

Министерство общего и профессионального
образования Российской Федерации
Пермский государственный технический университет

Творческая работа “ Финансовый калькулятор и задача коммивояжера”

Выполнила студентка группы РИС 23-36:

Федорова О.И.

Проверила

Доцент кафедры ИТАС

Полякова О.А.

Пермь 2024

Финансовый калькулятор

1. Постановка задачи для финансового калькулятора: создать калькулятор, который будет принимать на вход экономические данные и на их основе рассчитывать показатели.

2. Анализ задачи:

1) В конструкторе Windows Forms создается дизайн калькулятора.

2) В классе `public ref class Calculator : public`

`System::Windows::Forms::Form` автоматически, опираясь на конструктор создается база кнопок.

3) Функция, отвечающая за итоговый вывод ответов (“посчитать”) проверяет, какие показатели для расчета пользователь выбрал. На основе этого функция забирает данные из поля ввода, который ввел пользователь, преобразовывает их и по формулам рассчитывает ответ. Ответ записывается в поле для итогового вывода.

4) Функция, отвечающая за очистку полей (“очистить”) заполняет поле для итогового ответа нулями, а поля для ввода данных очищаются с помощью `clear()`.

3. UML диаграмма:

public ref class Calculator : public System::Windows::Forms::Form
<ul style="list-style-type: none">- System::Windows::Forms::Label^ resultGrossProfit- System::Windows::Forms::Label^ resultProfitFromSales- System::Windows::Forms::Label^ resultBalanceProfit- System::Windows::Forms::Label^ resultNetProfit- System::Windows::Forms::Label^ label4- System::Windows::Forms::Label^ label5- System::Windows::Forms::Label^ label6- System::Windows::Forms::Label^ label7- System::Windows::Forms::TextBox^ revenue- System::Windows::Forms::TextBox^ costPrice- System::Windows::Forms::TextBox^ businessExpenses- System::Windows::Forms::TextBox^ administrativeExpenses- System::Windows::Forms::TextBox^ otherIncome- System::Windows::Forms::TextBox^ otherExpenses- System::Windows::Forms::TextBox^ IncomeTax- System::Windows::Forms::CheckBox^ checkGrossProfit- System::Windows::Forms::CheckBox^ checkRevenueFromSales- System::Windows::Forms::CheckBox^ checkBookProfit- System::Windows::Forms::CheckBox^ checkNetProfit- System::Windows::Forms::Button^ calculationButton- System::Windows::Forms::Button^ clearButton- System::Windows::Forms::Label^ label3- System::Windows::Forms::Label^ label2- System::Windows::Forms::Label^ label1- System::ComponentModel::Container ^components- InitializeComponent(void): void- System::Void button_Click(System::Object^ sender, System::EventArgs^ e)- System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
<ul style="list-style-type: none">+ Calculator(void)# ~Calculator()

4. Код программы:

Файл Calculator.h:

```
#pragma once
```

```
#include <string>
```

```
namespace FinancialCalculator {
```

```
    using namespace System;
```

```
    using namespace System::ComponentModel;
```

```
    using namespace System::Collections;
```

```
    using namespace System::Windows::Forms;
```

```
    using namespace System::Data;
```

```
    using namespace System::Drawing;
```

```

/// <summary>
/// Сводка для Calculator
/// </summary>
public ref class Calculator : public System::Windows::Forms::Form
{
public:
    Calculator(void)
    {
        InitializeComponent();
        //
        //TODO: добавьте код конструктора
        //оапплгп
    }

protected:
    /// <summary>
    /// Освободить все используемые ресурсы.
    /// </summary>
    ~Calculator()
    {
        if (components)
        {
            delete components;
        }
    }

private: System::Windows::Forms::Label^ resultGrossProfit;
private: System::Windows::Forms::Label^ resultProfitFromSales;
private: System::Windows::Forms::Label^ resultBalanceProfit;
private: System::Windows::Forms::Label^ resultNetProfit;

```

```
private: System::Windows::Forms::Label^ label4;  
private: System::Windows::Forms::Label^ label5;  
private: System::Windows::Forms::Label^ label6;  
private: System::Windows::Forms::Label^ label7;  
protected:
```

```
private: System::Windows::Forms::TextBox^ revenue;  
private: System::Windows::Forms::TextBox^ costPrice;  
private: System::Windows::Forms::TextBox^ businessExpenses;  
private: System::Windows::Forms::TextBox^ administrativeExpenses;  
private: System::Windows::Forms::TextBox^ otherIncome;  
private: System::Windows::Forms::TextBox^ otherExpenses;  
private: System::Windows::Forms::TextBox^ IncomeTax;  
private: System::Windows::Forms::CheckBox^ checkGrossProfit;  
private: System::Windows::Forms::CheckBox^ checkRevenueFromSales;  
private: System::Windows::Forms::CheckBox^ checkBookProfit;  
private: System::Windows::Forms::CheckBox^ checkNetProfit;
```

```
private: System::Windows::Forms::Button^ calculationButton;  
private: System::Windows::Forms::Button^ clearButton;
```

```
private: System::Windows::Forms::Label^ label3;  
private: System::Windows::Forms::Label^ label2;  
private: System::Windows::Forms::Label^ label1;
```

```
protected:
```

```
private:
```

```
    /// <summary>
```

```
    /// Обязательная переменная конструктора.
```

```
    /// </summary>
```

```
    System::ComponentModel::Container ^components;
```

```
#pragma region Windows Form Designer generated code
```

```
/// <summary>
```

```
/// Требуемый метод для поддержки конструктора — не  
изменяйте
```

```
/// содержимое этого метода с помощью редактора кода.
```

```
/// </summary>
```

```
void InitializeComponent(void)
```

```
{
```

```
        this->resultGrossProfit = (gcnew  
System::Windows::Forms::Label());  
        this->resultProfitFromSales = (gcnew  
System::Windows::Forms::Label());  
        this->resultBalanceProfit = (gcnew  
System::Windows::Forms::Label());  
        this->resultNetProfit = (gcnew  
System::Windows::Forms::Label());  
        this->label4 = (gcnew System::Windows::Forms::Label());  
        this->label5 = (gcnew System::Windows::Forms::Label());  
        this->label6 = (gcnew System::Windows::Forms::Label());  
        this->label7 = (gcnew System::Windows::Forms::Label());  
        this->revenue = (gcnew System::Windows::Forms::TextBox());  
        this->costPrice = (gcnew  
System::Windows::Forms::TextBox());  
        this->businessExpenses = (gcnew  
System::Windows::Forms::TextBox());  
        this->administrativeExpenses = (gcnew  
System::Windows::Forms::TextBox());  
        this->otherIncome = (gcnew  
System::Windows::Forms::TextBox());
```

```

        this->otherExpenses = (gcnew
System::Windows::Forms::TextBox());
        this->IncomeTax = (gcnew
System::Windows::Forms::TextBox());
        this->checkGrossProfit = (gcnew
System::Windows::Forms::CheckBox());
        this->checkRevenueFromSales = (gcnew
System::Windows::Forms::CheckBox());
        this->checkBookProfit = (gcnew
System::Windows::Forms::CheckBox());
        this->checkNetProfit = (gcnew
System::Windows::Forms::CheckBox());
        this->calculationButton = (gcnew
System::Windows::Forms::Button());
        this->clearButton = (gcnew
System::Windows::Forms::Button());
        this->label3 = (gcnew System::Windows::Forms::Label());
        this->label2 = (gcnew System::Windows::Forms::Label());
        this->label1 = (gcnew System::Windows::Forms::Label());
        this->SuspendLayout();
        //
        // resultGrossProfit
        //
        this->resultGrossProfit->Font = (gcnew
System::Drawing::Font(L"Arial Narrow", 25.8F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->resultGrossProfit->Location =
System::Drawing::Point(571, 23);
        this->resultGrossProfit->Name = L"resultGrossProfit";

