Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по творческой работе

по теме « Задача Коммивояжера, автоматизированное рабочее место диспетчера службы доставки»

Дисциплина: «Основы алгоритмизации и программирования», 2 семестр

Выполнил

Студент группы ИВТ-22-2Б

Якушев М.В.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

г. Пермь-2023

**1.1 Постановка задачи**

Решить задачу Коммивояжера с визуализацией.

**1.2 Анализ задачи**

Для решение данной задачи был использован метод ветвей и границ, потому что он наиболее эффективный из известных метод отсечения "неперспективных" вершин, за счет анализа матрицы расстояний.

Алгоритм нахождения самого короткого пути.

1. Построение матрицы смежности;

2. Нахождение минимумов строк;

3. Редукция строк;

4. Нахождение минимумов столбцов;

5. Редукция столбцов;

6. Вычисление оценок нулевых элементов;

7. Выбор нулевого элемента с наибольшей оценкой.

8. Запоминаем отрезок пути.

9. Редукция матрицы;

10. Переход к шагу 2 или к шагу 11;

11. Построение маршрута;

12. Вычисление итоговой длины пути.

Функция prepare подготавливает все необходимые вспомогательные массивы и переменные.

В функции answer происходит редукция строк и столбцов, поиск наименьших ребер, оценка нулевых элементов, а также зануление ненужных строк и столбцов.

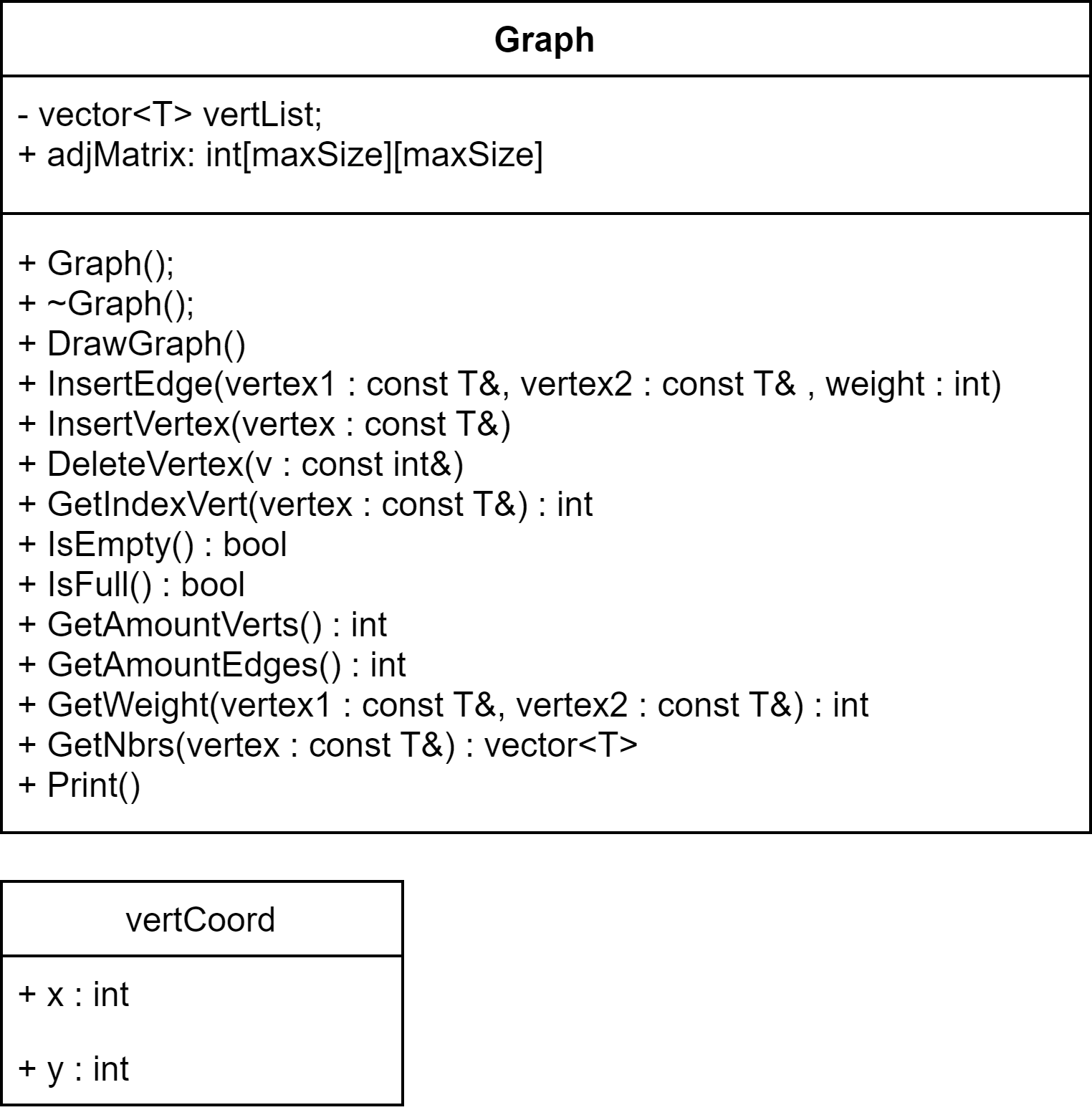
Функции tsalesman включается в себя проверку графа на возможность его обхода, если граф правильный и его возможно обойти не нарушив условия задачи, то будут вызваны функции preparation и answer, и будет получен результат наилучшего пути.

Для визуализации графа был использован OpenGL.

При решении задачи возникло несколько трудностей. В первую очередь, была сложность с реализацией метода ветвей и границ в коде. Также возникла трудность с визуализацией, так как приходилось с нуля изучать возможности OpenGL.

Считаю достоинствами моей реализации то, что программа разделена на части, а также простую и понятную визуализацию. Также считаю, что в моей программе удобное меню, которое работает без ошибок.

