Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №21**

Дисциплина: Основы теории алгоритмов и структуры данных.

Тема: “Графы”

**Вар.12**

Выполнила работу:

студент группы ИВТ-20-2Б

Ананина Арина Юрьевна

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

1. **Постановка задачи**

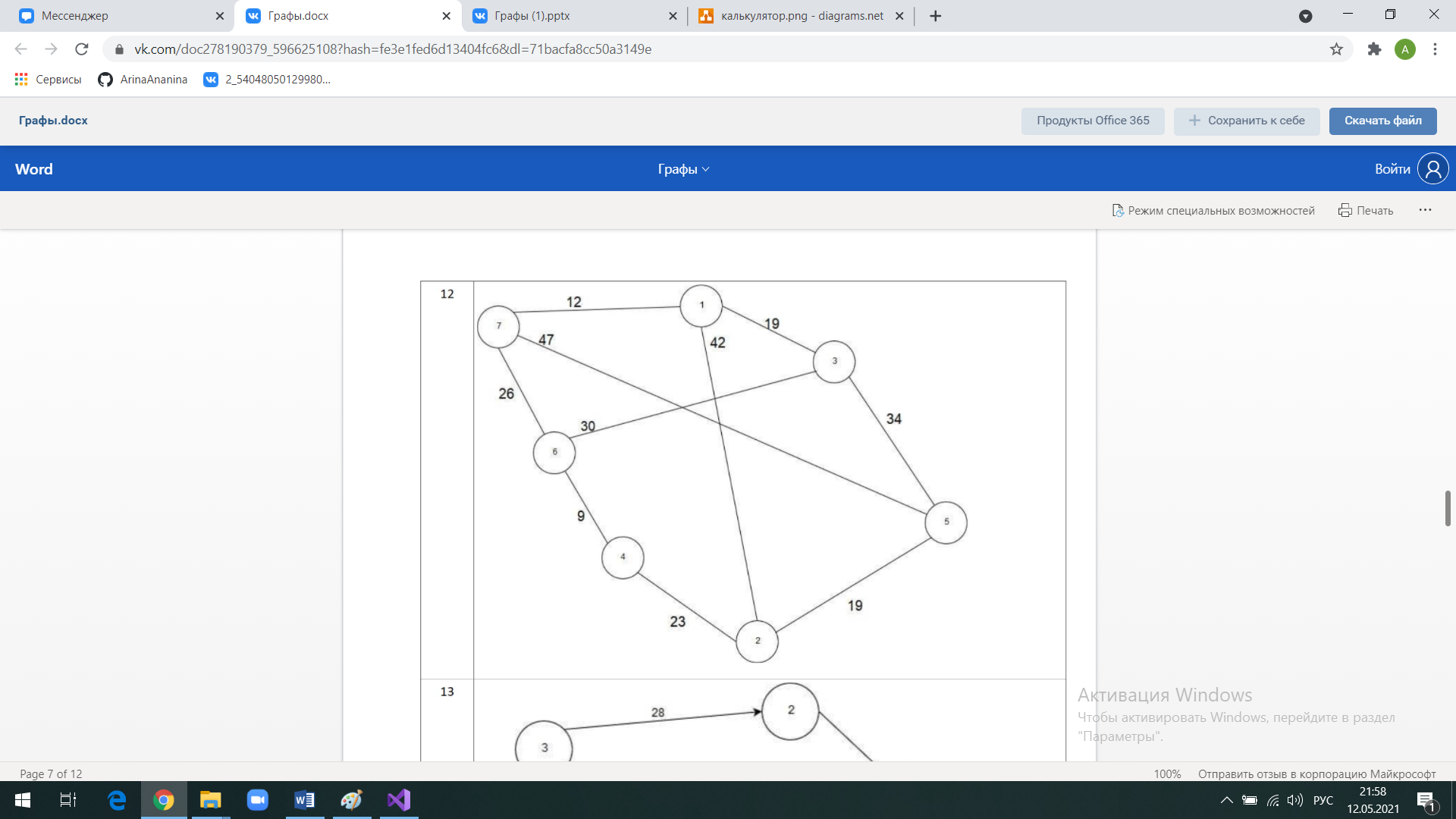
Реализовать алгоритм Дейкстры для графа, соответствующего выбранному варианту.

1. Без использования Qt.