```



```

        this->resultGrossProfit->Size = System::Drawing::Size(336,
48);

        this->resultGrossProfit->TabIndex = 25;
        this->resultGrossProfit->Text = L"0";
        this->resultGrossProfit->TextAlign =
System::Drawing::ContentAlignment::BottomRight;
        //
        // resultProfitFromSales
        //
        this->resultProfitFromSales->Font = (gcnew
System::Drawing::Font(L"Arial Narrow", 25.8F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->resultProfitFromSales->Location =
System::Drawing::Point(577, 93);
        this->resultProfitFromSales->Name =
L"resultProfitFromSales";
        this->resultProfitFromSales->Size =
System::Drawing::Size(330, 50);
        this->resultProfitFromSales->TabIndex = 26;
        this->resultProfitFromSales->Text = L"0";
        this->resultProfitFromSales->TextAlign =
System::Drawing::ContentAlignment::BottomRight;
        //
        // resultBalanceProfit
        //
        this->resultBalanceProfit->Font = (gcnew
System::Drawing::Font(L"Arial Narrow", 25.8F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));

```

```

        this->resultBalanceProfit->Location =
System::Drawing::Point(577, 166);
        this->resultBalanceProfit->Name = L"resultBalanceProfit";
        this->resultBalanceProfit->Size = System::Drawing::Size(330,
50);

        this->resultBalanceProfit->TabIndex = 27;
        this->resultBalanceProfit->Text = L"0";
        this->resultBalanceProfit->TextAlign =
System::Drawing::ContentAlignment::BottomRight;
        //
        // resultNetProfit
        //
        this->resultNetProfit->Font = (gcnew
System::Drawing::Font(L"Arial Narrow", 25.8F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->resultNetProfit->Location = System::Drawing::Point(575,
235);

        this->resultNetProfit->Name = L"resultNetProfit";
        this->resultNetProfit->Size = System::Drawing::Size(332, 50);
        this->resultNetProfit->TabIndex = 28;
        this->resultNetProfit->Text = L"0";
        this->resultNetProfit->TextAlign =
System::Drawing::ContentAlignment::BottomRight;
        //
        // label4
        //
        this->label4->AutoSize = true;
        this->label4->BackColor = System::Drawing::Color::LightPink;

```

```

        this->label4->BorderStyle =
System::Windows::Forms::BorderStyle::FixedSingle;
        this->label4->Font = (gcnew System::Drawing::Font(L"Arial",
16.2F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->label4->Location = System::Drawing::Point(445, 380);
        this->label4->Name = L"label4";
        this->label4->Size = System::Drawing::Size(341, 34);
        this->label4->TabIndex = 39;
        this->label4->Text = L"Управленческие расходы";
//
// label5
//
        this->label5->AutoSize = true;
        this->label5->BackColor = System::Drawing::Color::LightPink;
        this->label5->BorderStyle =
System::Windows::Forms::BorderStyle::FixedSingle;
        this->label5->Font = (gcnew System::Drawing::Font(L"Arial",
16.2F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->label5->Location = System::Drawing::Point(502, 495);
        this->label5->Name = L"label5";
        this->label5->Size = System::Drawing::Size(214, 34);
        this->label5->TabIndex = 40;
        this->label5->Text = L"Прочие доходы";
//
// label6
//

```

```

        this->label6->AutoSize = true;
        this->label6->BackColor = System::Drawing::Color::LightPink;
        this->label6->BorderStyle =
System::Windows::Forms::BorderStyle::FixedSingle;
        this->label6->Font = (gcnew System::Drawing::Font(L"Arial",
16.2F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->label6->Location = System::Drawing::Point(502, 608);
        this->label6->Name = L"label6";
        this->label6->Size = System::Drawing::Size(228, 34);
        this->label6->TabIndex = 41;
        this->label6->Text = L"Прочие расходы";
//
// label7
//
        this->label7->AutoSize = true;
        this->label7->BackColor = System::Drawing::Color::LightPink;
        this->label7->BorderStyle =
System::Windows::Forms::BorderStyle::FixedSingle;
        this->label7->Font = (gcnew System::Drawing::Font(L"Arial",
16.2F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->label7->Location = System::Drawing::Point(867, 382);
        this->label7->Name = L"label7";
        this->label7->Size = System::Drawing::Size(251, 34);
        this->label7->TabIndex = 42;
        this->label7->Text = L"Налог на прибыль";
//

```

```

// revenue
//
this->revenue->Location = System::Drawing::Point(87, 417);
this->revenue->Multiline = true;
this->revenue->Name = L"revenue";
this->revenue->Size = System::Drawing::Size(129, 34);
this->revenue->TabIndex = 43;
//
// costPrice
//
this->costPrice->Location = System::Drawing::Point(58, 530);
this->costPrice->Multiline = true;
this->costPrice->Name = L"costPrice";
this->costPrice->Size = System::Drawing::Size(217, 34);
this->costPrice->TabIndex = 44;
//
// businessExpenses
//
this->businessExpenses->Location = System::Drawing::Point(31,
643);

this->businessExpenses->Multiline = true;
this->businessExpenses->Name = L"businessExpenses";
this->businessExpenses->Size = System::Drawing::Size(320,
34);

this->businessExpenses->TabIndex = 45;
//
// administrativeExpenses
//
this->administrativeExpenses->Location =
System::Drawing::Point(444, 417);

```

```

        this->administrativeExpenses->Multiline = true;
        this->administrativeExpenses->Name =
L"administrativeExpenses";
        this->administrativeExpenses->Size =
System::Drawing::Size(353, 34);
        this->administrativeExpenses->TabIndex = 46;
        //
        // otherIncome
        //
        this->otherIncome->Location = System::Drawing::Point(502,
530);

        this->otherIncome->Multiline = true;
        this->otherIncome->Name = L"otherIncome";
        this->otherIncome->Size = System::Drawing::Size(237, 36);
        this->otherIncome->TabIndex = 47;
        //
        // otherExpenses
        //
        this->otherExpenses->Location = System::Drawing::Point(502,
643);

        this->otherExpenses->Multiline = true;
        this->otherExpenses->Name = L"otherExpenses";
        this->otherExpenses->Size = System::Drawing::Size(237, 36);
        this->otherExpenses->TabIndex = 48;
        //
        // IncomeTax
        //
        this->IncomeTax->Location = System::Drawing::Point(867,
417);

        this->IncomeTax->Multiline = true;

