node\* r, Node\* prev, int W, int Y, int L)

{

int buf;

buf = floor(W / pow(2, L));

if (r != NULL)

{

if (prev == NULL)

{

r->x = buf;

r->y = Y;

}

else

{

if (r->pos == 'l')

{

r->x = prev->x - buf + 10;

r->y = prev->y + 60;

}

else

{

if (r->pos == 'r')

{

r->x = prev->x + buf - 10;

r->y = prev->y + 60;

}

}

}

GenerateCoords(r->left, r, W, r->y, L + 1);

GenerateCoords(r->right, r, W, r->y, L + 1);

}

}

void BinaryTree::renderLineBetween(float x1, float y1, float x2, float y2)

{

glBegin(GL\_LINE\_STRIP);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

}

void BinaryTree::renderBitmapString(

float x,

float y,

float z,

void\* font,

string str) {

glRasterPos3f(x, y, z);

for (auto c = str.c\_str(); \*c != '\0'; c++) {

glutBitmapCharacter(font, \*c);

}

}

void BinaryTree::print(Node\* p)

{

workNode = p;

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowPosition(100, 100);

glutInitWindowSize(640, 480);

glutCreateWindow("Tree");

glClearColor(1, 1, 1, 1);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, 640, 480, 0);

glutDisplayFunc(drawScene);

glutMainLoop();

}

void BinaryTree::DrawCircle(float cx, float cy, float r, int num\_segments)

{

glBegin(GL\_LINE\_LOOP);

for (int ii = 0; ii < num\_segments; ii++)

{

float theta = 2.0 \* M\_PI \* float(ii) / float(num\_segments); //get the current angle

float x\_c = r \* cosf(theta);//calculate the x component

float y\_c = r \* sinf(theta);//calculate the y component

glColor3f(1.0f, 0.0f, 0.0f);

glVertex2f(x\_c + cx, y\_c + cy);//output vertex

}

glEnd();

}

void BinaryTree::DrawEllipse(float cx, float cy, float a, float b, int num\_segments)

{

glBegin(GL\_LINE\_LOOP);

for (int ii = 0; ii < num\_segments; ii++)

{

float theta = 2.0 \* M\_PI \* float(ii) / float(num\_segments); //get the current angle

float x\_e = a \* cosf(theta);//calculate the x component

float y\_e = b \* sinf(theta);//calculate the y component

glColor3f(0.0f, 0.0f, 1.0f);

glVertex2f(x\_e + cx, y\_e + cy);//output vertex

}

glEnd();

}

//Функция сохранения дерева в файл

void BinaryTree::saveToFile(const string & filename)

{

ofstream out(filename); //создаем поток на вывод

if (out.is\_open() && root != NULL) //если файл создался и есть рут

{

save(root, out); //вызываем вторую функцию сохранения

out.close(); //закрываем стрим (поток)

}

else

cout << "Cannot open file for save" << endl;

}

//Функция читает дерево из файла

void BinaryTree::openFile(const string & filename)

{

ifstream in(filename); //поток на чтение

if (in.is\_open()) //если удалось открыть файл

{

string idStr; //переменная для прочитанного ID в "сыром виде"

int id; //переменная для ID

string dest; //переменная для конечной точки

string departure; //переменная для даты

while (in >> idStr >> dest >> departure) //читаем файл до конца

{

sscanf(idStr.c\_str(), "%d", &id); //отдельно читаем ID и приводим его к INT

insert(Train(id, dest, departure)); //создаем и вставляем поезд с прочитанными данными

}

in.close(); //закрываем поток

}

else

cout << "Cannot open file" << endl;

}

//Функция уничтожает дерево

void BinaryTree::destroyTree(Node \* leaf)

{

if (leaf != NULL) //если лист есть

{

destroyTree(leaf->left); //рекурсивно уничтожаем левую часть

destroyTree(leaf->right); //рекурсивно уничтожаем правую часть

delete leaf; //уничтожаем из памяти лист

}

}

//Функция вставляет поезд

void BinaryTree::insert(Train key, Node\* leaf, char pos)

{

if (key < leaf->key) //если ID переданного поезда меньше листа

{

if (leaf->left != NULL) //если левый лист не NULL

insert(key, leaf->left, 'l'); //спускаемся

else //в противном случае создаем новый лист и указываем туда наш поезд

{

leaf->left = new Node();

leaf->left->key = key;

leaf->left->left = NULL;

leaf->left->right = NULL;

leaf->left->pos = 'l';