**1.3 UML – диаграмма**



**1.4 Исходный код программы**

*ком.cpp*

#include "Graph.h"

#include <iostream>

#include <GL/glut.h>

#include <stdio.h>

#include <vector>

#include <sstream>

using namespace std;

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

{

setlocale(LC\_ALL, "rus");

glutInit(&argc, argv);

graph = makeGraph();

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowSize(1350, 730);

glutCreateWindow("Graph");

WinW = glutGet(GLUT\_WINDOW\_WIDTH);

WinH = glutGet(GLUT\_WINDOW\_HEIGHT);

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMouseFunc(mouseClick);

glutMainLoop();

return 0;

}

*Graph.h*

#pragma once

#include <GL/glut.h>

#include <stdio.h>

#include <iostream>

#include <vector>

#include <sstream>

using namespace std;

int n;

int\*\* help;

int\* result;

int\*\*\* mat;

int WinW;

int WinH;

const int maxSize = 20;

int amountVerts;

template<class T>

class Graph

{

vector<T> vertList;

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

public:

int adjMatrix[maxSize][maxSize] = { 0 };

Graph();

~Graph();

void DrawGraph();

void InsertEdge(const T& vertex1, const T& vertex2, int weight);

void InsertVertex(const T& vertex);

void DeleteVertex(const int&);

int GetIndexVert(const T& vertex);

bool IsEmpty();

bool IsFull();

int GetAmountVerts();

int GetAmountEdges();

int GetWeight(const T& vertex1, const T& vertex2);

vector<T> GetNbrs(const T& vertex);

void Print();

};

int R;

struct vertCoord

{

int x, y;

};

vertCoord vertC[20];

Graph<int> graph;

void Answer(int\*\*\* mat, int n, int\*\* help, int\* path)

{

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

{

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

{

int min = 1000000;

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

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

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

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

if (mat[i][j])

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

}

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

{

int min = 1000000;

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

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

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

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

if (mat[i][j])

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

}

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

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

help[i][j] = 0;

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

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

{

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

{

int hmin = 1000000;

int vmin = 1000000;

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

if (l != i && mat[l][j] && hmin > \*mat[l][j])

hmin = \*mat[l][j];

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

if (l != j && mat[i][l] && vmin > \*mat[i][l])

vmin = \*mat[i][l];

help[i][j] = hmin + vmin;

}

}

int mcost = 0, mi = 0, mj = 0;

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

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

if (mat[i][j] && mcost < help[i][j])

{

mcost = help[i][j];

mi = i;

mj = j;

}

path[mi] = mj;

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

mat[i][mj] = nullptr;

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

mat[mi][i] = nullptr;

mat[mj][mi] = nullptr;

}

}

void prepare(int\*\*\*& mat, int& n, int\*\*& help, int\*& result)

{

n = amountVerts;

help = new int\* [n];

result = new int[n];

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

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

{

help[i] = new int[n];

}

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

{

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

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

{

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

mat[i][j] = nullptr;

continue;

}

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

}

}

}

void tsalesman(int\*\*\* mat, int n, int\*\* help, int\* result)

{

int count = 0;

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

{

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

{

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

{

count++;

}

}

if (count <=1)

{

cout << "Ошибка: невозможно обойти граф!" <<endl;

count = 0;

return;

}

count = 0;

}

prepare(mat, n, help, result);

int s = 0;

Answer(mat, n, help, result);

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

{

j = result[i];

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

}

cout << endl;

cout << "\nЛучший путь: ";

int temp = 0;

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 << "\nПройденное расстояние: " << s;

cout << endl;

}

template<class T>

std::vector<T> Graph<T>::GetNbrs(const T& vertex) {

std::vector<T> nbrsList;

int vertPos = this->GetIndexVert(vertex);

if (vertPos != (-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>::InsertVertex(const T& vertex) {

if (!this->IsFull()) {

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

}

else {

cout << "Невозможно добавить вершину! " << endl;

return;

}

}

template<class T>

void Graph<T>::DeleteVertex(const int& v) {

int vT = v - 1;

int pos = this->GetIndexVert(v);

if (pos != -1)

{

this->vertList.erase(vertList.begin() + pos);

for (int i = 0; i < maxSize - 1; i++)

{

for (int j = 0; j < maxSize - 1; j++)

{

if (j < vT && i < vT)

{

adjMatrix[i][j] = adjMatrix[i][j];

}

else if (j >= vT && i < vT)

{

adjMatrix[i][j] = adjMatrix[i][j + 1];

}

else if (j < vT && i >= vT)

{

adjMatrix[i][j] = adjMatrix[i + 1][j];

}

else if (j >= vT && i >= vT)

{

adjMatrix[i][j] = adjMatrix[i + 1][j + 1];

}

}

}

}

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

if (i >= vT)

this->vertList[i] = this->vertList[i] - 1;

}

template<class T>

int Graph<T>::GetAmountEdges() {

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++;

}

}

return (amount/2);

}

else

return 0;

}

template<class T>

int Graph<T>::GetWeight(const T& vertex1, const T& vertex2) {

if (!this->IsEmpty()) {

int vertPos1 = GetIndexVert(vertex1);

int vertPos2 = GetIndexVert(vertex2);

return adjMatrix[vertPos1][vertPos2];

}

return 0;

}

template<class T>

int Graph<T>::GetAmountVerts() {

return this->vertList.size();

}

template<class T>

bool Graph<T>::IsEmpty() {

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

return false;

else

return true;

}

template<class T>

bool Graph<T>::IsFull() {

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

}

template <class T>

int Graph<T>::GetIndexVert(const T& vertex) {

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> makeGraph()

{

Graph<int> graph;

int amountEdges, sourceVertex, targetVertex, edgeWeight,vertex;

cout << "Введите кол-во вершин графа: "; cin >> amountVerts; cout << endl;

cout << "Введите кол-во ребер графа: "; cin >> amountEdges; cout << endl;

for (int i = 1; i <= amountVerts; ++i) {

int\* vertPtr = &i;

graph.InsertVertex(\*vertPtr);

}

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

cout << "Исходная вершина: "; cin >> sourceVertex; cout << endl;

int\* sourceVertPtr = &sourceVertex;

cout << "Конечная вершина: "; cin >> targetVertex; cout << endl;

int\* targetVertPtr = &targetVertex;

cout << "Вес ребра: "; cin >> edgeWeight; cout << endl;

graph.InsertEdge(\*sourceVertPtr, \*targetVertPtr, edgeWeight);

}

cout << endl;

return graph;

}

template<class T>

void Graph<T>::InsertEdge(const T& vertex1, const T& vertex2, int weight) {

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

int vertPos1 = GetIndexVert(vertex1);

int vertPos2 = GetIndexVert(vertex2);

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

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

cout << "Ребро между вершинами уже есть!" << endl;

return;