```

```

this->IncomeTax->Name = L"IncomeTax";
this->IncomeTax->Size = System::Drawing::Size(261, 34);
this->IncomeTax->TabIndex = 49;
//
// checkGrossProfit
//
this->checkGrossProfit->AutoSize = true;
this->checkGrossProfit->Font = (gcnew
System::Drawing::Font(L"Arial", 19.8F, System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
this->checkGrossProfit->Location =
System::Drawing::Point(77, 38);
this->checkGrossProfit->Name = L"checkGrossProfit";
this->checkGrossProfit->Size = System::Drawing::Size(336,
42);

this->checkGrossProfit->TabIndex = 50;
this->checkGrossProfit->Text = L"Валовая прибыль";
this->checkGrossProfit->UseVisualStyleBackColor = true;
//
// checkRevenueFromSales
//
this->checkRevenueFromSales->AutoSize = true;
this->checkRevenueFromSales->Font = (gcnew
System::Drawing::Font(L"Arial", 19.8F, System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
this->checkRevenueFromSales->Location =
System::Drawing::Point(77, 99);

```

```

        this->checkRevenueFromSales->Name =
L"checkRevenueFromSales";
        this->checkRevenueFromSales->Size =
System::Drawing::Size(366, 42);
        this->checkRevenueFromSales->TabIndex = 51;
        this->checkRevenueFromSales->Text = L"Прибыль от
продаж";
        this->checkRevenueFromSales->UseVisualStyleBackColor =
true;

        //
        // checkBookProfit
        //
        this->checkBookProfit->AutoSize = true;
        this->checkBookProfit->Font = (gcnew
System::Drawing::Font(L"Arial", 19.8F, System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->checkBookProfit->Location =
System::Drawing::Point(77, 166);
        this->checkBookProfit->Name = L"checkBookProfit";
        this->checkBookProfit->Size = System::Drawing::Size(393,
42);

        this->checkBookProfit->TabIndex = 52;
        this->checkBookProfit->Text = L"Балансовая прибыль";
        this->checkBookProfit->UseVisualStyleBackColor = true;
        //
        // checkNetProfit
        //
        this->checkNetProfit->AutoSize = true;

```



```

        this->checkNetProfit->Font = (gcnew
System::Drawing::Font(L"Arial", 19.8F, System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->checkNetProfit->Location = System::Drawing::Point(77,
235);

        this->checkNetProfit->Name = L"checkNetProfit";
        this->checkNetProfit->Size = System::Drawing::Size(310, 42);
        this->checkNetProfit->TabIndex = 53;
        this->checkNetProfit->Text = L"Чистая прибыль";
        this->checkNetProfit->UseVisualStyleBackColor = true;
        //
        // calculationButton
        //
        this->calculationButton->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Zoom;
        this->calculationButton->FlatAppearance->BorderColor =
System::Drawing::Color::Black;
        this->calculationButton->FlatAppearance-
>MouseDownBackColor = System::Drawing::Color::Black;
        this->calculationButton->Font = (gcnew
System::Drawing::Font(L"Arial Narrow", 16.2F,
static_cast<System::Drawing::FontStyle>((System::Drawing::FontStyle::Bold |
System::Drawing::FontStyle::Italic)),
        System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
        this->calculationButton->ForeColor =
System::Drawing::SystemColors::ActiveCaptionText;
        this->calculationButton->Location =
System::Drawing::Point(895, 507);

```

```

this->calculationButton->Name = L"calculationButton";
this->calculationButton->Size = System::Drawing::Size(193,
59);

this->calculationButton->TabIndex = 54;
this->calculationButton->Text = L"Посчитать";
this->calculationButton->UseVisualStyleBackColor = true;
this->calculationButton->Click += gcnew
System::EventHandler(this, &Calculator::button_Click);

//
// clearButton
//

this->clearButton->Font = (gcnew
System::Drawing::Font(L"Arial Narrow", 16.2F,
static_cast<System::Drawing::FontStyle>((System::Drawing::FontStyle::Bold |
System::Drawing::FontStyle::Italic)),
        System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));

this->clearButton->Location = System::Drawing::Point(895,
620);

this->clearButton->Name = L"clearButton";
this->clearButton->Size = System::Drawing::Size(193, 59);
this->clearButton->TabIndex = 55;
this->clearButton->Text = L"Очистить";
this->clearButton->UseVisualStyleBackColor = true;
this->clearButton->Click += gcnew System::EventHandler(this,
&Calculator::button2_Click);

//
// label3
//

this->label3->AutoSize = true;

```

```

        this->label3->BackColor = System::Drawing::Color::LightPink;
        this->label3->BorderStyle =
System::Windows::Forms::BorderStyle::FixedSingle;
        this->label3->Cursor =
System::Windows::Forms::Cursors::Hand;
        this->label3->Font = (gcnew System::Drawing::Font(L"Arial",
16.2F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->label3->Location = System::Drawing::Point(35, 608);
        this->label3->Name = L"label3";
        this->label3->Size = System::Drawing::Size(318, 34);
        this->label3->TabIndex = 38;
        this->label3->Text = L"Коммерческие расходы";
//
// label2
//
        this->label2->AutoSize = true;
        this->label2->BackColor = System::Drawing::Color::LightPink;
        this->label2->BorderStyle =
System::Windows::Forms::BorderStyle::FixedSingle;
        this->label2->Font = (gcnew System::Drawing::Font(L"Arial",
16.2F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->label2->Location = System::Drawing::Point(62, 495);
        this->label2->Name = L"label2";
        this->label2->Size = System::Drawing::Size(215, 34);
        this->label2->TabIndex = 37;
        this->label2->Text = L"Себестоимость";

```

```

//
// label1
//
this->label1->AutoSize = true;
this->label1->BackColor = System::Drawing::Color::LightPink;
this->label1->BorderStyle =
System::Windows::Forms::BorderStyle::FixedSingle;
    this->label1->Font = (gcnew System::Drawing::Font(L"Arial",
16.2F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
this->label1->Location = System::Drawing::Point(87, 382);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(127, 34);
this->label1->TabIndex = 36;
this->label1->Text = L"Выручка";
//
// Calculator
//
this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);
this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
    this->BackColor = System::Drawing::Color::MistyRose;
this->ClientSize = System::Drawing::Size(1156, 707);
this->Controls->Add(this->clearButton);
this->Controls->Add(this->calculationButton);
this->Controls->Add(this->checkNetProfit);
this->Controls->Add(this->checkBookProfit);
this->Controls->Add(this->checkRevenueFromSales);
this->Controls->Add(this->checkGrossProfit);