}

}

else if (key >= leaf->key) //если ID переданного поезда больше или листа

{

if (leaf->right != NULL) //если правый лист не NULL

insert(key, leaf->right, 'l'); //спускаемся

else //в противном случае создаем новый лист и указываем туда наш поезд

{

leaf->right = new Node();

leaf->right->key = key;

leaf->right->left = NULL;

leaf->right->right = NULL;

leaf->right->pos = 'r';

}

}

}

//Функция поиска листа

BinaryTree::Node \* BinaryTree::search(Train key, Node \* leaf)

{

if (leaf != NULL) //если лист не NULL

{

if (key == leaf->key) //проверяем значение

return leaf;

if (key < leaf->key) //если искомый поезд (его ID) меньше ID в текущем листа

return search(key, leaf->left); //рекурсивно спускаемся

else //в противном случае идем вправо

return search(key, leaf->right);

}

else

return NULL;

}

//Функция поиска по конечной станции

BinaryTree::Node \* BinaryTree::searchByDestination(const string & key, Node \* leaf)

{

if (leaf != NULL) //если лист не NULL

{

if (key == leaf->key.getDestination()) //сверяем направление поезда текущего листа

return leaf;

if (key < leaf->key.getDestination()) //если искомая станция (длина строки) меньше текущий

return searchByDestination(key, leaf->left); //спускаемся влево

else

return searchByDestination(key, leaf->right); //иначе вправо

}

else

return NULL;

}

//Функция последнего элемента дерева

void BinaryTree::travesal(Node \* root)

{

if (root != NULL) { //если корневой элемент есть

travesal(root->left); //рекурсивно спускаеся до конца (левая часть)

cout << root->key << endl; //выводим

travesal(root->right);//рекурсивно спускаеся до конца (правая часть)

}

}

//Функция сохранения дерева

void BinaryTree::save(Node \* root, ofstream& out)

{

if (root == NULL) //если корня нет, выходим

return;

out << root->key.getID() << endl; //сохраняем ID

out << root->key.getDestination() << endl; //сохраняем станцию

out << root->key.getDepartureTime() << endl; //сохраняем дату

save(root->left, out); //рекурсивно продолжаем в левую часть

save(root->right, out); //рекурсивно продолжаем в правую часть

}

#include "stdafx.h" //стандартный инклюд VS 2017

#include "Dialog.h" //заголовочный файл диалога

#include <iostream> //ввод вывод

#include "BinaryTree.h" //заголовочный файл дерева

//пространство имен std, чтобы постояно его не писать

using namespace std;

//Конструктор без параметров

Dialog::Dialog()

{

}

//Деструктор

Dialog::~Dialog()

{

}

//Главная функция работы с деревом

void Dialog::execute()

{

//текущее дерево

BinaryTree tree;

//переменная для команды

int command = -1;

//по факту бесконечный цикл, пока пользователь не введет команду на выход

while (command != 0)

{

//выводим меню

printMenu();

//читаем команду

command = inputCommand();

//после ввода команды очищаем консоль

system("cls");

//свитч по команде

switch (command)

{

case 1: //если команда 1

insertTrain(tree); //вставить поезд

break;

case 2: //если команда 2

printAllTrain(tree); //показать все поезда

break;

case 3: //если команда 3

printTrainInfoById(tree); //вывод информации о поезде по ID

break;

case 4: //если команда 4

printTrainGoToDestination(tree); //вывод поезда, идущего до введеной станции

break;

case 5: //если команда 5

save(tree); //сохранить текущее дерево

break;

case 6: //если команда 6

open(tree); //загрузить дерево из файла

break;

default: //если команда не найдена

break;

}

cin.get();

}

}

//Выводим меню помощи

void Dialog::printMenu()

{

system("cls"); //очищаем консоль

cout << "1. Insert new train data." << endl;

cout << "2. Print all train." << endl;

cout << "3. Print info about train by id." << endl;

cout << "4. All train go to destination." << endl;

cout << "5. Save." << endl;

cout << "6. Open." << endl;

cout << "0. Exit." << endl;

}

//Функция для получения дальнейшей команды, для работы с меню

int Dialog::inputCommand()

{

cout << "\nChoose command: ";

int command; //переменная для команды

cin >> command; //читаем команду

cin.get();

while (command < 0 || command > 6) //цикл, для того, чтобы пользователь ввёл какую-либо команду. Завершится, когда пользователь введет существующий номер команды