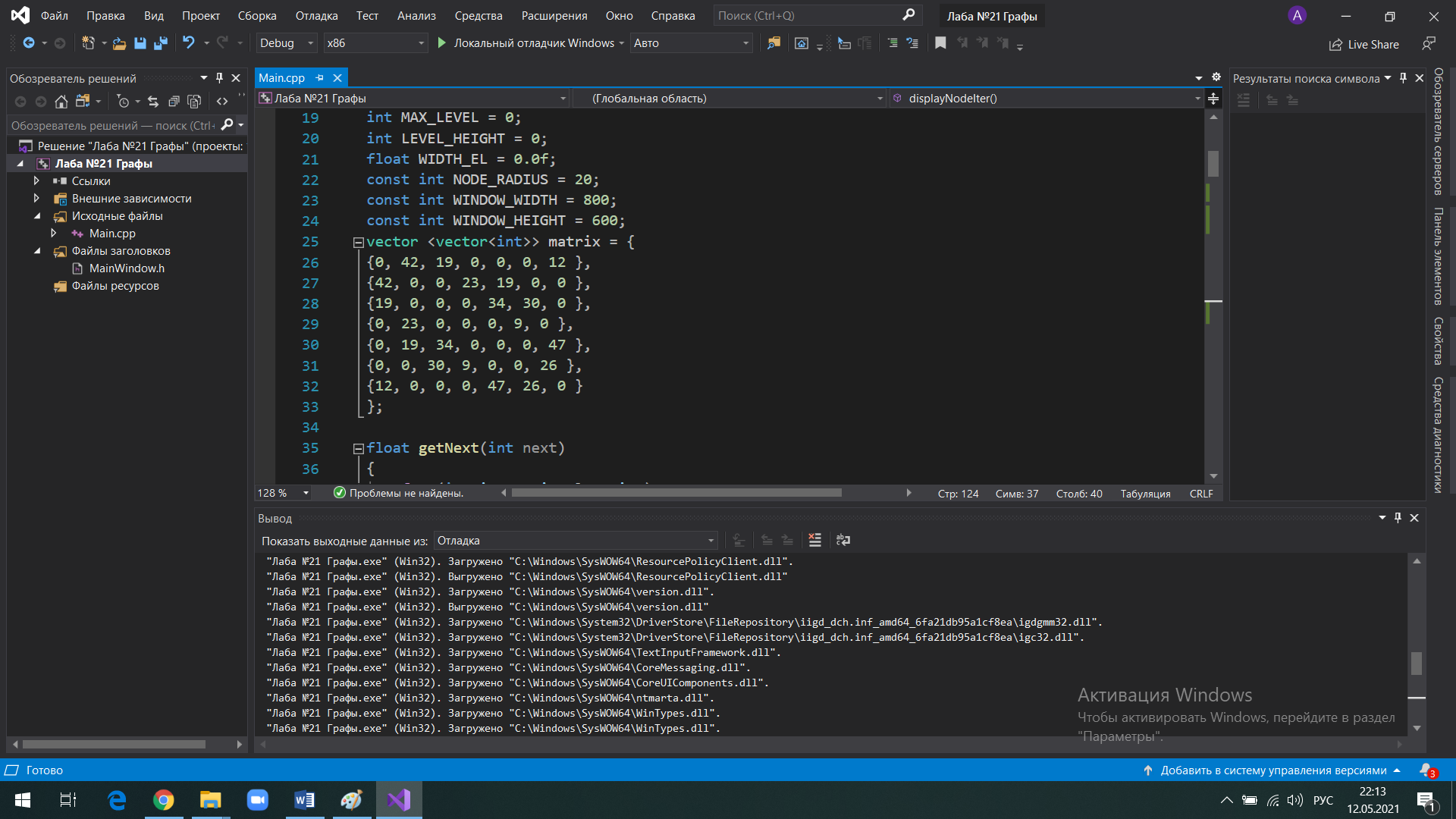
2. Интерфейс на усмотрение разработчика.

3. Визуализация графа в OpenGL.

