Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Пермский национальный исследовательский политехнический университет» (ПНИПУ)

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

**Творческая работа**

**по дисциплине**

**«Основы алгоритмизации и программирования»**

**семестр** 2

Выполнил студент гр. ИВТ-21-1б

Бурылов Максим Павлович

Проверил:

Яруллин Денис Владимирович

\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

(оценка) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

г. Пермь – 2022

**Калькулятор квадратных уравнений**

**Постановка задачи:**

1. Реализовать калькулятор Обыкновенных и десятичных дробей.
2. Создать интерфейс работы с калькулятором.

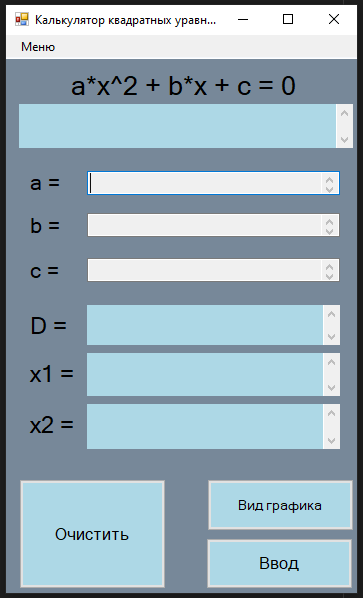
**Описание Решения**

Для решения задачи были использованы стандартные способы решения квадратных уравнений.

Интерфейс создан при помощи WindowsForms, вывод рисунков - SFML/Graphics.hp.

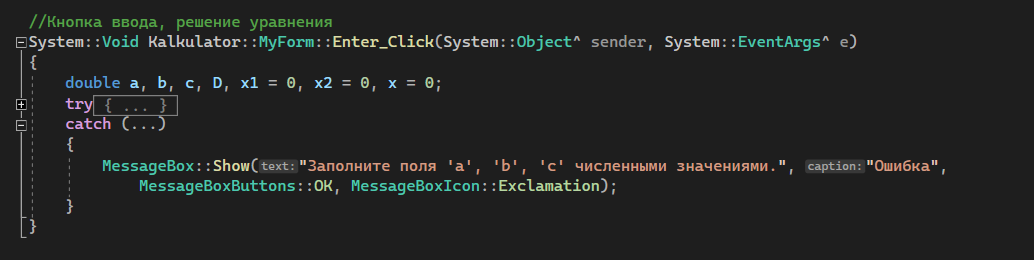
Трудность, связанная с отрисовкой графиков функций была решена использованием готовых графиков функций.

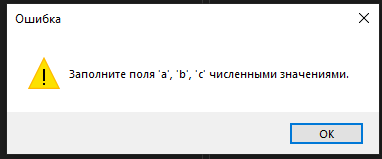
Автор гордится простотой использования калькулятора, наличием встроенных теоретической справки и 6-ти основных видов графиков функций.

Калькулятор выполнен в цветах не нагружающих зрение и обладает доступным дизайном.

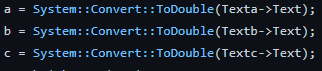
**Описание кода**

Проверка ввода и вывод окна с сообщением в случае ошибки



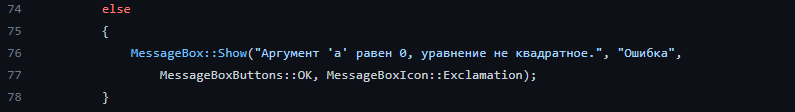


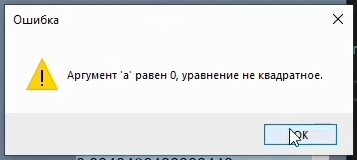
Считывание значений из переменных TextBox с переводом в численные значения

****

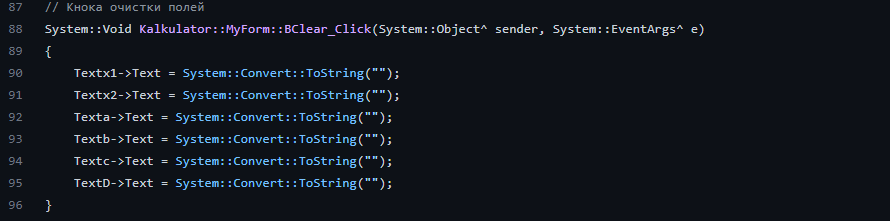
Основная логика решения квадратных уравнений с выводом соответствующих ответов