{

cout << "Wrong command. Try again" << endl;

cout << "\nChoose command: ";

cin >> command;

}

return command; //возвращаем ID команды (номер)

}

//вставляет новый поезд

void Dialog::insertTrain(BinaryTree & tree)

{

//Train train; //конструктор без параметров, наш поезд

int id; //переменная для ID

string dest; //название конечной станции

string departure; //дата прибытия

cout << "Input id: ";

cin >> id; //записываем ID поезда

cin.get();

cout << "Input destination station: ";

getline(cin, dest); //записываем конечную станцию

cout << "Input departure time (hh:mm) : ";

getline(cin, departure); //записываем время прибытия

tree.insert(Train(id, dest, departure)); //вставляет в дерево наш поезд, вызывая конструктор, в который передаем ID поезда, конечную станцию и время прибытия

}

//Выводит все поезда

void Dialog::printAllTrain(BinaryTree & tree)

{

tree.travesal(tree); //вызываем функцию travesal у дерева

}

//Выводим информацию о позде по его ID

void Dialog::printTrainInfoById(BinaryTree & tree)

{

int id; //переменная для ID

cout << "Input id for search: ";

cin >> id; //записываем ID

cin.get();

Train train(id, "", ""); //создаем пустой поезд, вызывая конструктор, куда передаем прочитанный ID, и пустые строки для конечной станции и времени прибытия

vector<BinaryTree::Node\*> res = tree.search(train); //вызываем функцию поиска у дерева, где в качестве ключа используем только что созданный поезд

if (res.size() > 0) //если что-то есть

{

for (int i = 0; i < res.size(); ++i) //выводим в цикле информацию о всех найденных поездах

cout << res[i]->key << endl;

}

else //ничего не нашли

cout << "Train with id = " << id << " doesnt find" << endl;

}

//Выводит список поездов, следующих до определенной станции

void Dialog::printTrainGoToDestination(BinaryTree & tree)

{

string dest; //переменная типа строка для направления (станции)

cout << "Input destination for search: ";

getline(cin, dest); //записываем введенноё направление с клавиатуры

vector<BinaryTree::Node\*> res = tree.searchByDestination(dest); //вызываем функцию поиска по станции

if (res.size() > 0) //если что-то нашли

{

for (int i = 0; i < res.size(); ++i) //выводим в цикле информацию о поездах

cout << res[i]->key << endl;

}

else //ничего не нашли

cout << "Train go to station = " << dest << " doesnt find" << endl;

}

//Сохраняет дерево в файл

void Dialog::save(BinaryTree & tree)

{

string filename; //переменная для имени файла

cout << "Input filename: ";

getline(cin, filename); //записываем имя файла в переменную

tree.saveToFile(filename); //вызываем функция сохранения в файл у дерева, передавая имя файла

}

//Открывает (читает) дерево из файла

void Dialog::open(BinaryTree & tree)

{

string filename; //переменная для имени файла

cout << "Input filename: ";

getline(cin, filename); //записываем имя файла в переменну.

tree.openFile(filename); //вызываю функцию чтения дерева из файла, передавая туда имя файла

}

#include "stdafx.h"

#include "Train.h" //заголовочный файл поезда

/Конструктор без параметров

Train::Train()

{

id = 0;

destinationStation = "";

departureTime = "";

}

//Конструктор с параметрами, принимает id поезда (без знаковый), конечную точку и время прибытия

Train::Train(unsigned int id, string destination, string departureTime)

{

//имя входного параметра совпадает с именем переменной, явно указывает через this, что приравнием в переменную = входной параметр

this->id = id;

destinationStation = destination;

//имя входного параметра совпадает с именем переменной, явно указывает через this, что приравнием в переменную = входной параметр

this->departureTime = departureTime;

}

//Конструктор копирования

Train::Train(const Train & train)

{

id = train.id;

destinationStation = train.destinationStation;

departureTime = train.departureTime;

}

//Перегрузка оператора приравнивания

Train & Train::operator=(const Train & train)

{

if (this != &train)

{

id = train.id;

destinationStation = train.destinationStation;

departureTime = train.departureTime;

}

return \*this;

}

//Деструктор

Train::~Train()

{

}

//Перегрузка оператора <

bool Train::operator<(const Train & train)