}

else {

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

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

}

}

else {

cout << "Обеих вершин или одной из них нет в графе!" << endl;

return;

}

}

template<class T>

void Graph<T>::Print() {

if (!this->IsEmpty()) {

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

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

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

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

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

}

cout << endl;

}

}

}

void setCoord(int i, int n)

{

int R\_;

int x0 = WinW / 2;

int y0 = WinH / 2;

if (WinW > WinH)

{

R = 5 \* (WinH / 13) / n;

R\_ = WinH / 2 - R - 10;

}

else {

R = 5 \* (WinW / 13) / n;

R\_ = WinW / 2 - R - 10;

}

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

float y1 = R\_ \* cos(theta) + y0;

float x1 = R\_ \* sin(theta) + x0;

vertC[i].x = x1;

vertC[i].y = y1;

}

void drawCircle(int x, int y, int R)

{

glColor3f(0.1, 0.9, 0.8);

float x1, y1;

glBegin(GL\_POLYGON);

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

{

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

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

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

glVertex2f(x1, y1);

}

glEnd();

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

float x2, y2;

glBegin(GL\_LINE\_LOOP);

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)

{

GLvoid\* font = GLUT\_BITMAP\_HELVETICA\_18;

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)

{

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

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

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

}

}

void drawLine(int text, int x0, int y0, int x1, int y1)

{

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

glBegin(GL\_LINES);

glVertex2i(x0, y0);

glVertex2i(x1, y1);

glEnd();

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

}

template<class T>

void Graph<T>::DrawGraph()

{

int n = vertList.size();

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

{

setCoord(i, n);

}

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

{

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

{

int a = adjMatrix[i][j];

if (a != 0)

{

drawLine(a, vertC[i].x, vertC[i].y, vertC[j].x, vertC[j].y);

}

}

}

drawVertex(n);

}

void reshape(int w, int h)

{

WinW = w;

WinH = h;

glViewport(0, 0, (GLsizei)WinW, (GLsizei)WinH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, (GLdouble)WinW, 0, (GLdouble)WinH);

glutPostRedisplay();

}

void drawMenuText(string text, int x1, int y1)

{

GLvoid\* font = GLUT\_BITMAP\_HELVETICA\_18;

string s = text;

glRasterPos2i(x1 + 5, y1 - 20);

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

glutBitmapCharacter(font, s[j]);

}

void drawMenu()

{

int shift = 60;

int height = 730;

glColor3d(0.2, 0.7, 0.3);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 140);

glVertex2i(shift + 135, height - shift - 140);

glVertex2i(shift + 135, height - shift - 110);

glVertex2i(shift, height - shift - 110);

glEnd();

glColor3d(0.8, 1.0, 0.9);

drawMenuText("Add edge", shift, height - shift - 112);

glColor3d(0.2, 0.7, 0.3);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 210);

glVertex2i(shift + 135, height - shift - 210);

glVertex2i(shift + 135, height - shift - 180);

glVertex2i(shift, height - shift - 180);

glEnd();

glColor3d(0.8, 1.0, 0.9);

drawMenuText("Add vertex", shift, height - shift - 182);

glColor3d(0.2, 0.7, 0.3);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 280);

glVertex2i(shift + 135, height - shift - 280);

glVertex2i(shift + 135, height - shift-250);

glVertex2i(shift, height - shift-250);

glEnd();

glColor3d(0.8, 1.0, 0.9);

drawMenuText("Insert element", shift, height - shift - 252);

glColor3d(0.2, 0.7, 0.3);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 350);

glVertex2i(shift + 135, height - shift - 350);

glVertex2i(shift + 135, height - shift - 320);

glVertex2i(shift, height - shift - 320);

glEnd();

glColor3d(0.8, 1.0, 0.9);

drawMenuText("Delete element", shift, height - shift - 322);

glColor3d(0.2, 0.7, 0.3);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 420);

glVertex2i(shift + 135, height - shift - 420);

glVertex2i(shift + 135, height - shift - 390);

glVertex2i(shift, height - shift - 390);

glEnd();

glColor3d(0.8, 1.0, 0.9);

drawMenuText("Matrix", shift, height - shift - 392);

glColor3d(0.2, 0.7, 0.3);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 490);

glVertex2i(shift + 135, height - shift - 490);

glVertex2i(shift + 135, height - shift - 460);

glVertex2i(shift, height - shift - 460);

glEnd();

glColor3d(0.8, 1.0, 0.9);

drawMenuText("Answer", shift, height - shift - 462);

glColor3d(0.2, 0.7, 0.3);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 560);

glVertex2i(shift + 135, height - shift - 560);

glVertex2i(shift + 135, height - shift - 530);

glVertex2i(shift, height - shift - 530);

glEnd();

glColor3d(0.8, 1.0, 0.9);

drawMenuText("New graph", shift, height - shift - 532);

}

void mouseClick(int btn, int stat, int x, int y) {

int shift = 60;

int height = 730;

if (stat == GLUT\_DOWN) {

if (x > shift && x < shift + 135 && y > shift + 110 && y < shift + 140)

{

int sourceVertex;

int targetVertex;

int edgeWeight;

cout << "Исходная вершина: "; cin >> sourceVertex; cout << endl;

int\* sourceVertPtr = &sourceVertex;

cout << "Конечная вершина: "; cin >> targetVertex; cout << endl;

int\* targetVertPtr = &targetVertex;

cout << "Вес ребра: "; cin >> edgeWeight; cout << endl;

graph.InsertEdge(\*sourceVertPtr, \*targetVertPtr, edgeWeight);

}

if (x > shift && x < shift + 135 && y > shift + 180 && y < shift + 210)

{

int vertex = amountVerts + 1;

amountVerts++;

graph.InsertVertex(vertex);

}

if (x > shift && x < shift + 135 && y > shift +250&& y < shift + 280)

{

int sourceVertex;

int targetVertex;

int edgeWeight;

cout << "Исходная вершина: "; cin >> sourceVertex; cout << endl;

int\* sourceVertPtr = &sourceVertex;

cout << "Конечная вершина: "; cin >> targetVertex; cout << endl;

int\* targetVertPtr = &targetVertex;

if (sourceVertex > amountVerts || targetVertex > amountVerts) {

amountVerts++;

int\* vertPtr = &amountVerts;

graph.InsertVertex(\*vertPtr);