```

```

this->Controls->Add(this->IncomeTax);
this->Controls->Add(this->otherExpenses);
this->Controls->Add(this->otherIncome);
this->Controls->Add(this->administrativeExpenses);
this->Controls->Add(this->businessExpenses);
this->Controls->Add(this->costPrice);
this->Controls->Add(this->revenue);
this->Controls->Add(this->label7);
this->Controls->Add(this->label6);
this->Controls->Add(this->label5);
this->Controls->Add(this->label4);
this->Controls->Add(this->label3);
this->Controls->Add(this->label2);
this->Controls->Add(this->label1);
this->Controls->Add(this->resultNetProfit);
this->Controls->Add(this->resultBalanceProfit);
this->Controls->Add(this->resultProfitFromSales);
this->Controls->Add(this->resultGrossProfit);
this->Name = L"Calculator";
this->Text = L"Calculator";
this->ResumeLayout(false);
this->PerformLayout();

```

```

}

```

```

#pragma endregion

```

```

private: System::Void button_Click(System::Object^ sender, System::EventArgs^

```

```

e) { // обработка события нажатия на кнопку счета

```

```

    if (checkGrossProfit->Checked) { // если нужно рассчитать валовую
прибыль

```

```

        double value1; // переменная для получения данных

```

```
        value1 = Convert::ToDouble(revenue->Text); // получения данных
из поля ввода и их преобразование
        double value2; // переменная для получения данных
        value2 = Convert::ToDouble(costPrice->Text); // получения данных
из поля ввода и их преобразование
        this->resultGrossProfit->Text = Convert::ToString(value1 - value2); //
запись ответа в поле итогового ответа
    }
    if (checkRevenueFromSales->Checked) { // если нужно рассчитать
прибыль от продаж
        double value3; // переменная для получения данных
        value3 = Convert::ToDouble(revenue->Text); // получения данных
из поля ввода и их преобразование
        double value4; // переменная для получения данных
        value4 = Convert::ToDouble(costPrice->Text); // получения данных
из поля ввода и их преобразование
        double value5; // переменная для получения данных
        value5 = Convert::ToDouble(businessExpenses->Text); // получения
данных из поля ввода и их преобразование
        double value6; // переменная для получения данных
        value6 = Convert::ToDouble(administrativeExpenses->Text); //
получения данных из поля ввода и их преобразование
        this->resultProfitFromSales->Text = Convert::ToString(value3 -
value4 - value5 - value6); // запись ответа в поле итогового ответа
    }
    if (checkBookProfit->Checked) { // если нужно рассчитать балансовую
прибыль
        double value7; // переменная для получения данных
        value7 = Convert::ToDouble(revenue->Text); // получения данных
из поля ввода и их преобразование
```

```

double value8; // переменная для получения данных
value8 = Convert::ToDouble(costPrice->Text); // получения данных
из поля ввода и их преобразование

double value9; // переменная для получения данных
value9 = Convert::ToDouble(bisnessExpenses->Text); // получения
данных из поля ввода и их преобразование

double value10; // переменная для получения данных
value10 = Convert::ToDouble(administrativeExpenses->Text); //
получения данных из поля ввода и их преобразование

double value11; // переменная для получения данных
value11 = Convert::ToDouble(otherIncome->Text); // получения
данных из поля ввода и их преобразование

double value12; // переменная для получения данных
value12 = Convert::ToDouble(otherExpenses->Text);
this->resultBalanceProfit->Text = Convert::ToString(value7 - value8 -
value9 - value10 + value11 - value12); // запись ответа в поле итогового ответа
}

if (checkNetProfit->Checked) { // если нужно рассчитать чистую
прибыль

double value13; // переменная для получения данных
value13 = Convert::ToDouble(revenue->Text); // получения данных
из поля ввода и их преобразование

double value14; // переменная для получения данных
value14 = Convert::ToDouble(costPrice->Text); // получения
данных из поля ввода и их преобразование

double value15; // переменная для получения данных
value15 = Convert::ToDouble(bisnessExpenses->Text); // получения
данных из поля ввода и их преобразование

double value16; // переменная для получения данных

```

```

        value16 = Convert::ToDouble(administrativeExpenses->Text); //
получения данных из поля ввода и их преобразование
        double value17; // переменная для получения данных
        value17 = Convert::ToDouble(otherIncome->Text); // получения
данных из поля ввода и их преобразование
        double value18; // переменная для получения данных
        value18 = Convert::ToDouble(otherExpenses->Text); // получения
данных из поля ввода и их преобразование
        double value19; // переменная для получения данных
        value19 = Convert::ToDouble(IncomeTax->Text); // получения
данных из поля ввода и их преобразование
        this->resultNetProfit->Text = Convert::ToString(value13 - value14 -
value15 - value16 + value17 - value18 - value19); // запись ответа в поле
итогового ответа
    }
}
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^
e) { // обработка события нажатия на кнопку очищения полей
    resultGrossProfit->Text = "0"; // установление в поле итогового ответа
значения по умолчанию(0)
    resultProfitFromSales->Text = "0"; // установление в поле итогового
ответа значения по умолчанию(0)
    resultBalanceProfit->Text = "0"; // установление в поле итогового ответа
значения по умолчанию(0)
    resultNetProfit->Text = "0"; // установление в поле итогового ответа
значения по умолчанию(0)
    revenue->Clear(); // очищение поля для ввода данных
    costPrice->Clear(); // очищение поля для ввода данных
    bisnessExpenses->Clear(); // очищение поля для ввода данных
    administrativeExpenses->Clear(); // очищение поля для ввода данных

```



```
otherIncome->Clear(); // очищение поля для ввода данных
otherExpenses->Clear(); // очищение поля для ввода данных
IncomeTax->Clear(); // очищение поля для ввода данных
}
};
}
```

Файл Calculator.cpp:

```
#include "Calculator.h"

using namespace System;
using namespace System::Windows::Forms;

[STAThread]
void main(array<String^>^ arg) {
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);

    FinancialCalculator::Calculator form;
    Application::Run(% form);
}
```

5. Результат работы программы:

Calculator

☐ Валовая прибыль 0

☐ Прибыль от продаж 0

☐ Балансовая прибыль 0

☐ Чистая прибыль 0

Выручка
|

Управленческие расходы
|

Налог на прибыль
|

Себестоимость
|

Прочие доходы
|

Посчитать

Коммерческие расходы
|

Прочие расходы
|

Очистить

Calculator

☒ Валовая прибыль 234608677

☒ Прибыль от продаж 132574349

☒ Балансовая прибыль 126681069

☒ Чистая прибыль 105638871

Выручка
849688545

Управленческие расходы
8341322

Налог на прибыль
21042198

Себестоимость
615079868

Прочие доходы
8618830

Посчитать

Коммерческие расходы
93693006

Прочие расходы
14512110

Очистить

Задача коммивояжера

1. Постановка задачи: Существует N городов, соединенных путями по принципу «каждый с каждым». Все пути имеют вес, расстояние между городами. Задача коммивояжера состоит в том, чтобы объехать все города, побывав в каждом лишь один раз, при этом требуется, чтобы сумма расстояний между городами была минимальной.