{

return id < train.id;

}

//Перегрузка оператора == (сравнение)

bool Train::operator==(const Train & train)

{

return id == train.id;

}

//Перегрузка оператора (потокового) вывода

std::ostream & operator<<(std::ostream & os, const Train & train)

{

os << "Id: " << train.id << ", Destination: " << train.destinationStation << ", Time: " << train.departureTime;

return os;

}

//Перегрузка оператора >

bool Train::operator>(const Train & train)

{

return id > train.id;

}

//Перегрузка оператора >=

bool Train::operator>=(const Train & train)

{

return id >= train.id;

}

//Перегрузка оператора <=

bool Train::operator<=(const Train & train)

{

return id <= train.id;

}

1. **Calculator**

#include "stdafx.h" //либа visual studio

#include "Calculator.h" //заголовочный файл калькулятора, с описанием функций

#include <iostream> //библиотека ввода/вывода

//Подключаем пространство имен STD, чтобы постояно его не писать

using namespace std;

//Конструктор без парамертов

Calculator::Calculator()

{

}

//Конструктор, принимающий 3 параметра: a,b,c типа float (плавающая точка, дробь)

Calculator::Calculator(float A, float B, float C)

{

this->A = A;

this->B = B;

this->C = C;

}

//Деструктор без параметров

Calculator::~Calculator()

{

}

void Calculator::setCoeff(float A, float B, float C)

{

this->A = A;

this->B = B;

this->C = C;

}

//Функция, считающая выражение

void Calculator::eval()

{

//float переменные, нужны для расчетов

float x1, x2, discriminant, realPart, imaginaryPart;

//Считаем дискриминант

discriminant = B \* B - 4 \* A \* C;

//Если дискриминант большн 0

if (discriminant > 0) {

//считаем x1

x1 = (-B + sqrt(discriminant)) / (2 \* A);

//считаем x2

x2 = (-B - sqrt(discriminant)) / (2 \* A);

//Выводим результат

cout << "\nRoots are real and different." << endl;

cout << "x1 = " << x1 << endl;

cout << "x2 = " << x2 << endl;

}

//если дискриминант равен 0

else if (discriminant == 0) {

cout << "\nRoots are real and same." << endl;

//считаем только x1

x1 = (-B + sqrt(discriminant)) / (2 \* A);

//Результат

cout << "x1 = x2 =" << x1 << endl;

}

//если меньше 0

else {

//считаем первую часть комплексного числа

realPart = -B / (2 \* A);

imaginaryPart = sqrt(-discriminant) / (2 \* A);

//выводим результат

cout << "\nRoots are complex and different." << endl;

cout << "x1 = " << realPart << "+" << imaginaryPart << "i" << endl;

cout << "x2 = " << realPart << "-" << imaginaryPart << "i" << endl;

}

}

//Выводим шаблон выражения с текущими переменные, установленными в конструкторе или через setKoeff

void Calculator::printEquationWithCoef()

{

cout << A << " \* x^2 + " << B << " \* x + " << C << " = 0" << endl;

}

//Выводи шаблон выражения

void Calculator::printBaseEquation()

{

cout << "A \* x^2 + B \* x + C = 0" << endl;

}

// QuadEqSolver.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <iostream>

#include "Calculator.h"

using namespace std;

//Точка входа в программу

int main()

{

//Создаем новый экземпляр калькулятора

Calculator calculator;

//Выводим подсказу - шаблон изображения

calculator.printBaseEquation();

//Создаеи 3 переменные типа float (плавающая точка, дроби)

float a, b, c;

//Выводим информацию о том, что нужно ввести 3 коэффициента

cout << "Enter coefficients a, b and c: " << endl;

cout << "a = ";

cin >> a; //читаем первую цифру и пишем в 'a'

cout << "b = ";

cin >> b; //читаем вторую цифру и пишем в 'b'

cout << "c = ";

cin >> c; //читаем третью цифру и пишем в 'c'

cout << endl;

//Устанавливаем числа в калькуляторе

calculator.setCoeff(a, b, c);

//Выводим построенное изображение с подстановкой введенных данных

calculator.printEquationWithCoef();

//Считаем

calculator.eval();

cout << "Enter to exit...";

system("pause");

return 0;