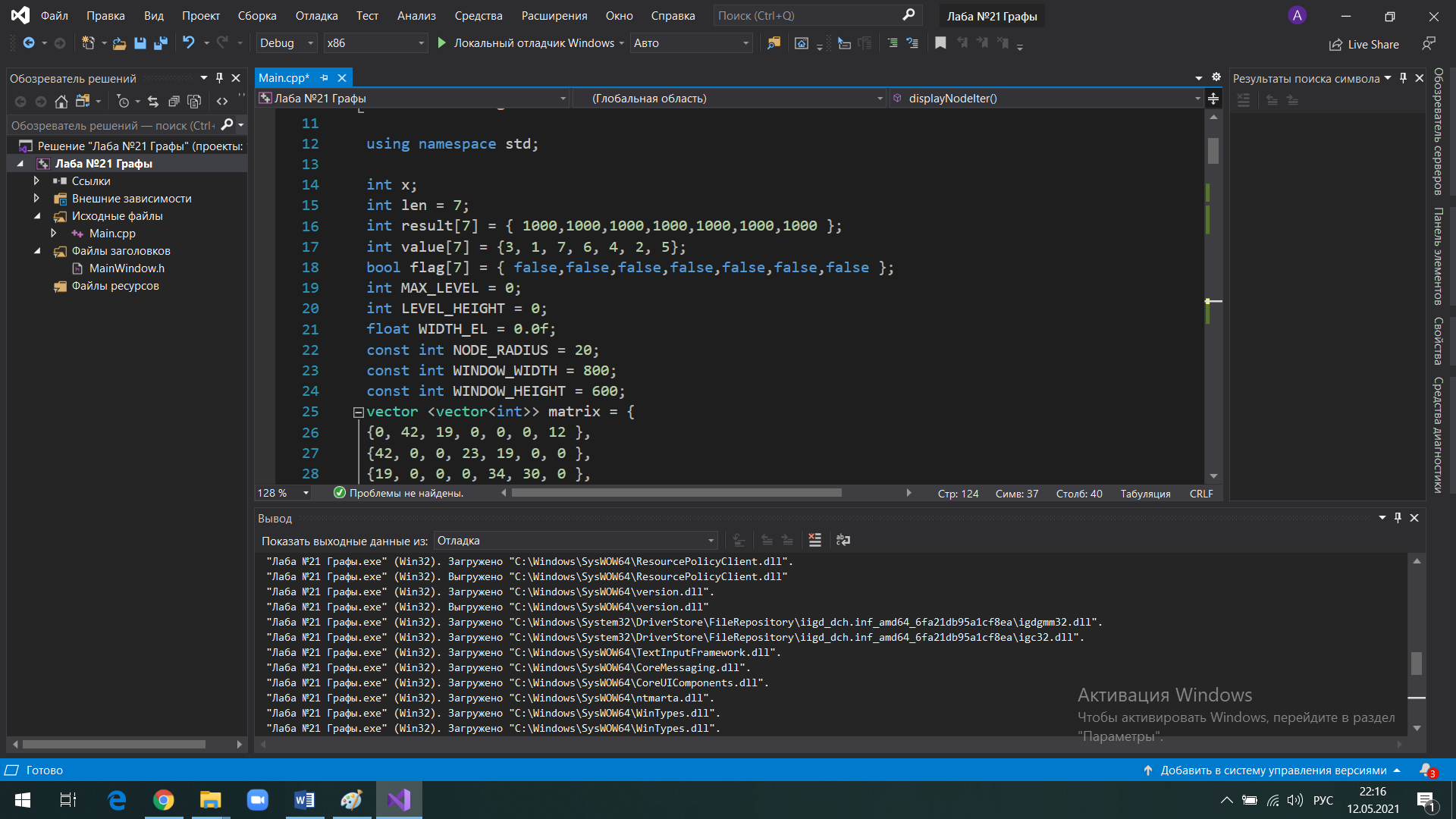
4. Реализуется только метод Дейкстры



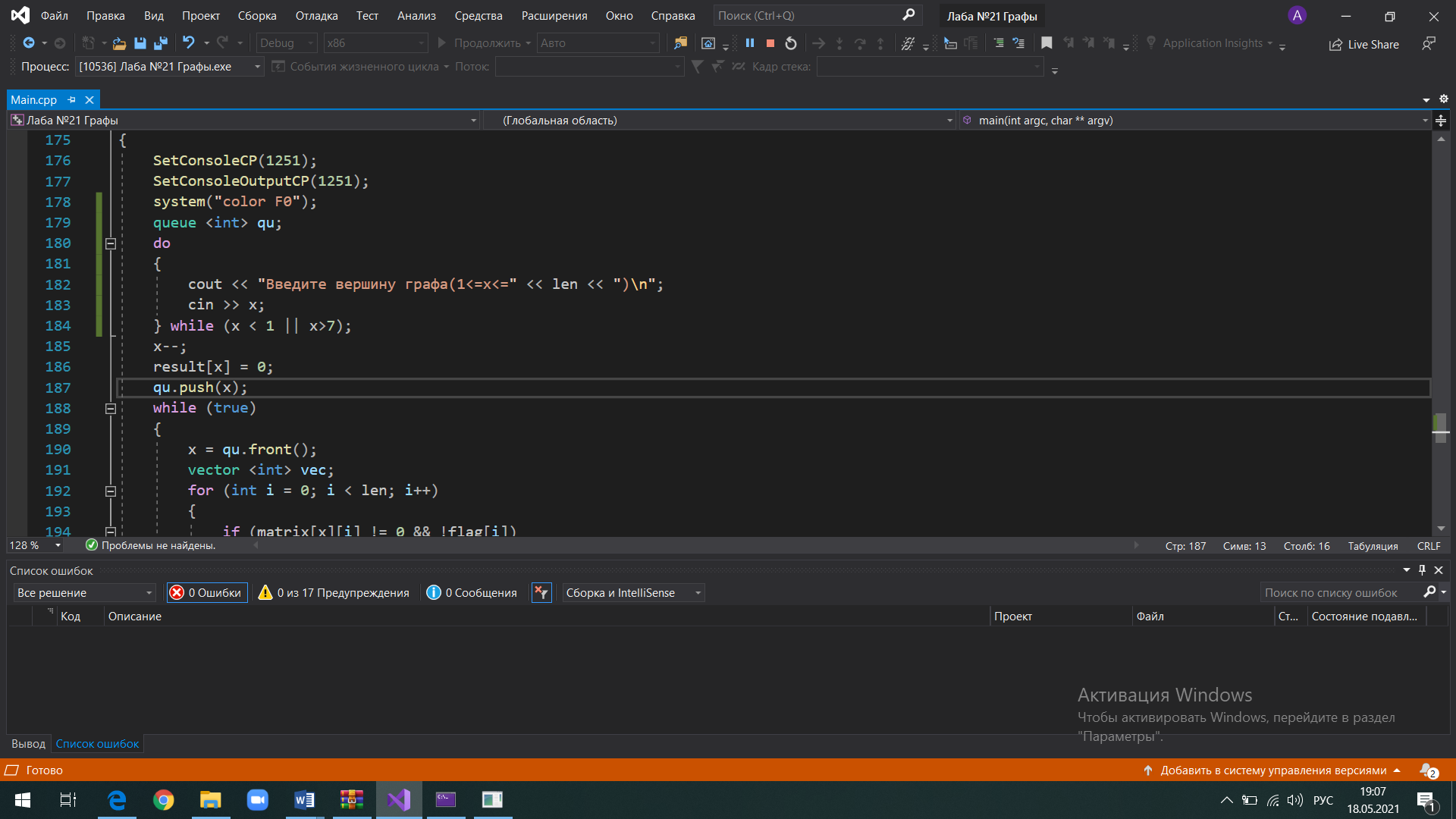
1. **Анализ работы**
2. Какие типы данных использовались.
   1. Использовались типы int, float, char и bool.
3. Какие с этими данными надо выполнить действия.
   1. Переменные типа int используются в качестве счётчика в циклах, для хранения значений, необходимых для отрисовки графа с помощью OpenGL.
   2. Переменные типа float используются для хранения значений координат, высоты, ширины и т.д. ( это также необходимо для отрисовки).
   3. Тип данных char нужен для отрисовки значений длин дорог между городами и отрисовки номера узла.
   4. Тип данных bool используется в массиве из меток, необходимых для обработки графа согласно алгоритму Дейкстры.
4. В каком виде эти данные представлены.
   1. Значения длин дорог представлены в виде матрицы.