}

cout << "Вес ребра: "; cin >> edgeWeight; cout << endl;

graph.InsertEdge(\*sourceVertPtr, \*targetVertPtr, edgeWeight);

}

if (x > shift && x < shift + 135 && y > shift + 320 && y < shift + 350)

{

int sourceVertex;

int targetVertex;

int edgeWeight;

cout << "Удаление вершины: "; cin >> sourceVertex; cout << endl;

int\* sourceVertPtr = &sourceVertex;

if (sourceVertex > 0 && sourceVertex <= amountVerts)

{

amountVerts--;

graph.DeleteVertex(sourceVertex);

}

else cout << "Такой вершины нет! \n";

}

if (x > shift && x < shift + 135 && y > shift + 390 && y < shift + 410)

{

graph.Print();

}

if (x > shift && x < shift + 135 && y > shift + 460 && y < shift + 480)

{

tsalesman(mat, n, help, result);

}

if (x > shift && x < shift + 135 && y > shift + 530 && y < shift + 550)

{

graph = makeGraph();

}

}

glutPostRedisplay();

}

void display()

{

glShadeModel(GL\_SMOOTH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, WinW, 0, WinH);

glViewport(0, 0, WinW, WinH);

glClearColor(1.0, 1.0, 0.7, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

graph.DrawGraph();

drawMenu();

glutSwapBuffers();

}

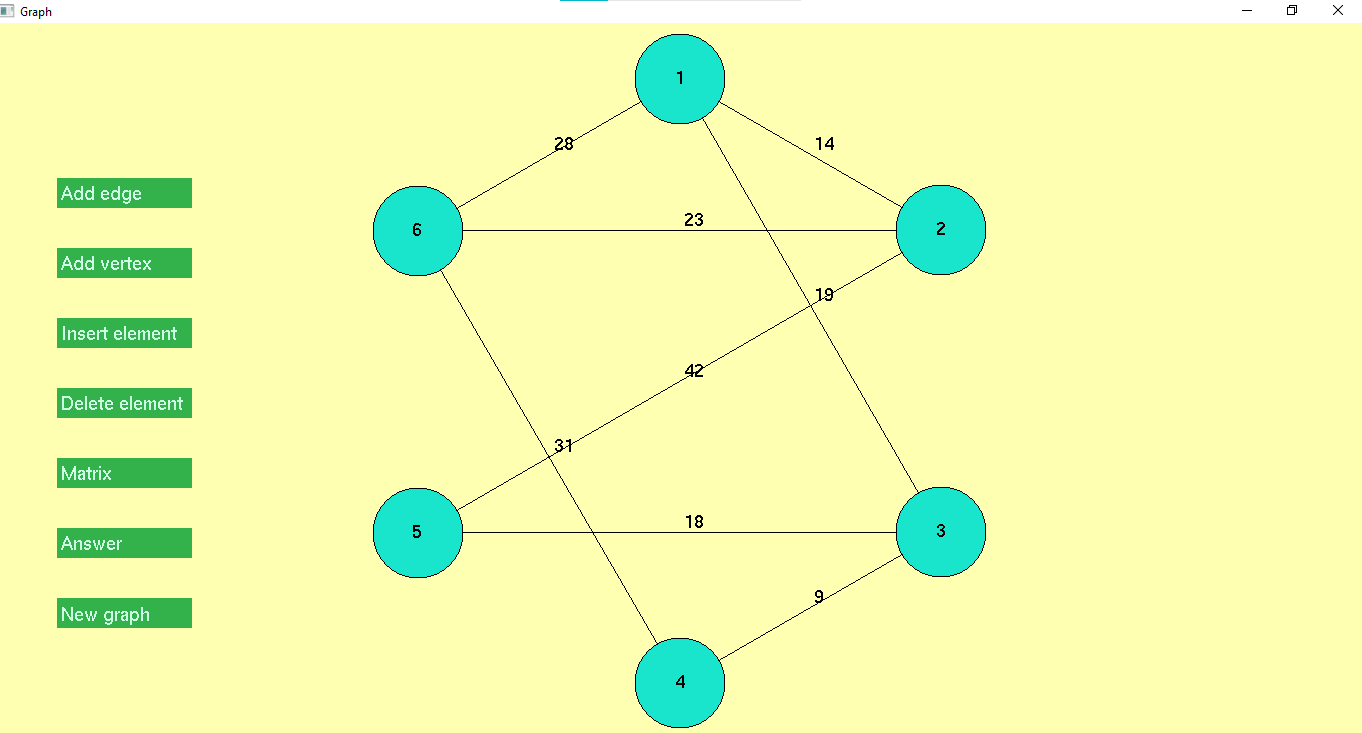
void visual()