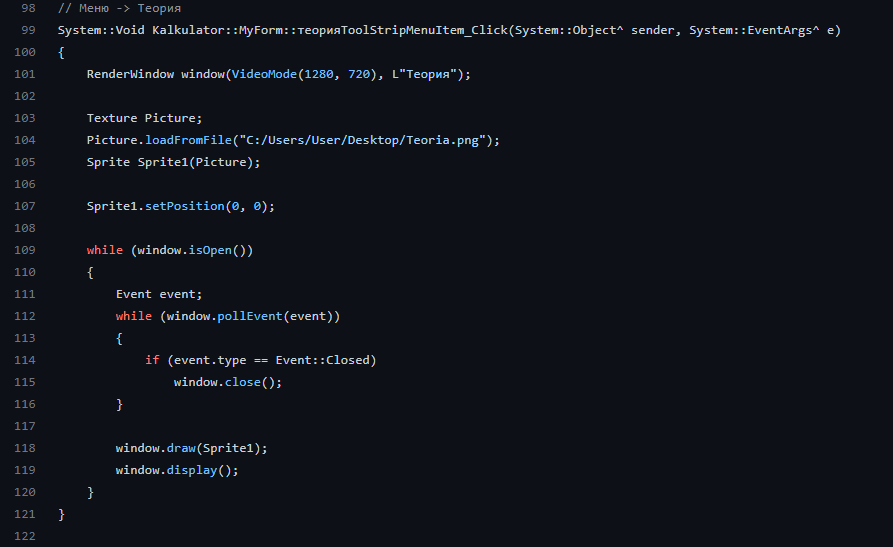
Проверка на тип уравнения

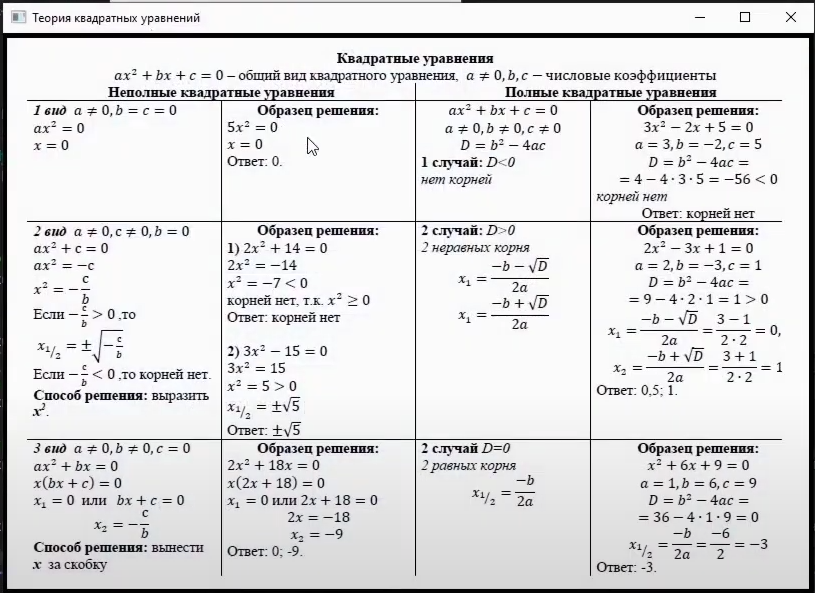




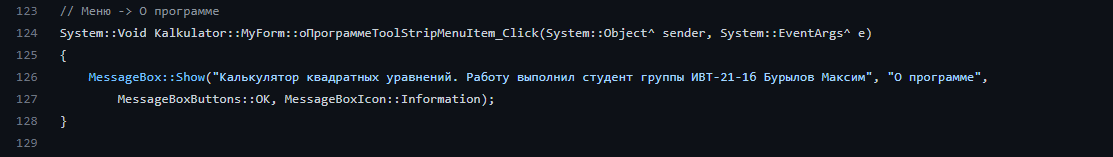
Код кнопки очистки полей

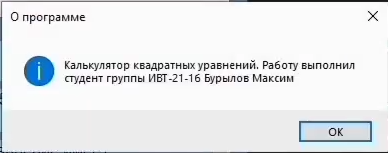


Создание окна при помощи ----- и вывод в него теоретической справки при нажатии кнопки «Теория» 