2. Анализ задачи:

1) Глобальные переменные: `maxSize`, `width` и `height` определяются в начале файла. `maxSize` задает максимальное количество вершин в графе, а `width` и `height` определяют размеры окна программы.

2) Структура `vertCoord`: Используется для хранения координат вершин графа. Содержит два поля `x` и `y`.

3) Класс `Graph` : Представляет граф и содержит множество методов для работы с ним. Некоторые из них:

- `checkEmpty()` и `checkFull()`: Проверяют, пуст ли граф и полон ли он соответственно.

- `getElementFromAdjMatrix()`: Получает элемент матрицы смежности.

- `amountVertices()`: Возвращает количество вершин в графе.

- `vertexPosition()`: Получает позицию вершины.

- `insertVertex()`: Добавляет вершину к графу.

- `insertEdge()`: Вставляет новое ребро в граф.

- `print()`: Выводит матрицу смежности на экран.

- `eraseVertex()`: Удаляет вершину из графа.

- `eraseEdge()`: Удаляет ребро из графа.

- `drawGraph()`: Отображает граф на экране.

4) Функции для решения задачи коммивояжера: Эти функции не представлены в данном фрагменте кода, но их названия говорят о том, что они выполняют различные операции для решения данной задачи.

5) Функции для отрисовки интерфейса: `drawButton1` до `drawButton6` отвечают за отрисовку кнопок в интерфейсе пользователя.

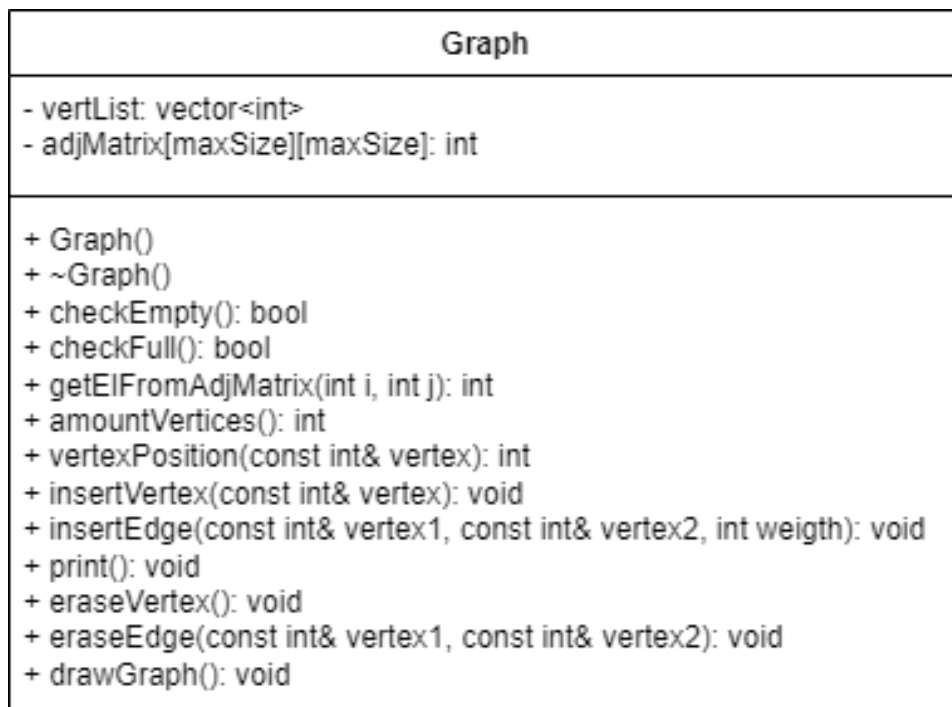
6) Функции для отрисовки графа: drawCircle, drawText, drawLine, drawVertex и createGraph используются для визуализации графа на экране.

7) Обработчики событий: mouseMove, mouseClicked и hoverCheckButton обрабатывают события мыши и кнопок.

8) Функция reshape: Вызывается при изменении размера окна и обновляет размеры экрана.

9) Функция display: Основной цикл отрисовки, который вызывается каждый раз, когда требуется перерисовать экран.

3. UML диаграмма:



4. Код программы:

Файл Graph.h:

```
#pragma once // для предотвращения повторной обработки файлов
#include <gl/glut.h> // подключение заголовочного файла библиотеки OpenGL
#include <iostream> // подключение заголовочного файла, который позволяет
использовать потоковый ввод-вывод
#include <vector> // подключение библиотеки для использования вектора
```

```
#include <string> // подключение библиотеки для использования строк
using namespace std; // разрешает использование пространства имен
стандартной библиотеки
```

```
const int maxSize = 20; // максимальное количество вершин в графе
extern int width; // ширина окна
extern int height; // высота окна
struct vertCoord { // структура для объектов с данными - координатами
    int x, y;
};
```

```
class Graph {
private:
    vector<int> vertList; // вектор вершин
    int adjMatrix[maxSize][maxSize]; // матрица смежности
public:
    Graph(); // конструктор
    ~Graph(); // деструктор
    bool checkEmpty(); // функция проверки, пуст ли граф
    bool checkFull(); // функция проверки, полон ли граф
    int getElFromAdjMatrix(int i, int j) { return adjMatrix[i][j]; } // получение
элемента матрицы смежности
    int amountVertices() { return vertList.size(); } // получение числа вершин
    int vertexPosition(const int& vertex); // получение позиции вершины
    void insertVertex(const int& vertex); // добавление вершины к графу
    void insertEdge(const int& vertex1, const int& vertex2, int weigth); //
вставка нового ребра
    void print(); // функция печати матрицы смежности
    void eraseVertex(); // функция удаления вершины
```

```
void eraseEdge(const int& vertex1, const int& vertex2); // функция  
удаления ребра
```

```
void drawGraph(); // функция для отображения графа  
};
```

```
bool checkData(int** matrix); // функция проверяет, есть ли данные для  
решения задачи коммивояжера
```

```
int** modificaMatrix(); // получение измененной матрицы
```

```
int* searchMinEl(int* line, int n); // поиск минимального элемента
```

```
int** reductMatrix(int** oldmatrix); // функция редуцирует матрицу смежности
```

```
int** findZero(int** oldmatrix); // функция возвращает измененную матрицу
```

```
void printAnswer(); // функция печатает результат работы программы
```

```
void drawButton1(); // отрисовка 1 кнопки
```

```
void drawButton2(); // отрисовка 2 кнопки
```

```
void drawButton3(); // отрисовка 3 кнопки
```

```
void drawButton4(); // отрисовка 4 кнопки
```

```
void drawButton5(); // отрисовка 5 кнопки
```

```
void drawButton6(); // отрисовка 6 кнопки
```

```
void drawCircle(int x, int y, int radius); // отрисовка окружности для вершины
```

```
void drawText(int num, int x1, int y1); // отображение текста
```

```
void drawLine(int text, int x0, int y0, int x1, int y1); // отрисовка линий(ребер)
```

```
void drawVertex(int n); // функция для отрисовки вершин графа
```

```
void createGraph(); // функция для создания графа
```

```
void coords(int i, int n); // функция для установки координат вершин графа
```

```
void hoverCheckButton(int x, int y); // проверка, курсор находится на кнопке  
или нет
```

```
void mouseMove(int x, int y); // перемещение мыши по экрану
```

```
void mouseClicked(int button, int state, int x, int y); // действия при нажатии на кнопки
```

```
void reshape(int w, int h); // ресайз окна OpenGL
```

```
void display(); // основной цикл отрисовки
```