{

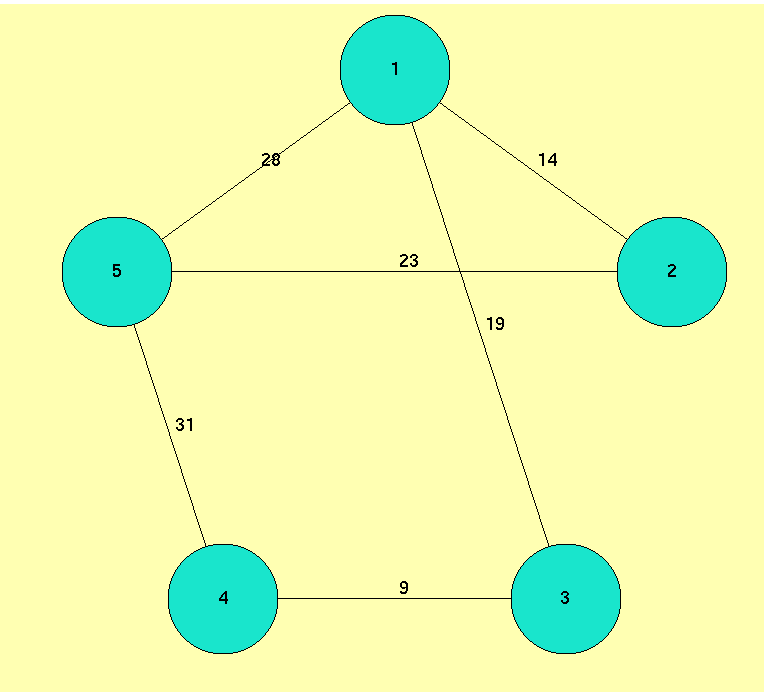
}

**1.5 Скриншоты работы программы**

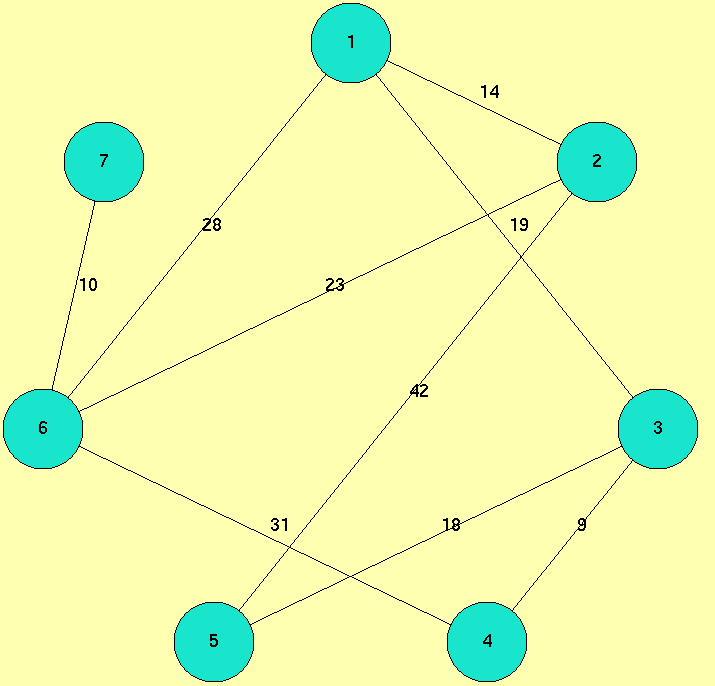
Программа после запуска

****

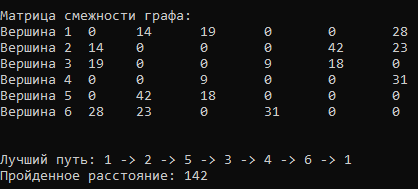
Программа после удаления вершины



Программа после добавления вершины и пути



Матрица смежности и наилучший путь



**2.1 Постановка задачи**

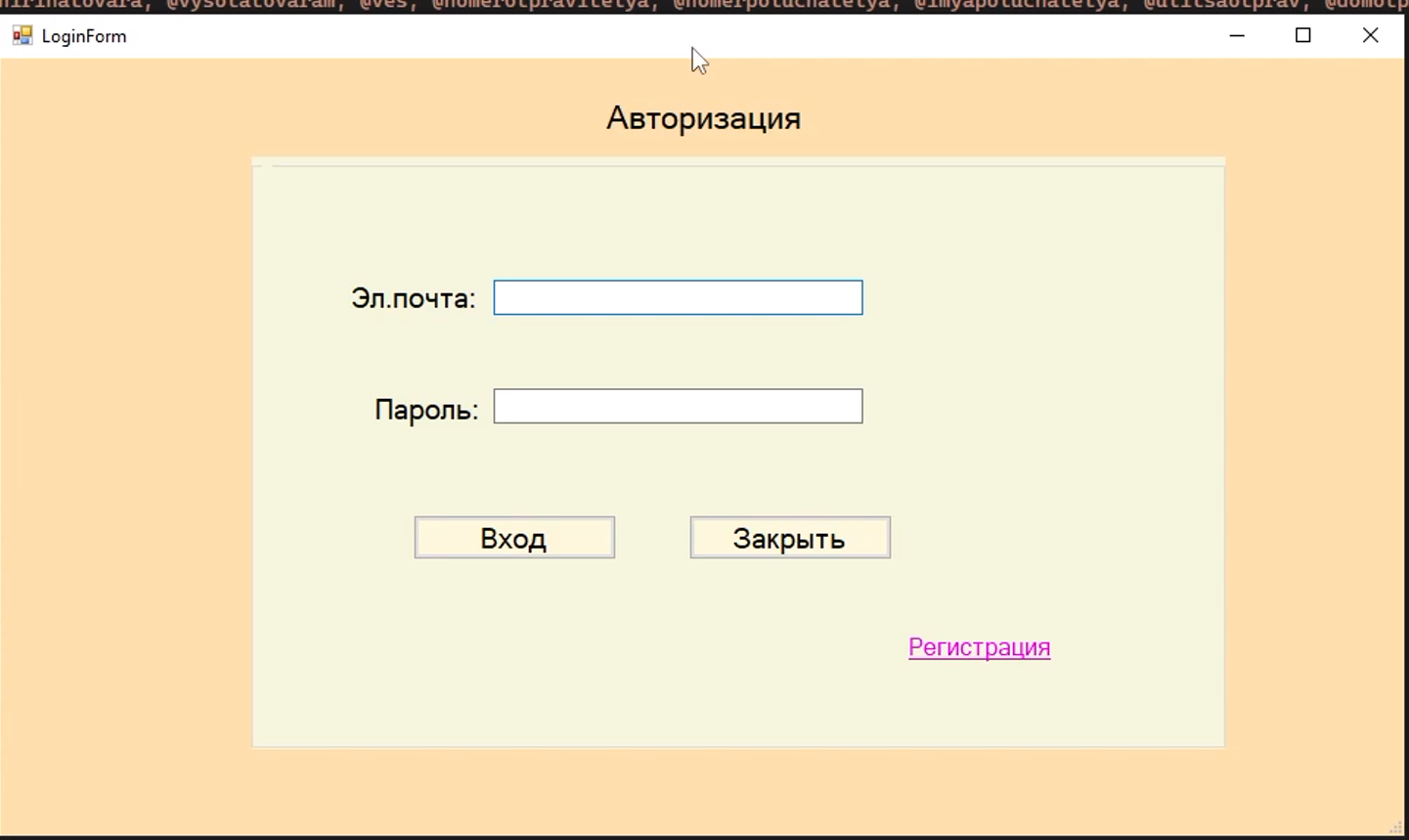
Реализовать автоматизированное рабочее место диспетчера службы доставки.