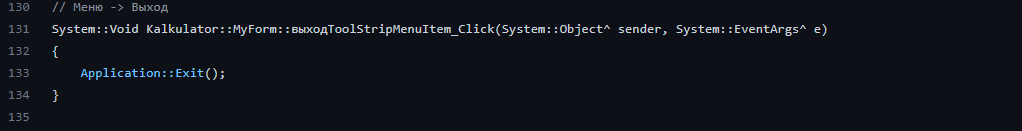


Вывод окна с краткой информацией о программе

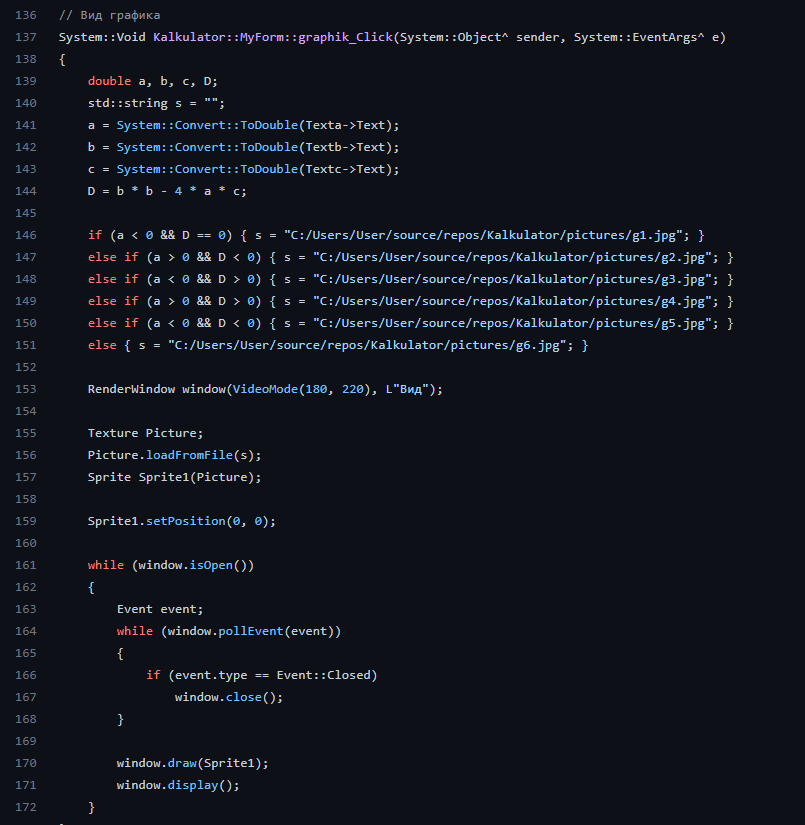


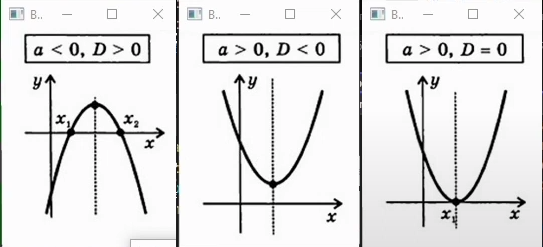
****

Функционал выхода из программы



Определение вида графика, создание и вывод окна с данным графиком

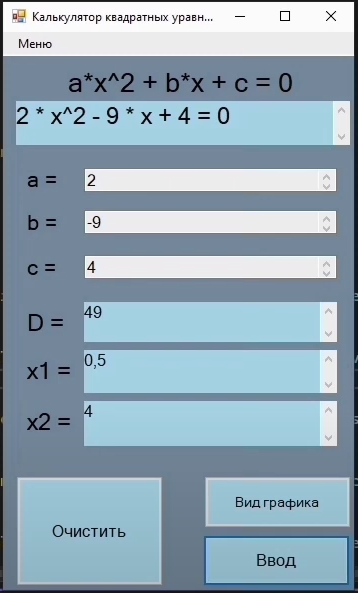


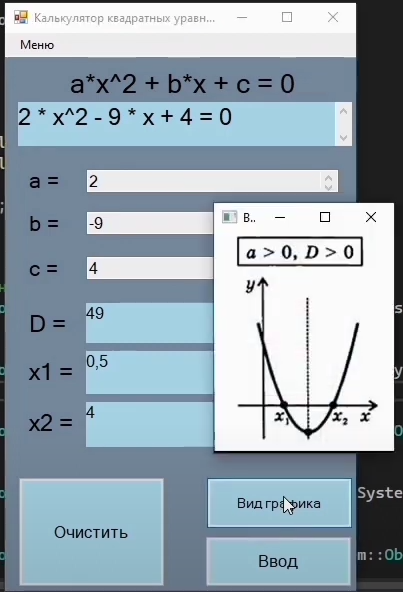


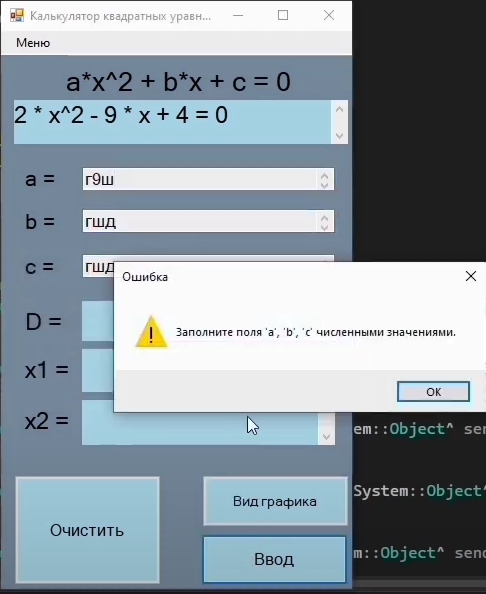
**Диаграмма классов**

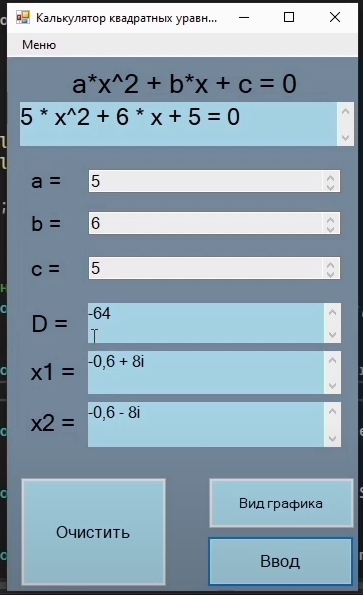