}

# }**Примеры экранных форм или диалогов**

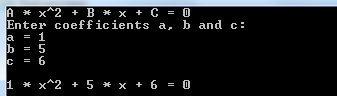


Рисунок 3 – Меню калькулятора

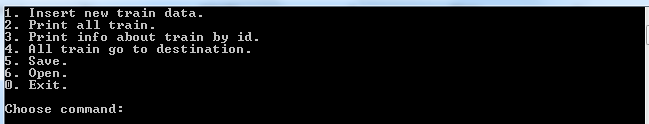


Рисунок 4 – Меню управления деревьями

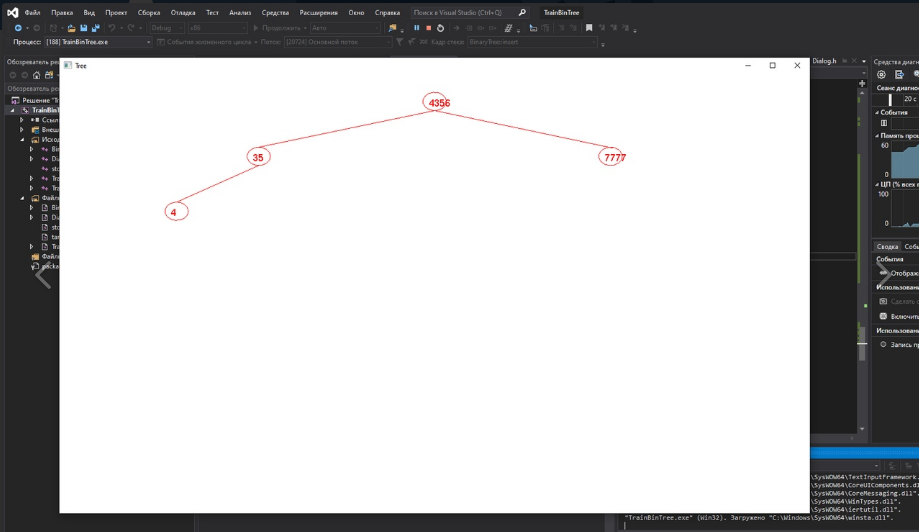


Рисунок 5 – Визуализация дерева

# **Примеры результатов работы программы**

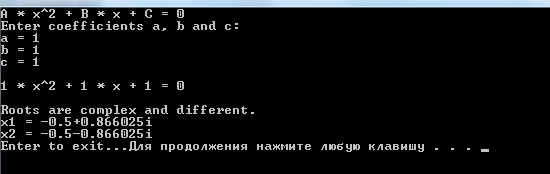


Рисунок 6 – Результат работы программы (калькулятор)

# **Программная документация**

1. **Состав проекта:**

- QuadEqSolver.sln –проект QuadEqSolver;

-TrainBinTree.sln- проект TrainBinTree;

- stdafx.h – заголовочный файл;

- Calculator.h – заголовочный файл с реализацией калькулятора;

- Train.h – заголовочный файл с реализацией дерева;

Для запуска проекта потребуется Visual studio 2019. Используется 17 стандарт C++. Дополнительных параметров при компиляции нет.

1. **Работа с программой:**

Запустите QuadEqSolver.exe из QuadEqSolver\Debug, TrainBinTree.exe из TrainBinTree\Debug. Для ввода команд и значений используйте клавиатуру.

# **Список литературы**

1. Вирт Н. Алгоритмы + структуры данных = программы . – М. : Мир, 1985. – 406 с.
2. Голуб А. И. С и С++. Правила программирования. – М.: БИНОМ, 1996. – 272 с.
3. Ноткин А. М. Динамические структуры данных. Учебное по-собие. - Пермь, 1994, -66с.
4. Павловская Т. А. С/С++. Программирование на языке высо-кого уровня. – СПб.: Питер, 2001. – 464с.
5. Павловская Т. А., Щупак Ю. А. С++. Объектно-ориентированное программирование: Практикум. – СПб.:Питер, 2004. – 265 с.
6. Подбельский В. В. Язык Си++ : Учеб. пособие.– М.: Фи-нансы и статистика, 1996.–560с.
7. Подбельский В. В., Фомин С. С. Программирование на язы-ке Си: Учеб. пособие. – М.:Финансы и статистика, 1998.–600с.
8. Страуструп Б. Язык программирования С++. –СПб.:БИНОМ, 1999. – 991 с.
9. Страуструп Б. Язык программирования Си++: Пер. с англ. – М.: Радио и связь, 1991.-352с