**2.2 Анализ задачи**

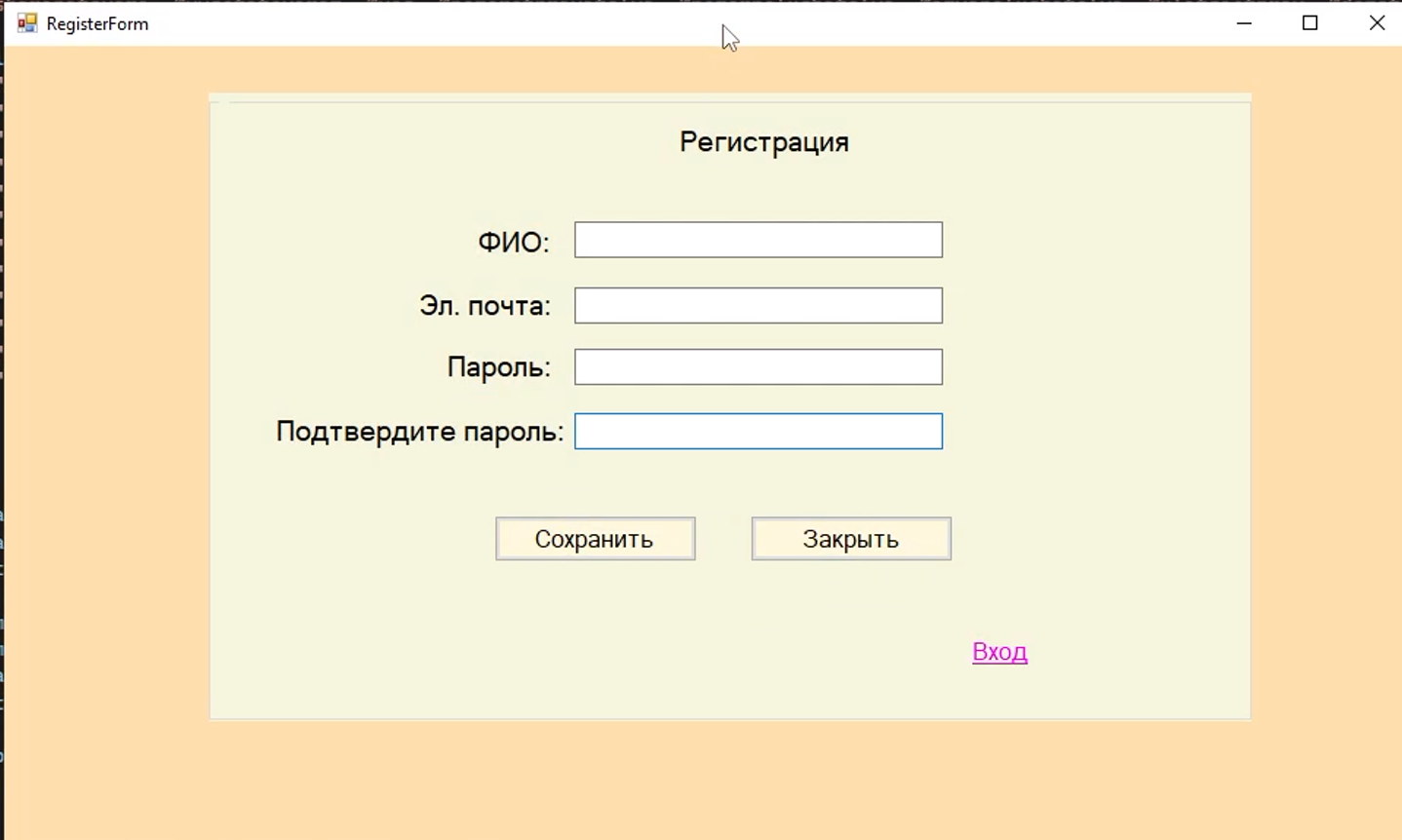
Программа была реализована на языке программирования С++. Для визуализации используется Windows Forms.