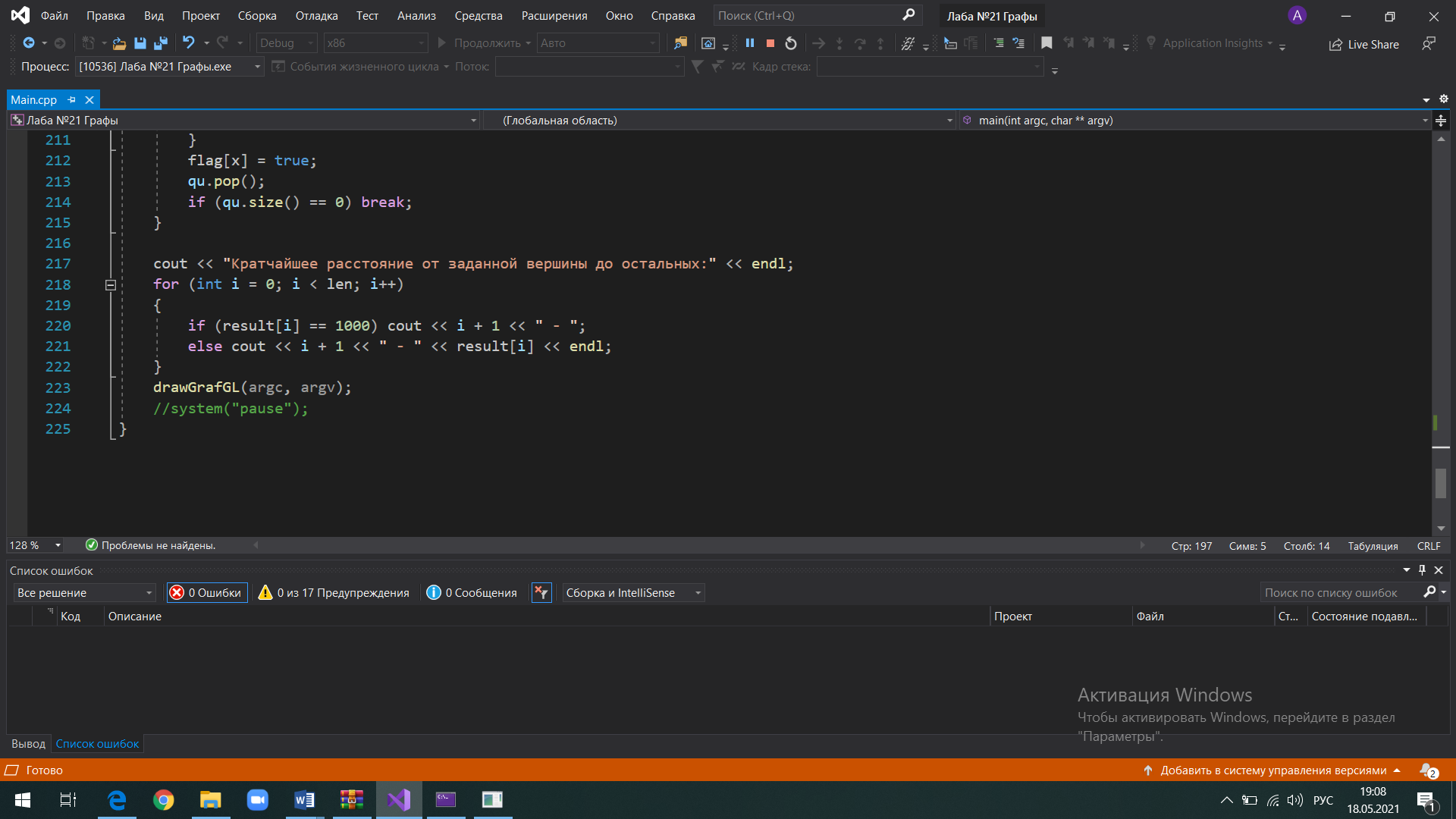
* 1. Длины кратчайших путей из заданного узла будут записываться в массив result. Первоначально все значения этого массива представлены большими числами, для условия алгоритма Дейкстры.



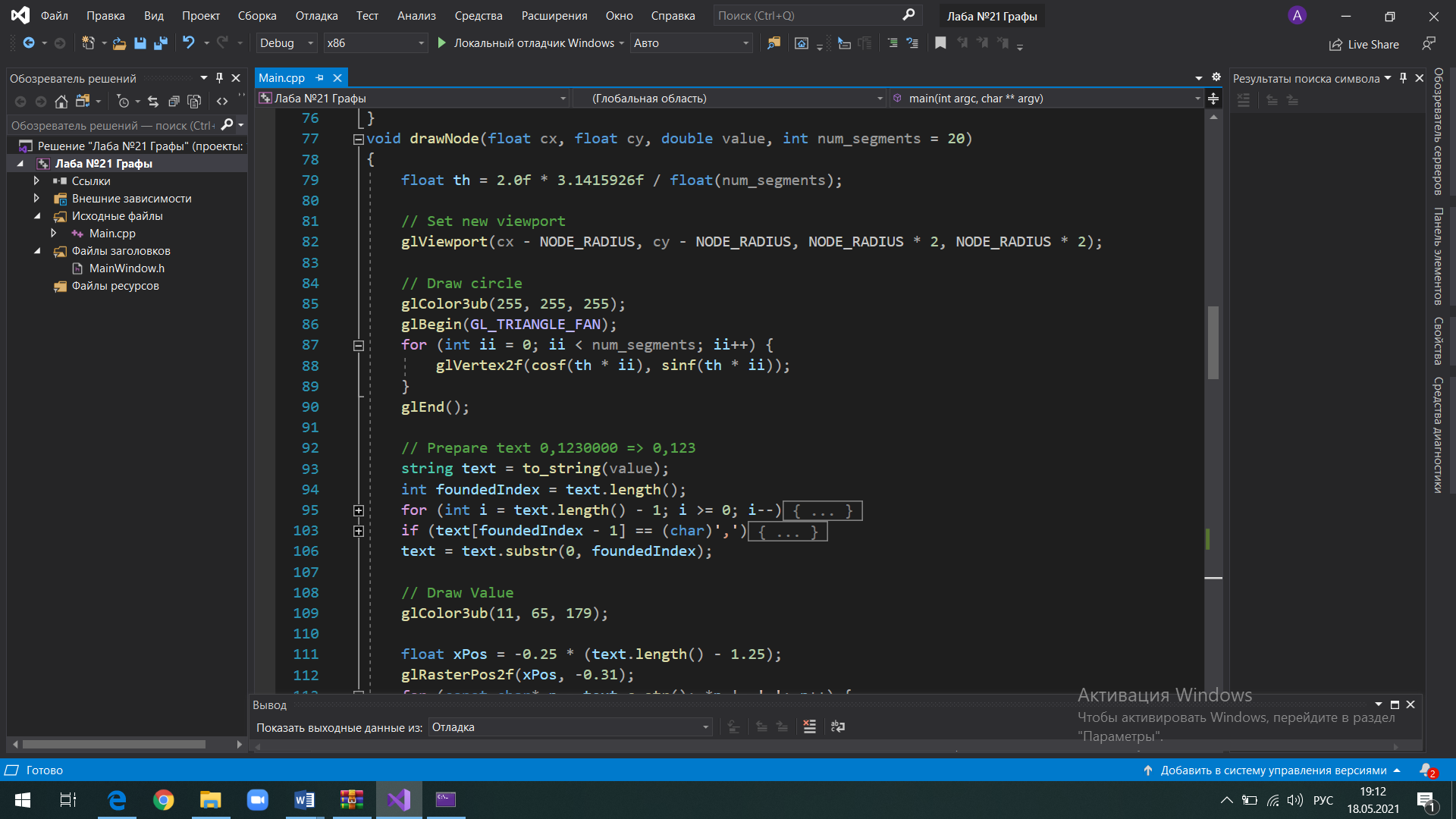
1. Какими операторами организован ввод и вывод.
   1. Ввод данных организован с помощью оператора cout, пользователь ввод номер вершины, от которой необходимо найти длины путей.