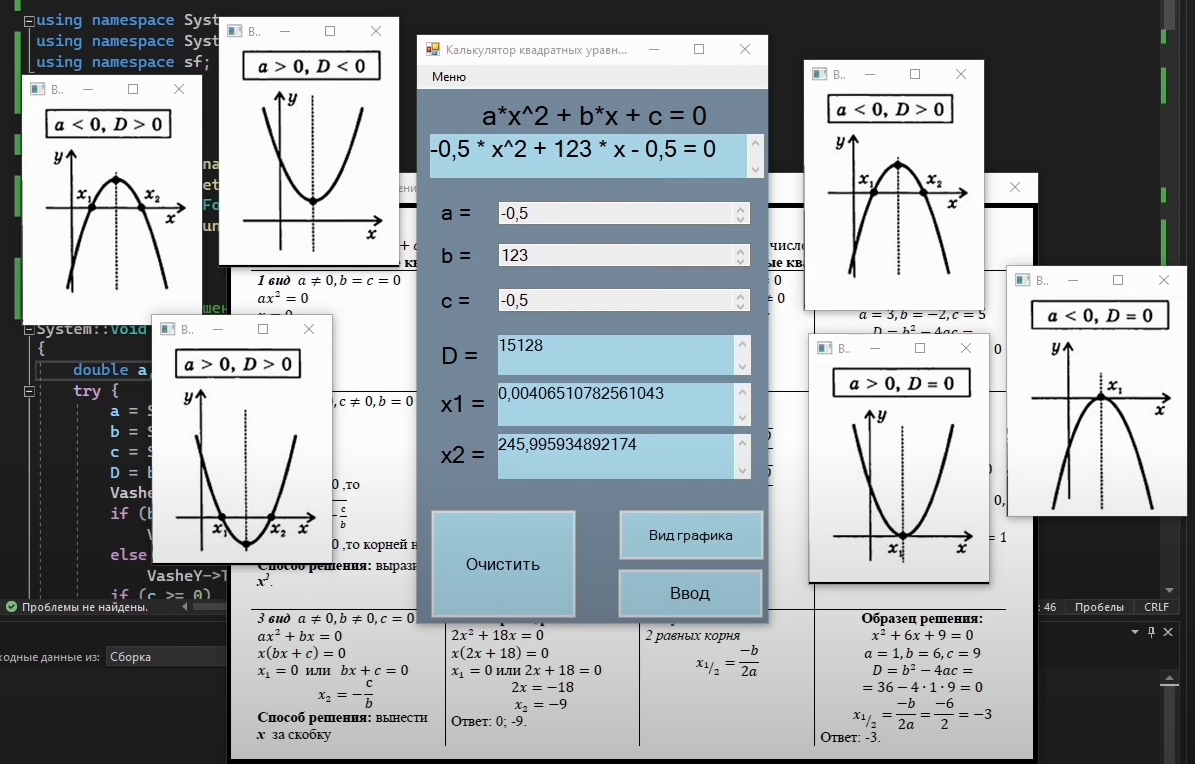
**Скриншоты работы**

****

****

****

****

****

**Задача коммивояжера**

**Постановка задачи:**

1. Реализовать решение задачи Коммивояжера методом ветвей и границ.
2. Вывести граф.
3. Добавить возможность удаления и добавления вершин

**Описание решения:**

Для визуализации использовался инструмент OpenGL.

Трудности:

Сложность визуализации графа, но примеры готовых решений помогли исправить ситуацию.