**2.3 Работа программы**

Окно авторизации

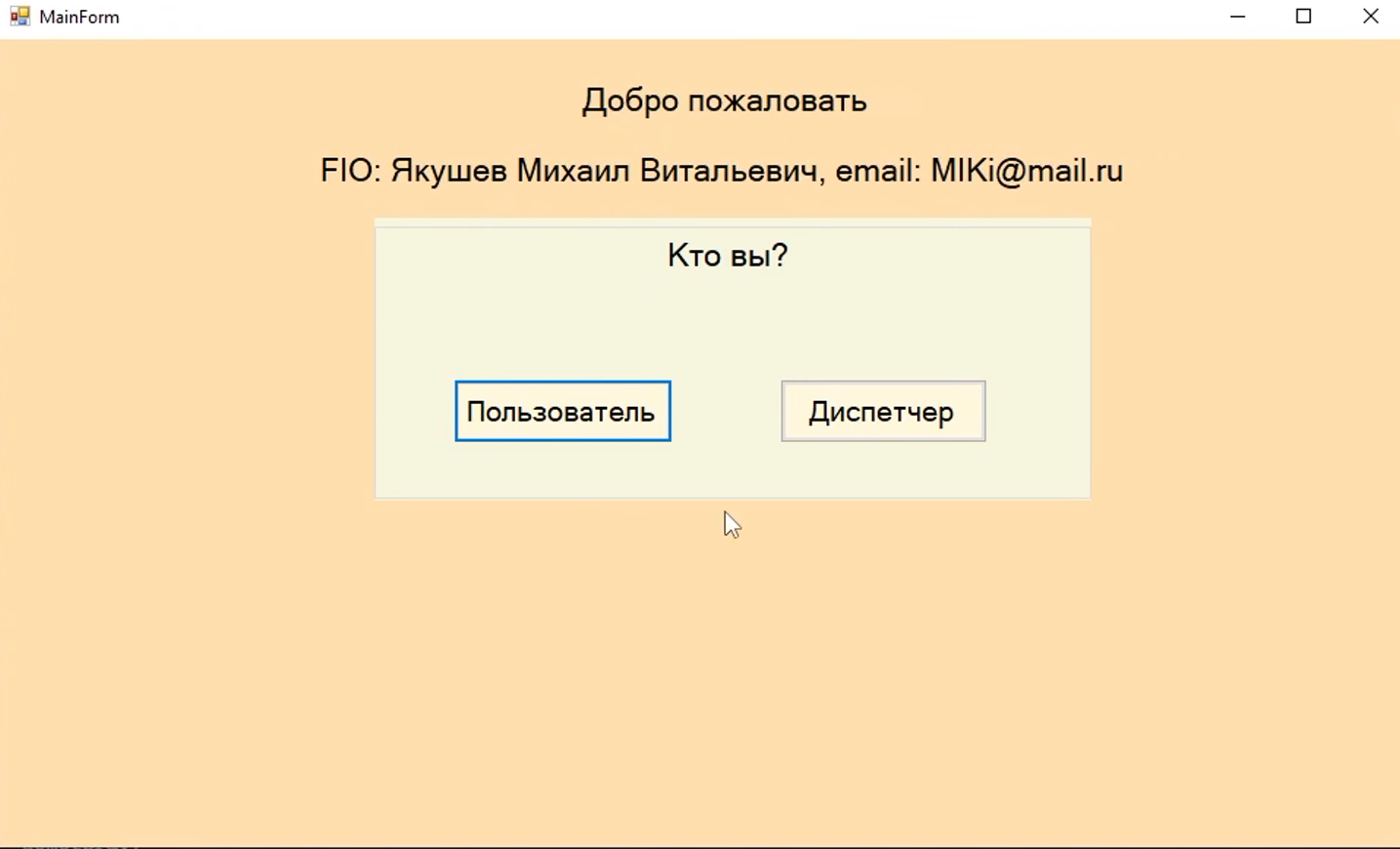


Окно регистрации

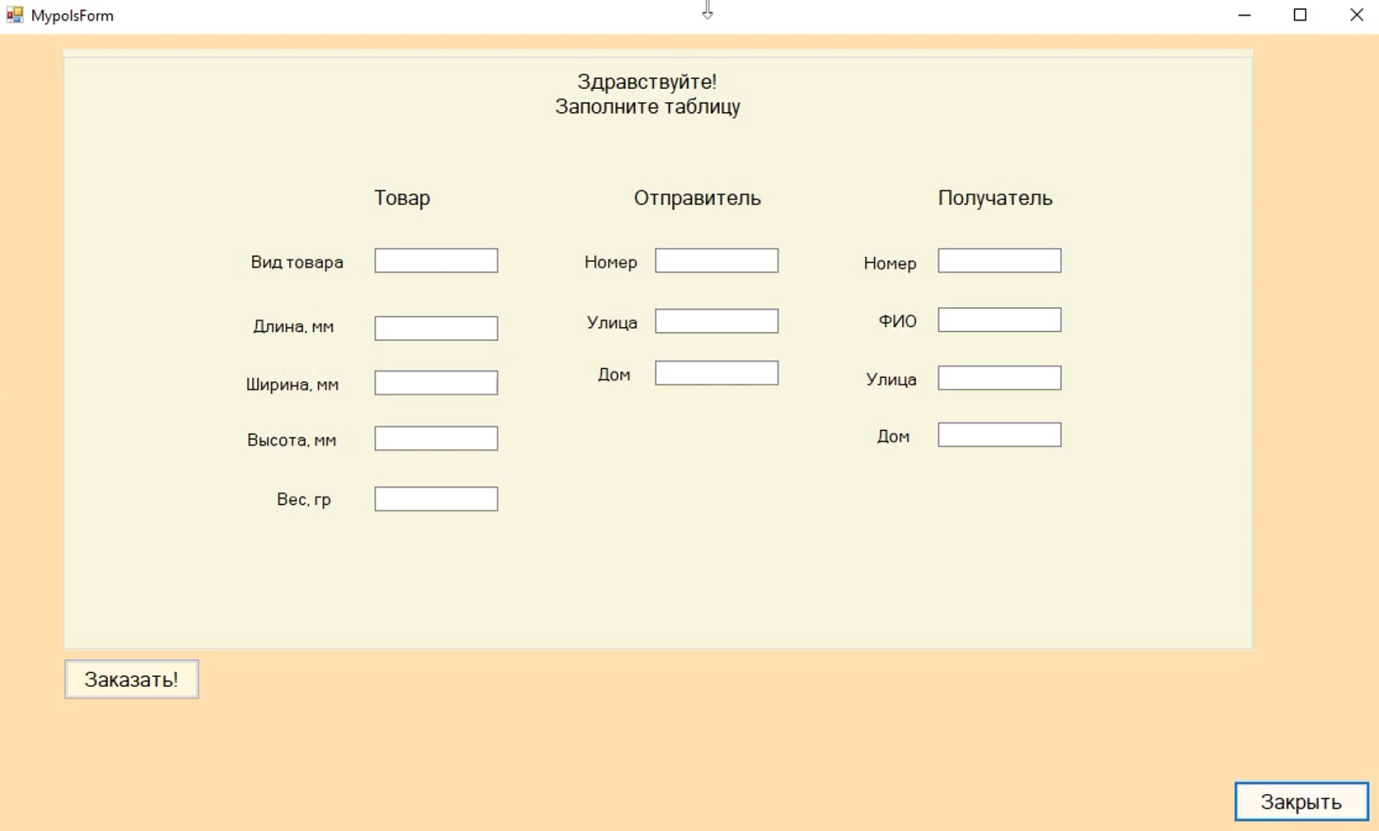


Все данные после заполнения сохраняются в базу SQL Servera.Если вы запустите снова программу, то вы без особого труда сможете войти под своей эл.почтой и паролем.