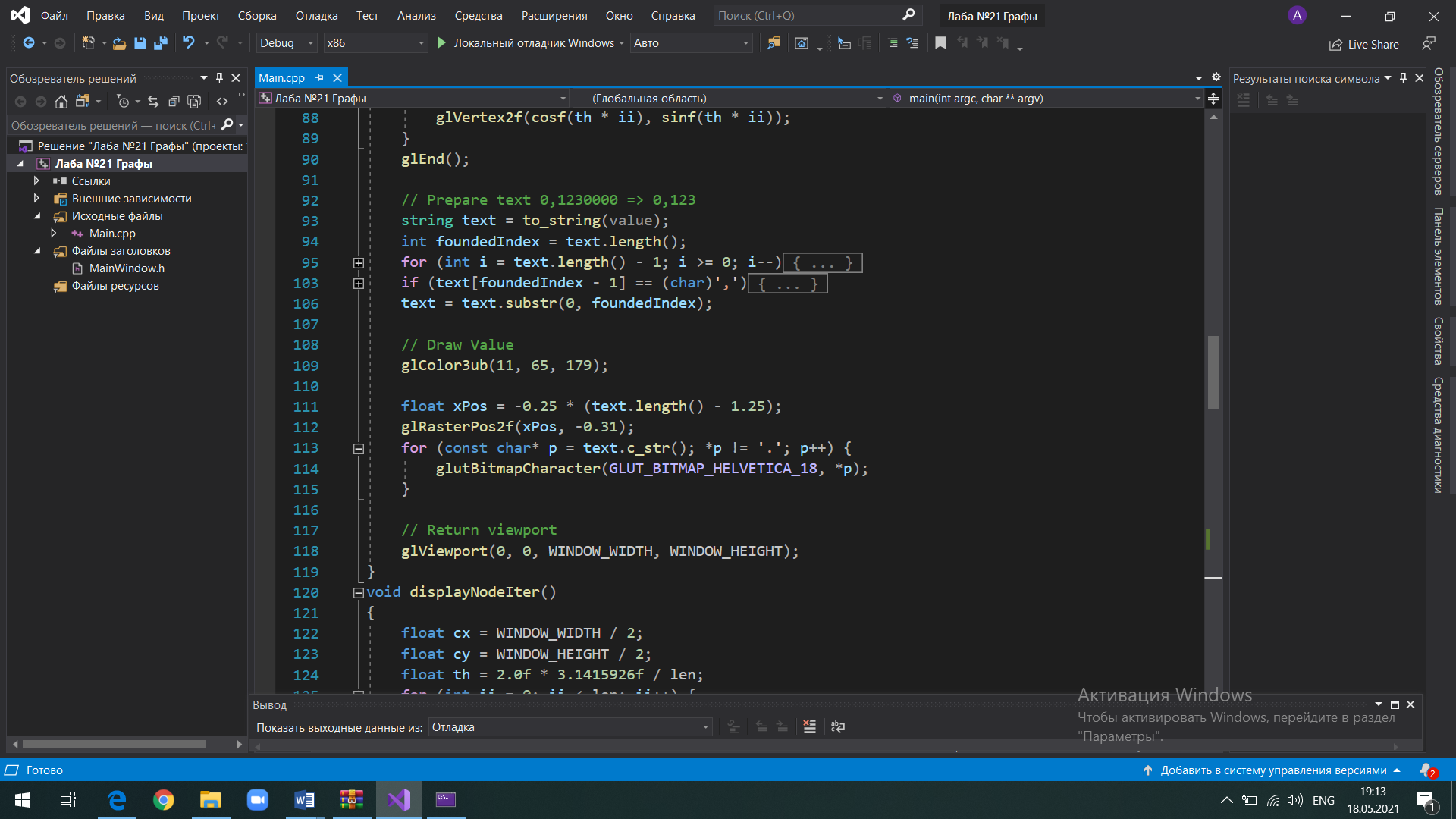


* 1. Вывод организован с помощью оператора cin, выводятся значения кратчайших путей от исходной вершины до всех остальных.



1. Какими действиями и функциями были решены поставленные задачи.
   1. Для отрисовки вершин графа в OpenGL существует функция drawNode()