Описание кода:

#pragma once

#include <GL/glut.h>

#include <stdio.h>

#include <iostream>

#include <vector>

#include <sstream>

using namespace std;

int Win\_W;

int Win\_H;

int n;

int\*\* dynamic\_mas;

int\* result;

int\*\*\* ukazatel\_na\_matrix;

const int maxSize = 15;

int kolvo\_vershin;

template<class T>

class Graph

{

vector<T> vertList;

vector<T> labelList;

bool\* visitedVerts = new bool[vertList.size()];

public:

int adjMatrix[maxSize][maxSize];

Graph();

~Graph();

void RenderingGraph();

void Way\_km(const T& vertex1, const T& vertex2, int weight);

void Add\_Vertex(const T& vertex);

void Delete\_this\_Vertex();

int GetVertPos(const T& vertex);

bool IsEmpty();

bool IsFull();

int Getter\_kolvo\_veshin();

int Getter\_kolvo\_border();

int Getter\_length(const T& vertex1, const T& vertex2);

vector<T> Getter\_n(const T& vertex);

void Print();

};

const int Predel = 100000; // заглушка для прохода в цикл

int parametr; //

struct vertCoord

{

int x, y;

};

vertCoord vertC[20];

Graph<int> graph;

void creating\_variables(int\*\*\*& matrix, int& n, int\*\*& dynamic\_mas, int\*& result)

{

n = kolvo\_vershin;

dynamic\_mas = new int\* [n];

result = new int[n];

matrix = new int\*\* [n];

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

{

dynamic\_mas[i] = new int[n];

}

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

{

matrix[i] = new int\* [n];

for (int j = 0; j < n; j++)

{

if (graph.adjMatrix[i][j] == 0)

{

matrix[i][j] = nullptr;

continue;

}

matrix[i][j] = new int(graph.adjMatrix[i][j]);

}

}

}

void task\_solution(int\*\*\* matrix, int n, int\*\* dynamic\_mas, int\* path)

{

int fist\_min;

int second\_min;

for (int l = 0; l < n; l++) // Редукция строк

{

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

{

int min = Predel;

for (int j = 0; j < n; j++)

if (matrix[i][j] && min > \*matrix[i][j])

min = \*matrix[i][j];

for (int j = 0; j < n; j++)

if (matrix[i][j])

\*matrix[i][j] -= min;

}

for (int j = 0; j < n; j++) // Редукция столбцов

{

int min = Predel;

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

if (matrix[i][j] && min > \*matrix[i][j])

min = \*matrix[i][j];

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

if (matrix[i][j])

\*matrix[i][j] -= min;

}

for (int i = 0; i < n; i++) // Происходит оценка нулевых элементов

for (int j = 0; j < n; j++)

dynamic\_mas[i][j] = 0;

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

for (int j = 0; j < n; j++)

{

if (matrix[i][j] && !\*matrix[i][j])

{

fist\_min = Predel;

second\_min = Predel;

for (int l = 0; l < n; l++)

if (l != i && matrix[l][j] && fist\_min > \*matrix[l][j])

fist\_min = \*matrix[l][j];

for (int l = 0; l < n; l++)

if (l != j && matrix[i][l] && second\_min > \*matrix[i][l])

second\_min = \*matrix[i][l];

dynamic\_mas[i][j] = fist\_min + second\_min;

}

}

int waste = 0, line\_i = 0, column\_j = 0;

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

for (int j = 0; j < n; j++)

if (matrix[i][j] && waste < dynamic\_mas[i][j])

{

waste = dynamic\_mas[i][j];

line\_i = i;

column\_j = j;

}

path[line\_i] = column\_j;

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

matrix[i][column\_j] = nullptr;

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

matrix[line\_i][i] = nullptr;

matrix[column\_j][line\_i] = nullptr;

}

}

void traveler\_man(int\*\*\* matrix, int n, int\*\* dynamic\_mas, int\* result)

{

creating\_variables(matrix, n, dynamic\_mas, result);

task\_solution(matrix, n, dynamic\_mas, result);

cout << "\nЗаданные пути: ";

int s = 0;

int j;

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

{

j = result[i];

cout << "( " << i + 1 << " --> " << j + 1 << " )" << " "; // вывод путей между пунктами

s += graph.adjMatrix[i][j];

}

cout << endl << endl;

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

{

cout << "-------------";

}

cout << endl << "Лучший путь: ";

int temp = 0; // переменная для сравнения

for (int l = 0; l < n;)

{

for (int i = 0, j = 0; i < n; i++)

{

if (temp == 0 || i + 1 == temp)

{

if (temp == 0) cout << i + 1;

j = result[i];

temp = j + 1;

if (temp > 0)

{

cout << " --> " << temp;

l++;

}

}

}

}

cout << endl << "\nПуть равен: " << s << endl;

}

template<class T>

vector<T> Graph<T>::Getter\_n(const T& vertex)

{

vector<T> nbrsList;

int vertPos = this->GetVertPos(vertex);

if (vertPos != (-1)) // если позиция вершины не равна -1,

{

for (int i = 0, vertListSize = this->vertList.size(); i < vertListSize; ++i)