Окно запроса тем кем вы являетесь в данной программе

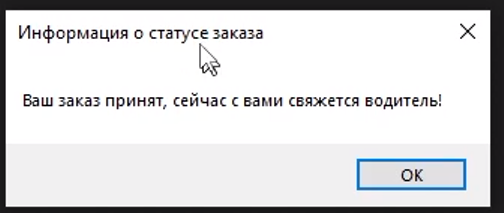


Окно пользователя

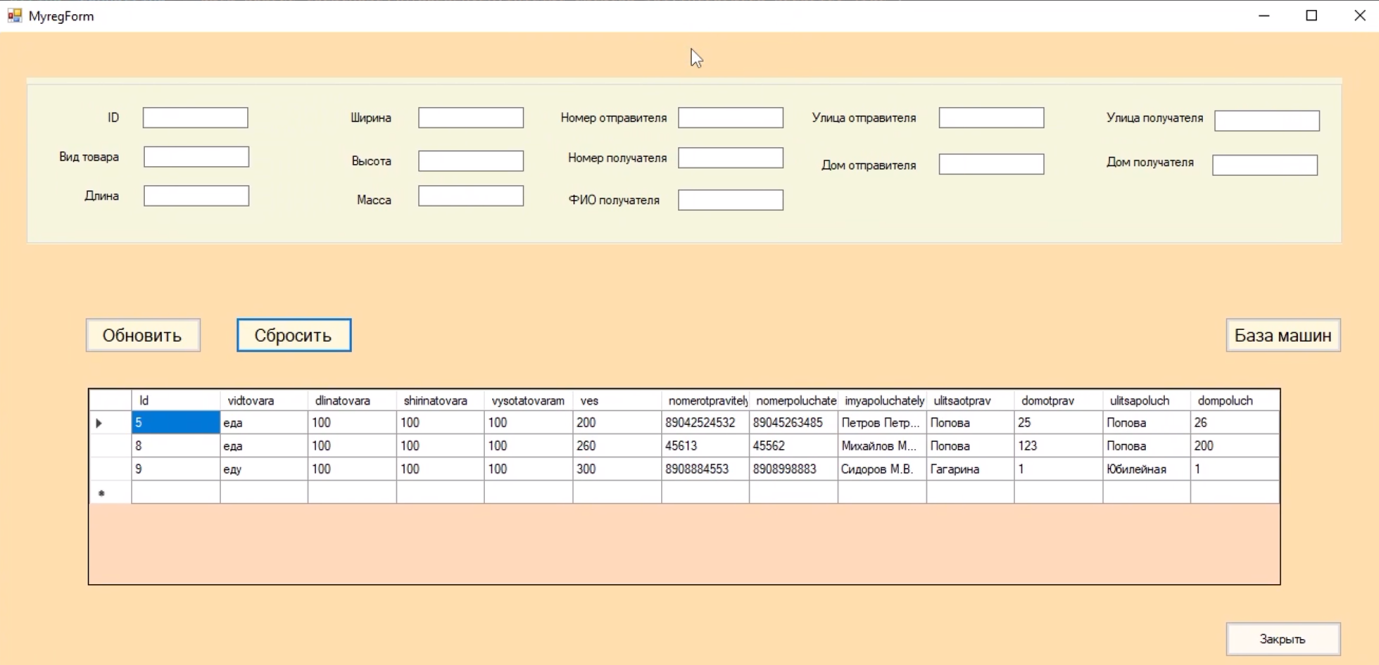
****

После того, как вы заполните данную таблицу, все данные сохраняются в базе данных.

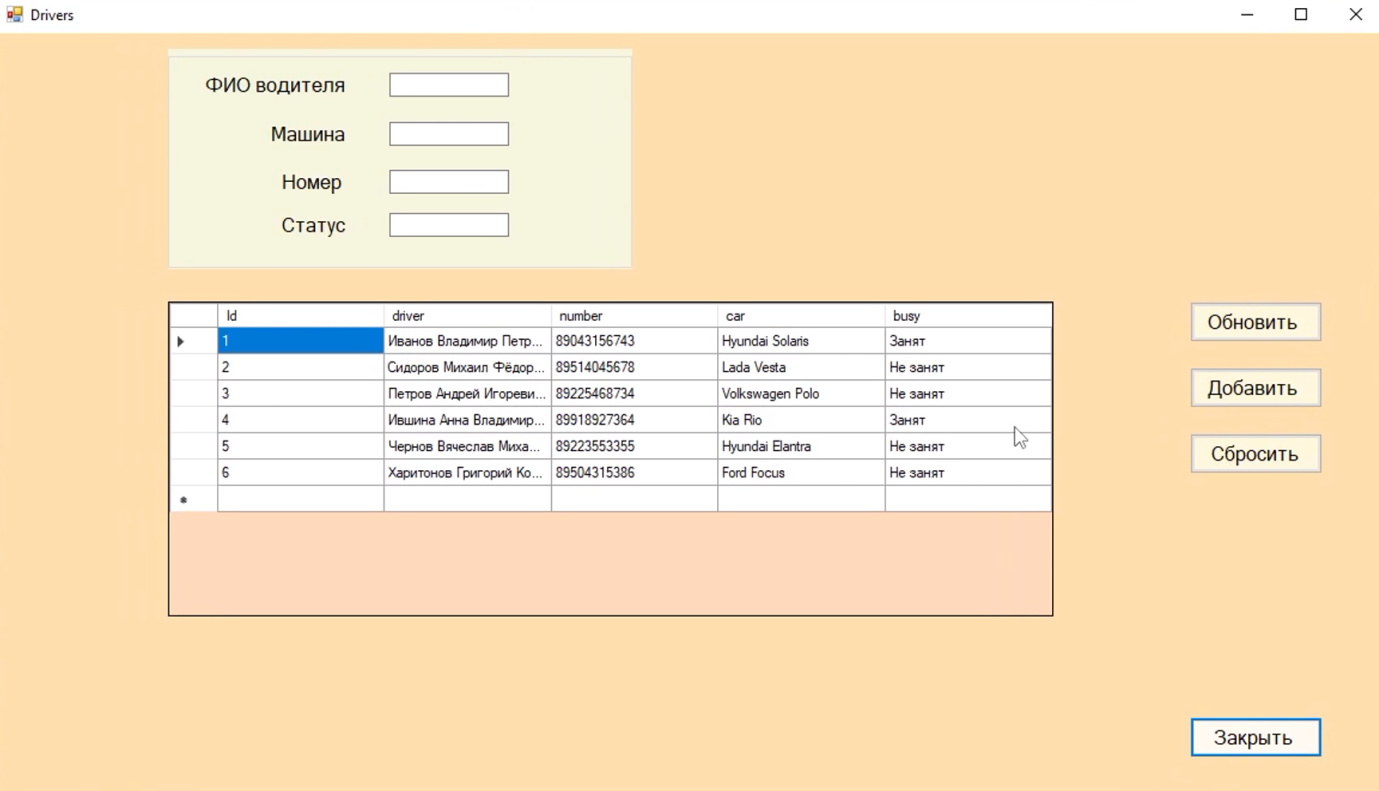
Информационное окно, которое подтверждает, что заказ прошёл успешно



Окно диспетчера с информационной таблицей о заказах



Окно “база машин” с информационной таблицей о водителях



**6 Ссылки YouTube**

1. [https://youtu.be/4NFuvdf55UM](https://vk.com/away.php?to=https%3A%2F%2Fyoutu.be%2F4NFuvdf55UM&cc_key=)- ссылка на видео.