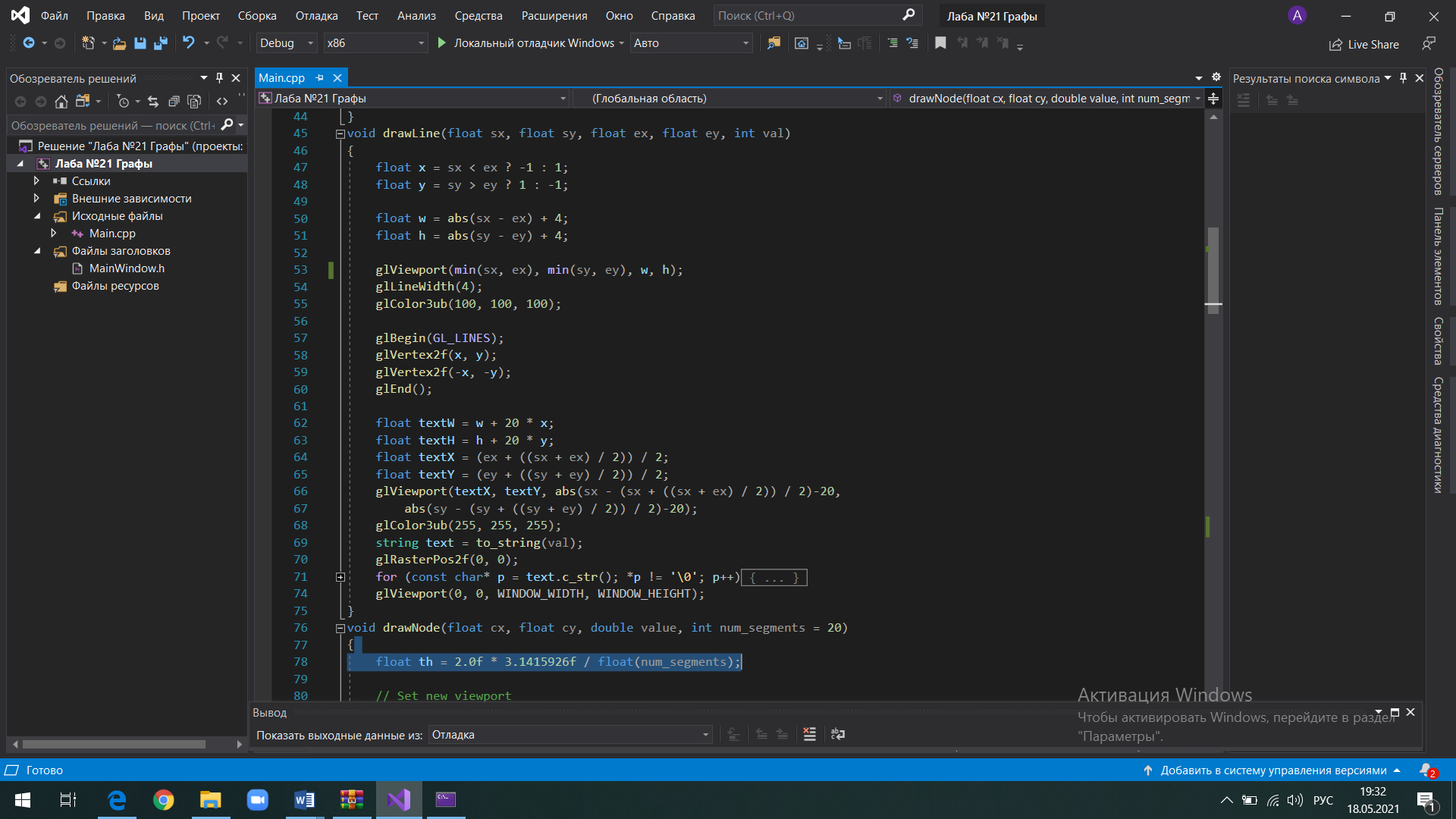




Внутри функции происходит подсчёт координат каждого узла с помощью функций cos и sin, отрисовка круга осуществляется из нескольких треугольников, чем их кол-во больше, тем плавней границы круга, я использую 20 сегментов.

В качестве параметров в эту функцию поступают координаты центра текущего узла и номер вершины.

* 1. Для отрисовки линий используется функция drawLine(), в качестве параметров поступают координаты начала линии и конца, а также длина дороги.



1. **Код программы**

Main.cpp:

#include <iostream>

#include <vector>

#include <windows.h>

#include <queue>

#include <algorithm>

#include <Windows.h>

#include <gl/GL.h>

#include <gl/GLU.h>

#include <glut.h>

#include <string>

using namespace std;

int x;

int len = 7;

int result[7] = { 1000,1000,1000,1000,1000,1000,1000 };

int value[7] = {3, 1, 7, 6, 4, 2, 5};

bool flag[7] = { false,false,false,false,false,false,false };

int MAX\_LEVEL = 0;

int LEVEL\_HEIGHT = 0;

float WIDTH\_EL = 0.0f;

const int NODE\_RADIUS = 20;

const int WINDOW\_WIDTH = 800;

const int WINDOW\_HEIGHT = 600;

vector <vector<int>> matrix = {

{0, 42, 19, 0, 0, 0, 12 },

{42, 0, 0, 23, 19, 0, 0 },

{19, 0, 0, 0, 34, 30, 0 },

{0, 23, 0, 0, 0, 9, 0 },

{0, 19, 34, 0, 0, 0, 47 },

{0, 0, 30, 9, 0, 0, 26 },

{12, 0, 0, 0, 47, 26, 0 }

};

float getNext(int next)

{

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

{

if (next + 1 == value[i])

{

return i;

}

}

}

void drawLine(float sx, float sy, float ex, float ey, int val)

{

float x = sx < ex ? -1 : 1;

float y = sy > ey ? 1 : -1;

float w = abs(sx - ex) + 4;

float h = abs(sy - ey) + 4;

glViewport(min(sx, ex), min(sy, ey), w, h);

glLineWidth(4);

glColor3ub(100, 100, 100);

glBegin(GL\_LINES);

glVertex2f(x, y);

glVertex2f(-x, -y);

glEnd();

float textW = w + 20 \* x;

float textH = h + 20 \* y;

float textX = (ex + ((sx + ex) / 2)) / 2;

float textY = (ey + ((sy + ey) / 2)) / 2;

glViewport(textX, textY, abs(sx - (sx + ((sx + ex) / 2)) / 2)-20,

abs(sy - (sy + ((sy + ey) / 2)) / 2)-20);

glColor3ub(255, 255, 255);

string text = to\_string(val);

glRasterPos2f(0, 0);

for (const char\* p = text.c\_str(); \*p != '\0'; p++) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_12, \*p);

}

glViewport(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT);