Файл Graph.cpp:

```
#include "Graph.h" // подключение файла с определением структуры данных
using namespace std; // разрешает использование пространства имен
стандартной библиотеки
```

```
Graph::Graph() { // конструктор
```

```
    for (int i = 0; i < maxSize; i++) { // цикл для перебора ребер
```

```
        for (int j = 0; j < maxSize; j++) { // цикл для перебора ребер
```

```
            adjMatrix[i][j] = 0; // по умолчанию ребра заполняются 0
```

```
        }
```

```
    }
```

```
}
```

```
Graph::~~Graph() // деструктор
```

```
{ }
```

```
int Graph::vertexPosition(const int& vertex) { // нахождение позиции вершины в
векторе вершин
```

```
    for (size_t i = 0; i < vertList.size(); i++) { // перебор всех вершин
```

```
        if (vertList[i] == vertex) // поиск индекса нужной вершины в
```

```
векторе
```

```
            return i;
```

```
    }
```

```
    return -1;
```

```
}
```

```

bool Graph::checkEmpty() { // функция проверки, пуст ли граф
    if (vertList.size() != 0) // если элементов в векторе не 0
        return false;
    else
        return true;
}

```

```

bool Graph::checkFull() { // функция проверки, полон ли граф
    return (vertList.size() == maxSize); // если элементов в векторе не больше
    чем максимальный размер
}

```

```

void Graph::insertVertex(const int& vertex) { // добавление вершины к графу
    if (!checkFull()) // проверка, полон ли граф
        vertList.push_back(vertex); // добавление вершины в вектор
    else {
        cout << "Достигнуто максимальное количество вершин! " << endl
        << endl;
        return;
    }
}

```

```

void Graph::insertEdge(const int& vertex1, const int& vertex2, int weight) { //
вставка нового ребра
    if (weight < 1) { // проверка веса
        cout << "Вес не может быть дробным или отрицательным
значением!" << endl;
        return;
    }
}

```



```

        if (vertexPosition(vertex1) != (-1) && vertexPosition(vertex2) != (-1)) { //
проверка существуют ли заданные вершины
            int index = vertexPosition(vertex1); // индекс первой заданной
вершины
            int index2 = vertexPosition(vertex2); // индекс второй заданной
вершины
            if (adjMatrix[index][index2] != 0 && adjMatrix[index2][index] != 0)
{ // проверка, есть ли уже такое ребро
                cout << "Такое ребро уже существует!" << endl;
                return;
            }
            else {
                adjMatrix[index][index2] = weight; // установка веса ребра
                adjMatrix[index2][index] = weight; // установка веса ребра
            }
        }
        else {
            cout << "Заданных вершин не существует! " << endl;
            return;
        }
    }
}

```

```

void Graph::print() { // функция печати матрицы смежности
    if (!checkEmpty()) { // проверка, не пуст ли граф
        cout << "Матрица смежности: " << endl;
        cout << " ";
        for (int i = 0; i < vertList.size(); ++i) { // обозначение вершин
            cout << vertList[i] << " ";
        }
        cout << endl;
    }
}

```

```

        for (int i = 0; i < vertList.size(); ++i) { // обозначение вершин
            cout << vertList[i] << " ";
            for (int j = 0; j < vertList.size(); ++j) { // перебор для вывода
                cout << adjMatrix[i][j] << " ";
            }
            cout << endl << endl;
        }
    }
    else
        cout << "Граф пуст!" << endl << endl;
}

```

```

void Graph::eraseVertex() { // функция удаления ребра
    if (checkEmpty()) { // проверка на пустоту графа
        cout << "Граф пуст!" << endl;
        return;
    }
    int n = vertList.size(); // получение размера списка вершин
    for (int j = 0; j < n; j++) { // удаление ребер, ведущих к удаляемой
        adjMatrix[n - 1][j] = 0; // обнуление элемента матрицы смежности
        adjMatrix[j][n - 1] = 0; // обнуление элемента матрицы смежности
    }
    vertList.pop_back(); // удаление вершины из списка вершин
}

```

```

void Graph::eraseEdge(const int& vertex1, const int& vertex2) { // функция
удаления ребра
    if (vertexPosition(vertex1) != (-1) && vertexPosition(vertex2) != (-1)) { //
проверка, существуют ли обе вершины в графе
        int position1 = vertexPosition(vertex1); // индекс первой вершины
        int position2 = vertexPosition(vertex2); // индекс первой вершины
        if (adjMatrix[position1][position2] == 0) { // проверка, уществует
ли ребро между вершинами
            cout << "Такого ребра не существует!" << endl;
            return;
        }
        else {
            adjMatrix[position1][position2] = 0; // обновление данных
            adjMatrix[position2][position1] = 0; // обновление данных
        }
    }
    else { // если заданных вершин не существует
        cout << "Такой вершины(вершин) не существует! " << endl;
        return;
    }
}

```

Файл DrawNTSPFunctions.cpp:

```

#include "Graph.h" // подключение файла с определением структуры данных
using namespace std; // разрешает использование пространства имен
стандартной библиотеки

int width = 1080; // ширина
int height = 780; // высота
int radius; // радиус окружностей