{

if (this->adjMatrix[vertPos][i] != 0 && this->adjMatrix[i][vertPos] != 0)

nbrsList.push\_back(this->vertList[i]);

}

}

return nbrsList;

}

template<class T>

void Graph<T>::Add\_Vertex(const T& vertex) // добавление вершины

{

if (!this->IsFull())

{

this->vertList.push\_back(vertex);

}

else

{

cout << "Граф заполнен. Нужно добавить новую вершину " << endl;

return;

}

}

template<class T>

void Graph<T>::Delete\_this\_Vertex() // удаление вершины

{

this->vertList.pop\_back(); // удаляет вершины в стиле стека, те начиная с вершины

}

template<class T>

int Graph<T>::Getter\_kolvo\_border() // количество границ

{

int amount = 0;

if (!this->IsEmpty())

{

for (int i = 0, vertListSize = this->vertList.size(); i < vertListSize; ++i)

{

for (int j = 0; j < vertListSize; ++j)

{

if (this->adjMatrix[i][j] == this->adjMatrix[j][i] && this->adjMatrix[i][j] != 0)

amount += 1;

}

}

return (amount / 2);

}

else

return 0;

}

template<class T>

int Graph<T>::Getter\_length(const T& vertex1, const T& vertex2) // получение длины пути

{

if (!this->IsEmpty())

{

int vertPos1 = GetVertPos(vertex1);

int vertPos2 = GetVertPos(vertex2);

return adjMatrix[vertPos1][vertPos2];

}

return 0;

}

template<class T>

int Graph<T>::Getter\_kolvo\_veshin() // получение количества вершин

{

return this->vertList.size(); // возвращает размер

}

template<class T>

bool Graph<T>::IsEmpty() // пустота

{

if (this->vertList.size() != 0)

return false;

else

return true;

}

template<class T>

bool Graph<T>::IsFull() // Места нет

{

return (vertList.size() == maxSize);

}

template <class T>

int Graph<T>::GetVertPos(const T& vertex) // получение позиции вершины

{

for (int i = 0; i < this->vertList.size(); ++i)

{

if (this->vertList[i] == vertex)

return i;

}

return -1;

}

template<class T>

Graph<T>::Graph()

{

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

{

for (int j = 0; j < maxSize; ++j)

{

this->adjMatrix[i][j] = 0;

}

}

}

template<class T>

Graph<T>::~Graph()

{

}

Graph<int> graph\_implementation() // реализация графа, его логики работы...

{

Graph<int> graph; // объект класса Graph

int amountEdges, sourceVertex, targetVertex, edgeWeight; // создаем переменные

cout << endl << "Укажите, в скольких городах вы хотите побывать: ";

cin >> kolvo\_vershin; // вводим количество вершин

cout << endl;

while (kolvo\_vershin < 0) // проверка ввода

{

cout << "Количество городов не может быть отрицательным числом!" << endl << endl;

cout << "Попробуйте еще раз: ";

cin >> kolvo\_vershin;

cout << endl;

}

cout << endl;

cout << "Введите количество дорог между городами: ";

cin >> amountEdges;

cout << endl;

while (amountEdges < 0)

{

cout << endl << "Количество дорог не может быть отрицательным числом!" << endl << endl;

cout << "Попробуйте еще раз: ";

cin >> amountEdges;

cout << endl;

}

cout << endl;

for (int i = 1; i <= kolvo\_vershin; ++i) // проход до количества вершин(включительно)

{

int\* vertPtr = &i;

graph.Add\_Vertex(\*vertPtr);

}

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

{

cout << "----------------------" << endl;

cout << "Введите " << i + 1 << " пару вершин: " << endl;

cout << "----------------------" << endl << endl;

cout << "Идём из вершины: ";

cin >> sourceVertex;

cout << endl;

int\* sourceVertPtr = &sourceVertex;

cout << "В вершину: ";

cin >> targetVertex;

cout << endl;

int\* targetVertPtr = &targetVertex;

cout << "Введите длину пути между данными вершинами(пунктами): ";

cin >> edgeWeight;

while (edgeWeight < 0) { cout << endl << "Длина пути не может быть отрицательной велечиной. Введите еще раз: "; cin >> edgeWeight; cout << endl; }

cout << endl;

graph.Way\_km(\*sourceVertPtr, \*targetVertPtr, edgeWeight);

}

cout << endl;

return graph;

}

template<class T>

void Graph<T>::Way\_km(const T& vertex1, const T& vertex2, int weight) // работа с длиной пути

{

if (this->GetVertPos(vertex1) != (-1) && this->GetVertPos(vertex2) != (-1))

{

int vertPos1 = GetVertPos(vertex1); // присваиваем позицию

int vertPos2 = GetVertPos(vertex2);

if (this->adjMatrix[vertPos1][vertPos2] != 0 && this->adjMatrix[vertPos2][vertPos1] != 0)