}

void drawNode(float cx, float cy, double value, int num\_segments = 20)

{

float th = 2.0f \* 3.1415926f / float(num\_segments);

// Set new viewport

glViewport(cx - NODE\_RADIUS, cy - NODE\_RADIUS, NODE\_RADIUS \* 2, NODE\_RADIUS \* 2);

// Draw circle

glColor3ub(255, 255, 255);

glBegin(GL\_TRIANGLE\_FAN);

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

glVertex2f(cosf(th \* ii), sinf(th \* ii));

}

glEnd();

// Prepare text 0,1230000 => 0,123

string text = to\_string(value);

int foundedIndex = text.length();

for (int i = text.length() - 1; i >= 0; i--) {

if (text[i] == (char)'0') {

foundedIndex = i;

}

else {

continue;

}

}

if (text[foundedIndex - 1] == (char)',') {

foundedIndex++;

}

text = text.substr(0, foundedIndex);

// Draw Value

glColor3ub(11, 65, 179);

float xPos = -0.25 \* (text.length() - 1.25);

glRasterPos2f(xPos, -0.31);

for (const char\* p = text.c\_str(); \*p != '.'; p++) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, \*p);

}

// Return viewport

glViewport(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT);

}

void displayNodeIter()

{

float cx = WINDOW\_WIDTH / 2;

float cy = WINDOW\_HEIGHT / 2;

float th = 2.0f \* 3.1415926f / len;

for (int ii = 0; ii < len; ii++) {

for (int j = value[ii]; j < len; j++)

{

if (matrix[value[ii]-1][j] != 0)

{

drawLine(

// ii => 0..len

cx + cosf(th \* ii) \* 200,

cy + sinf(th \* ii) \* 200,

cx + cosf(th \* getNext(j)) \* 200,

cy + sinf(th \* getNext(j)) \* 200,

matrix[value[ii] - 1][j]);

}

}

}

for (int ii = 0; ii < len; ii++) {

drawNode(cx + cosf(th \* ii) \* 200, cy + sinf(th \* ii) \* 200, value[ii]);

}

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT); // очищение

float y = getNext(1);

displayNodeIter();

glFlush();

}

void drawGrafGL(int argc, char\*\* argv) {

//WIDTH\_EL = round(WINDOW\_WIDTH / (pow(2, MAX\_LEVEL - 1) + 1) / 2);

//LEVEL\_HEIGHT = round(WINDOW\_HEIGHT / (MAX\_LEVEL + 1));

glutInit(&argc, argv);

glutInitWindowSize(WINDOW\_WIDTH, WINDOW\_HEIGHT); //создание окна

glutInitWindowPosition(80, 80); // указание позиции окна

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutCreateWindow("Graf");

glutDisplayFunc(display);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glutMainLoop();

}

bool comp(int a, int b)

{

return matrix[x][a] < matrix[x][b];

}

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

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

queue <int> qu;

cout << "Введите вершину графа(1<=x<=" << len << ")\n";

cin >> x;

x--;

result[x] = 0;

qu.push(x);

while (true)

{

x = qu.front();

vector <int> vec;

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

{

if (matrix[x][i] != 0 && !flag[i])

{

vec.push\_back(i);

}

}

if (vec.size() == 0)

{

flag[x] = true;

qu.pop();

if (qu.size() == 0) break;

continue;

}

sort(vec.begin(), vec.end(), comp);

for (int i = 0; i < vec.size(); i++)

{

result[vec[i]] = min(result[vec[i]], matrix[x][vec[i]] + result[x]);

qu.push(vec[i]);

}

flag[x] = true;

qu.pop();

if (qu.size() == 0) break;

}

cout << "Кратчайшее расстояние от заданной вершины до остальных:" << endl;

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

{

if (result[i] == 1000) cout << i + 1 << " - ";

else cout << i + 1 << " - " << result[i] << endl;

}

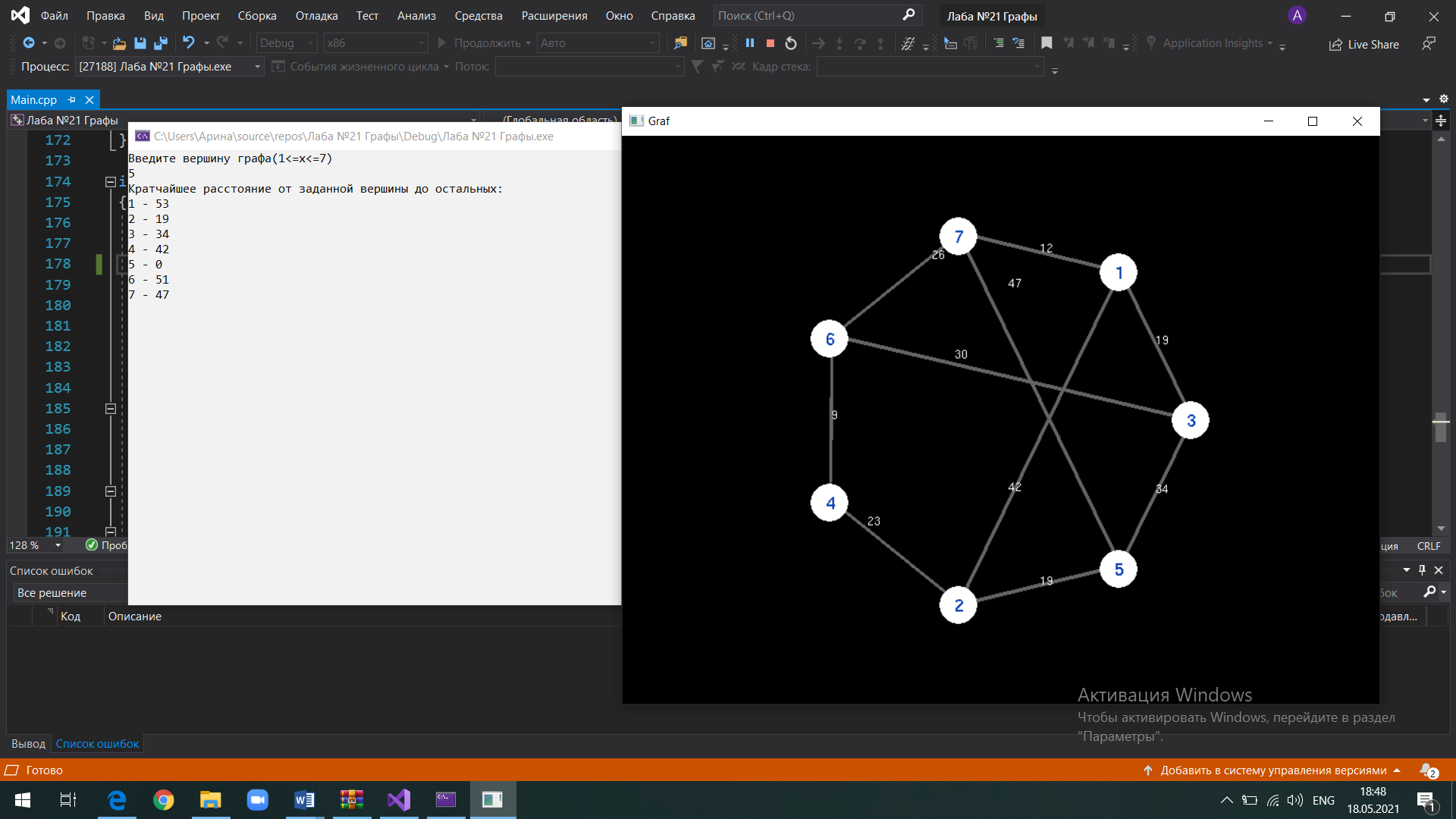
drawGrafGL(argc, argv);