```

```
int buttonBacklight; // переменная, которая будет содержать номер кнопки, на
которую наведен курсор
```

```
vertCoord coordinates[maxSize + 2]; // вектор координат
```

```
Graph graph; // объявление объекта класса
```

```
vector<pair<int, int>> indexes; // вектор пар для хранения индексов
```

```
vector<int> newIndexes; // правильная последовательность индексов для
решения задачи коммивояжера
```

```
void Graph::drawGraph() { // функция для отображения графа
```

```
    int n = graph.amountVertices(); // получение количества вершин в графе
```

```
    for (int i = 0; i < n; i++) { // цикл для установки координат для всех
```

```
вершин
```

```
        coords(i, n); // установка координат для вершины с индексом i
```

```
    }
```

```
    for (int i = 0; i < n; i++) { // основной цикл для рисования рёбер
```

```
        for (int j = 0; j < n; j++) { // подцикл для проверки наличия рёбер
```

```
между вершинами
```

```
            int a = adjMatrix[i][j]; // получение веса ребра между
```

```
вершинами i и j
```

```
            if (a != 0) { // если ребро существует
```

```
                drawLine(a, coordinates[j].x, coordinates[j].y,
```

```
coordinates[i].x, coordinates[i].y); // рисование линии
```

```
            }
```

```
        }
```

```
    }
```

```
    drawVertex(n); // рисование вершин
```

```
    glutPostRedisplay(); // обновление отображения
```

```
}
```

```
int** modificaMatrix() { // получение измененной матрицы
    int n = graph.amountVertices(); // получение количества вершин в графе
    int** matrix = new int* [n]; // создание новой матрицы размером n x n
    for (int i = 0; i < n; i++)
        matrix[i] = new int[n];
    for (int i = 0; i < n; i++) { // заполнение матрицы элементами из
        исходной матрицы смежности
            for (int j = 0; j < n; j++) {
                int elem = graph.getElFromAdjMatrix(i, j); // получение
                значения элемента исходной матрицы смежности
                if (elem == 0 || i == j) // если элемент равен нулю или i равно
                j
                    matrix[i][j] = -1;
                else // в противном случае сохраняется значение элемента
                    matrix[i][j] = elem;
            }
        }
    return matrix; // возврат измененной матрицы
}
```

```
int* searchMinEl(int* line, int n) { // поиск минимального элемента
    int min = 1000000; // инициализация переменной min максимально
    возможным значением
    for (int j = 0; j < n; j++) { // перебор всех вершин
        if (line[j] >= 0 && line[j] < min) { // поиск минимального
        положительного элемента
```

```

        min = line[j];
    }
}
for (int j = 0; j < n; j++) { // перебор всех вершин
    if (line[j] >= 0) { // уменьшение всех элементов на значение
МИНИМАЛЬНОГО
        line[j] -= min; // обновление
    }
}
return line; // возвращение измененного массива
}

```

```

int** reductMatrix(int** oldmatrix) { // функция редуцирует матрицу
смежности
    int** matrix = oldmatrix; // указатель на матрицу смежности
    int n = graph.amountVertices(); // количество вершин в графе
    for (int i = 0; i < n; i++) { // цикл проходит по всем строкам матрицы
смежности
        matrix[i] = searchMinEl(matrix[i], n); // поиск минимального
элемента в строке i и его редукция
    }
    for (int i = 0; i < n; i++) { // этот цикл проходит по всем столбцам
матрицы смежности
        int min = 1000000; // инициализация min максимально возможным
значением
        for (int j = 0; j < n; j++) { // цикл находит минимальный
положительный элемент в столбце i

```

```

        if (matrix[j][i] >= 0 && matrix[j][i] < min) // если элемент
matrix[j][i] положительный и меньше min
            min = matrix[j][i]; // обновление
    }
    for (int j = 0; j < n; j++) { // цикл уменьшает все элементы в
столбце i на min
        if (matrix[j][i] >= 0) // если элемент matrix[j][i]
положительный
            matrix[j][i] -= min; // уменьшение
    }
}
return matrix; // возврат редуцированной матрицы
}

```

```

int** findZero(int** oldmatrix) { // функция возвращает измененную матрицу
    int n = graph.amountVertices(); // получение количества вершин в графе
    int** matrix = reductMatrix(oldmatrix); // получаем редуцированной
матрицы
    int max = -1; // инициализация переменной, которая будет содержать
максимальное значение
    int line, column; // инициализация переменных для хранения индексов
    for (int i = 0; i < n; i++) { // перебор строк матрицы
        for (int j = 0; j < n; j++) { // перебор столбцов матрицы
            if (matrix[i][j] == 0) { // если элемент матрицы равен нулю
                int minLine = 1000000; // инициализация переменной,
которая будет содержать минимальное значение среди строк
                int minColumn = 1000000; // инициализация
переменной, которая будет содержать минимальное значение среди столбцов

```

```

        for (int k = 0; k < n; k++) { // перебор всех строк
матрицы
            if (matrix[i][k] != -1 && k != j && matrix[i][k] <
minLine) // условие для поиска минимального элемента среди строк
                minLine = matrix[i][k]; // обновление
        }
        for (int k = 0; k < n; k++) { // перебор столбцов
матрицы
            if (matrix[k][j] != -1 && k != i && matrix[k][j] <
minColumn) // условие для поиска минимального элемента среди столбцов
                minColumn = matrix[k][j];
        }
        if (max < minColumn + minLine) { // условие для
обновления максимального значения
            max = minColumn + minLine; // обновление
            line = i; // обновление
            column = j; // обновление
        }
    }
}

pair<int, int> p; // объявление пары для хранения индексов
p.first = line + 1; // обновление первого элемента пары
p.second = column + 1; // обновление второго элемента пары
indexes.push_back(p); // добавление пары в вектор пар
matrix[line][column] = -1; // установка элементов матрицы равными -
1
matrix[column][line] = -1; // установка элементов матрицы равными -
1
for (int i = 0; i < n; i++) { // перебор всех строк матрицы

```



```

        matrix[line][i] = -1; // установка элементов строки равными -1
        matrix[i][column] = -1; // установка элементов столбцов
        равными -1
    }
    return matrix; // возврат измененной матрицы
}

void printAnswer() { // функция печатает результат работы программы
    int second = indexes[0].second; // second используется для хранения
    индекса второго узла в пути
    int i = 2; // переменная i используется для подсчёта количества
    добавленных в путь узлов
    newIndexes.push_back(indexes[0].first); // добавление первого узла в путь
    newIndexes.push_back(indexes[0].second); // добавление второго узла в
    путь
    while (i != graph.amountVertices() + 1) { // пока количество добавленных
    узлов меньше количества всех узлов в графе
        for (int j = 1; j < graph.amountVertices(); j++) { // поиск
        следующего узла в пути
            if (indexes[j].first == second) { // если текущий узел равен
            второму узлу в пути
                second = indexes[j].second; // обновление второго узла
                в пути
                newIndexes.push_back(second); // добавление нового
                второго узла в путь
                i++; // увеличение счётчика добавленных узлов
                break; // выход из цикла, так как узел найден
            }
        }
    }
}

```

```

    }
    cout << "Решение задачи коммивояжера: "; // печать результата
    for (int i = 0; i < newIndexes.size(); i++) { // перебор всех узлов в пути
        cout << newIndexes[i]; // печать текущего узла
        if (i != newIndexes.size() - 1) { // если текущий узел не последний
            cout << " -> "; // печать символа перехода к следующему
узелу
        }
    }
    int sum = 0; // сумма длин ребер в найденном пути
    for (int i = 0; i < indexes.size(); i++) { // перебор всех ребер в пути
        int line = indexes[i].first - 1; // получение номера строки для
текущего ребра
        int column = indexes[i].second - 1; // получение номера столбца для
текущего ребра
        sum += graph.getElFromAdjMatrix(line, column); // добавление
длины ребра к сумме
    }
    cout << endl << "S = " << sum << endl << endl; // печать суммы длин
ребер
}