{

cout << endl << "Переопределяем длину данного пути " << endl << endl;

if (weight < 0) { cout << endl << "Длина добавляемого пути не может быть отрицательной велечиной" << endl; return; }

this->adjMatrix[vertPos1][vertPos2] = weight;

this->adjMatrix[vertPos2][vertPos1] = weight;

return;

}

else

{

if (weight < 0) { cout << endl << "Длина добавляемого пути не может быть отрицательной велечиной" << endl; return; }

this->adjMatrix[vertPos1][vertPos2] = weight;

this->adjMatrix[vertPos2][vertPos1] = weight;

}

}

else

{

cout << endl << "Таких пунктов нет" << endl;

cout << "Подрузамевается, что вы хотели добавить пункт, поэтому даже в случае ошибочного ввода, в граф добавляется пункт " << endl << endl;

return;

}

}

template<class T>

void Graph<T>::Print() // вывод матрицы смежности

{

if (!this->IsEmpty()) // если не пусто

{

cout << endl << "Матрица смежности: " << endl;

for (int i = 0, vertListSize = this->vertList.size(); i < vertListSize; ++i)

{

cout << this->vertList[i] << " ";

for (int j = 0; j < vertListSize; ++j)

{

cout << " " << this->adjMatrix[i][j] << " ";

}

cout << endl;

}

}

}

void setCoord(int i, int n) // устанавливаем координаты

{

int R\_; // в зависимости от случая принимает ширину окна или длину окна

int x0 = Win\_W / 2;

int y0 = Win\_H / 2;

if (Win\_W > Win\_H)

{

parametr = 5 \* (Win\_H / 13) / n;

R\_ = Win\_H / 2 - parametr - 10;

}

else {

parametr = 5 \* (Win\_W / 13) / n;

R\_ = Win\_W / 2 - parametr - 10;

}

float theta = 2.0f \* 3.1415926f \* float(i) / float(n);

float y1 = R\_ \* cos(theta) + y0; // cos угла для вычисления положения линии(путя)

float x1 = R\_ \* sin(theta) + x0; // sin угла для вычисления положения линии(путя)

vertC[i].x = x1;

vertC[i].y = y1;

}

void drawCircle(int x, int y, int R) // отрисовка круга

{

glColor3f(0.3, 0.9, 0.7); // установка цвета

float x1, y1;

glBegin(GL\_POLYGON);

for (int i = 0; i < 360; i++)

{

float theta = 2.0f \* 3.141592653589793f \* float(i) / float(360); // формула точности круга

y1 = R \* cos(theta) + y;

x1 = R \* sin(theta) + x;;

glVertex2f(x1, y1);

}

glEnd();

glColor3f(0.0f, 0.0f, 0.0f); // установка цвета

float x2, y2;

glLineWidth(2); // установка ширины прямой

glBegin(GL\_LINE\_LOOP); // отличие от предыдущего примитива только в том, что последний отрезок определяется последней и первой вершиной, образуя замкнутую ломаную

for (int i = 0; i < 360; i++)

{

float theta = 2.0f \* 3.1415926f \* float(i) / float(360);

y2 = R \* cos(theta) + y;

x2 = R \* sin(theta) + x;

glVertex2f(x2, y2);

}

glEnd();

}

void drawText(int nom, int x1, int y1) // отрисовка текста

{

GLvoid\* font = GLUT\_BITMAP\_HELVETICA\_18; // шрифт по умолчанию есть в Opengl

string s = to\_string(nom);

glRasterPos2i(x1 - 5, y1 - 5);

for (int j = 0; j < s.length(); j++)

glutBitmapCharacter(font, s[j]);

}

void drawVertex(int n) // отрисовка вершины

{

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

{

drawCircle(vertC[i].x, vertC[i].y, parametr);

drawText(i + 1, vertC[i].x, vertC[i].y);

}

}

void draw\_Line\_for\_way(int text, int x0, int y0, int x1, int y1) // отрисовка линий для путей

{

glColor3d(0.6, 0.6, 0.6);

glLineWidth(2);

glBegin(GL\_LINES); // каждая отдельная пара вершин определяет отрезок; если задано нечетное число вершин, то последняя вершина игнорируется

glVertex2i(x0, y0);

glVertex2i(x1, y1);

glEnd();

drawText(text, (x0 + x1) / 2 + 10, (y0 + y1) / 2 + 10); // отрисовка текста

}

template<class T>

void Graph<T>::RenderingGraph() // отрисовка графа

{

int n = vertList.size();

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

{

setCoord(i, n); // установка координат

}

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

{

for (int j = i + 1; j < n; j++)

{

int a = adjMatrix[i][j];

if (a != 0)