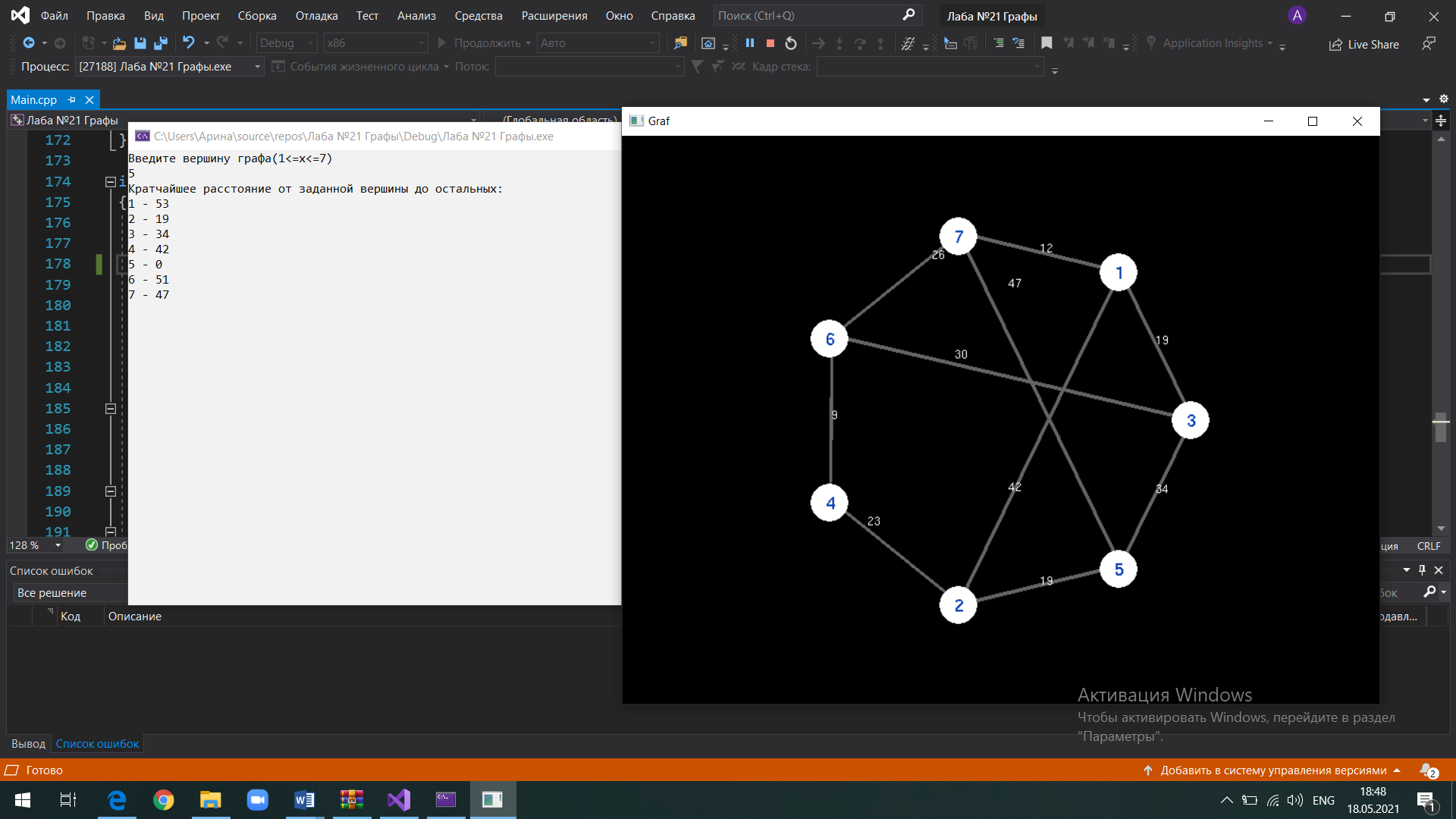
//system("pause");

}

1. **Выполнение программы**

Пользователь выбирает вершину от которой необходимо вычислить кратчайшие расстояния до остальных городов.





При вводе пользователем несуществующей вершины программа будет запрашивать её ввод до тех пор, пока данные не будут введены корректно.