{

draw\_Line\_for\_way(a, vertC[i].x, vertC[i].y, vertC[j].x, vertC[j].y);

}

}

}

drawVertex(n); // отрисовка вершины

}

void reshape(int w, int h) // функция изменения окна

{

Win\_W = w;

Win\_H = h;

glViewport(0, 0, (GLsizei)Win\_W, (GLsizei)Win\_H);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, (GLdouble)Win\_W, 0, (GLdouble)Win\_H);

glutPostRedisplay();

}

void draw\_Text\_for\_menu(string text, int x1, int y1) // отрисовка текста для меню

{

GLvoid\* font = GLUT\_BITMAP\_HELVETICA\_18; // установка шрифта

string s = text;

glRasterPos2i(x1 + 5, y1 - 20);

for (int j = 0; j < s.length(); j++)

glutBitmapCharacter(font, s[j]);

}

/\*/ Координаты и текст для кнопки /\*/

void drawMenu() // отрисовка меню

{

int move\_right = 45;

int height = 330;

glColor3d(0.14, 0.10, 0.45);

glBegin(GL\_QUADS);

glVertex2i(move\_right, height - move\_right - 110);

glVertex2i(move\_right + 145, height - move\_right - 110);

glVertex2i(move\_right + 145, height - move\_right - 80);

glVertex2i(move\_right, height - move\_right - 80);

glEnd();

glColor3d(0.8, 1.0, 0.9);

draw\_Text\_for\_menu("Solution", move\_right, height - move\_right - 82);

glColor3d(0.14, 0.10, 0.45);

glBegin(GL\_QUADS);

glVertex2i(move\_right, height - move\_right - 150);

glVertex2i(move\_right + 145, height - move\_right - 150);

glVertex2i(move\_right + 145, height - move\_right - 120);

glVertex2i(move\_right, height - move\_right - 120);

glEnd();

glColor3d(0.8, 1.0, 0.9);

draw\_Text\_for\_menu("Add element", move\_right, height - move\_right - 122);

glColor3d(0.14, 0.10, 0.45);

glBegin(GL\_QUADS);

glVertex2i(move\_right, height - move\_right - 190);

glVertex2i(move\_right + 145, height - move\_right - 190);

glVertex2i(move\_right + 145, height - move\_right - 160);

glVertex2i(move\_right, height - move\_right - 160);

glEnd();

glColor3d(0.8, 1.0, 0.9);

draw\_Text\_for\_menu("Delete element", move\_right, height - move\_right - 162);

}

void button\_click(int btn, int stat, int x, int y) // реализация кнопок и их расположения

{

int move = 60;

int height = 730;

if (stat == GLUT\_DOWN)

{

if (x > move + 30 && x < move + 145 && y > move + 110 && y < move + 140)

{

graph = graph\_implementation();

}

if (x > move + 30 && x < move + 145 && y > move + 150 && y < move + 180)

{

graph.Print();

}

if (x > move + 30 && x < move + 145 && y > move + 190 && y < move + 220)

{

traveler\_man(ukazatel\_na\_matrix, n, dynamic\_mas, result);

}

if (x > move + 30 && x < move + 145 && y > move + 230 && y < move + 260)

{

int sourceVertex;

int targetVertex;

int edgeWeight;

cout << endl << "Добавить пункт: ";

cin >> sourceVertex;

cout << endl;

int\* sourceVertPtr = &sourceVertex;

cout << "Соединить с пунктом: ";

cin >> targetVertex;

cout << endl;

int\* targetVertPtr = &targetVertex;

if (sourceVertex > kolvo\_vershin || targetVertex > kolvo\_vershin)

{

kolvo\_vershin++;

int\* vertPtr = &kolvo\_vershin;

graph.Add\_Vertex(\*vertPtr);

}

cout << "Длина дороги: ";

cin >> edgeWeight;

cout << endl;

graph.Way\_km(\*sourceVertPtr, \*targetVertPtr, edgeWeight);

}

if (x > move + 30 && x < move + 145 && y > move + 270 && y < move + 300)

{

int sourceVertex;

cout << endl << "Удалим пункт: ";

cin >> sourceVertex;

cout << endl;

int\* sourceVertPtr = &sourceVertex;

if (sourceVertex == kolvo\_vershin)

{

kolvo\_vershin--;

graph.Delete\_this\_Vertex();

}

else cout << "Такого пункта нет. Удалять нечего! " << endl << endl;

}

}

glutPostRedisplay();

}

void watch\_display()

{

glShadeModel(GL\_SMOOTH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, Win\_W, 0, Win\_H);

glViewport(0, 0, Win\_W, Win\_H);

glClearColor(0.52, 0.92, 1.0, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

graph.RenderingGraph();

drawMenu();

glutSwapBuffers();

}