```

```

void drawCircle(int x, int y, int radius) { // отрисовка окружности для вершины
    glColor3f(255.0f, 235.0f, 192.0f); // установка цвета круга
    glBegin(GL_POLYGON); { // начало рисования
        float theta;
        float X, Y;

```

```

        for (int i = 0; i < 360; i++) {
            theta = 2.0f * 3.1415926f * float(i) / float(360); // вычисление
координат точки на окружности
            X = radius * sin(theta) + x; // координата по x для
окружности
            Y = radius * cos(theta) + y; // координата по y для
окружности

            glVertex2f(X, Y); // добавление точки
        }
    }

    glEnd(); // конец рисования
    glBegin(GL_LINE_LOOP); { // начало рисования замкнутой линии
        for (int i = 0; i < 360; i++) {
            float theta;
            float X, Y;
            theta = 2.0f * 3.1415926f * float(i) / float(360); // вычисление
координат точки на окружности
            X = radius * sin(theta) + x; // координата по x для линии
            Y = radius * cos(theta) + y; // координата по x для линии
            glVertex2f(X, Y); // добавление точки к линии
        }
    }

    glEnd(); // конец рисования замкнутой линии
}

```

```

void drawText(int text, int x1, int y1) { // отображение текста
    glColor3f(0.0, 0.0, 0.0); // установка цвета текста
    GLvoid* font = GLUT_BITMAP_HELVETICA_18; // установка шрифта
    std::string s = std::to_string(text); // преобразование числа text в строку
    glRasterPos2i(x1 - 5, y1 - 5); // определение позиции текста
}

```

```
for (size_t j = 0; j < s.length(); j++) { // отображение каждого символа строки
```

```
    glutBitmapCharacter(font, s[j]);  
}  
}
```

```
void drawLine(int text, int x0, int y0, int x1, int y1) { // отрисовка линий(ребер)  
    glColor3i(0, 0, 0); // установка цвета линии  
    glBegin(GL_LINES); // рисование линии  
    glVertex2i(x0, y0); // указание начала линии  
    glVertex2i(x1, y1); // указание конца линии  
    glEnd(); // конец рисования линии  
    drawText(text, (x0 + x1) / 2 + 10, (y0 + y1) / 2 + 10); // рисование текста с информацией о линии  
}
```

```
void drawVertex(int n) { // функция для отрисовки вершин графа  
    for (int i = 0; i < n; i++) { // цикл для перебора всех вершин  
        drawCircle(coordinates[i].x, coordinates[i].y, radius); // отрисовка круга (вершины)  
        drawText(i + 1, coordinates[i].x, coordinates[i].y); // отрисовка текста (номера вершины)  
    }  
}
```

```
void coords(int i, int n) { // функция для установки координат вершин графа  
    int radius2; // переменная для хранения радиуса круга
```

```

int x0 = width / 2.8; // координаты центра графа
int y0 = height / 2;
if (width > height) { // проверка соотношения сторон окна
    radius = 3 * (height / 13) / n; // расчет радиуса круга
    radius2 = height / 2 - radius - 10; // расчет координаты
}
else {
    radius = 3 * (width / 13) / n; // расчет радиуса круга
    radius2 = width / 2 - radius - 10; // расчет координаты
}
float theta = 2.0f * 3.1415926f * i / n; // расчет угла theta для i-й
вершины
int y1 = radius2 * cos(theta) + y0; // расчет координаты y1
int x1 = radius2 * sin(theta) + x0; // расчет координаты x1
coordinates[i].x = x1; // установка координаты x для i-й вершины
coordinates[i].y = y1; // установка координаты y для i-й вершины
}

void createGraph() { // функция для создания графа
    int amountVerts, AmountEdges, originalVertex, finalVertex, weight; //
объявление переменных
    cout << "Введите количество вершин: "; // запрос у пользователя
количества вершин
    cin >> amountVerts; // ввод количества вершин
    cout << "Введите количество ребер: "; // запрос у пользователя
количества рёбер
    cin >> AmountEdges; // ввод количества рёбер
    cout << endl; // переход на новую строку

```

```

        for (int i = 1; i <= amountVerts; i++) { // цикл для добавления всех
вершин в граф
            graph.insertVertex(i); // добавление вершины в граф
        }
        for (int i = 0; i < AmountEdges; i++) { // цикл для добавления всех
ориентированных ребер в граф
            cout << "Введите начальную вершину: "; // запрос у пользователя
исходной вершины
            cin >> originalVertex; // ввод исходной вершины
            cout << "Введите конечную вершину: "; // запрос у пользователя
конечной вершины
            cin >> finalVertex; // ввод конечной вершины
            cout << "Введите вес ребра: "; // запрос у пользователя веса ребра
            cin >> weight; // ввод веса ребра
            cout << endl;
            graph.insertEdge(originalVertex, finalVertex, weight); // добавление
ребра в граф
        }
        cout << endl; // переход на новую строку
        graph.print(); // вывод графа на экран
    }

```

```

bool checkData(int** matrix) { // функция проверяет, есть ли данные для
решения задачи коммивояжера
    if (graph.checkEmpty()) // проверка, пуст ли граф
        return false;
    for (int i = 0; i < graph.amountVertices(); i++) { // для каждого узла в
графе
        int cnt = 0; // счет количества соседей текущего узла
        for (int j = 0; j < graph.amountVertices(); j++) {

```

```

        if (matrix[i][j] > 0) // если есть ребро от текущего узла к j-му
узелу
            cnt++; // увеличение
    }
    if (cnt < 1) // если узел не имеет соседей, возврат false
        return false;
    }
    return true; // если все узлы имеют соседей, возврат true
}

```

```

void drawButton1() { // отрисовка 1 кнопки
    if (buttonBacklight == 1) { // проверка состояния подсветки кнопки
        glColor3f(245.0f / 255.0f, 222.0f / 255.0f, 179.0f / 255.0f); //
установка цвета подсветки
    }
    else {
        glColor3f(0.9, 0.8, 0.9); // установка цвета кнопки
    }
    glBegin(GL_QUADS); { // рисовка кнопки
        glVertex2i(width - width / 6, height - height / 10 - 20); // верхний
левый угол кнопки
        glVertex2i(width - width / 6, height - 2 * (height / 10)); // нижний
левый угол кнопки
        glVertex2i(width - 50, height - 2 * (height / 10)); // нижний правый
угол кнопки
        glVertex2i(width - 50, height - height / 10 - 20); // верхний правый
угол кнопки
    }
    glEnd();
}

```

```

    glColor3f(0.0f, 0.0f, 0.0f); // установка цвета для границ кнопки
    glBegin(GL_LINE_LOOP); { // граница вокруг кнопки
        glVertex2i(width - width / 6, height - height / 10 - 20); // верхний
        левый угол кнопки
        glVertex2i(width - width / 6, height - 2 * (height / 10)); // нижний
        левый угол кнопки
        glVertex2i(width - 50, height - 2 * (height / 10)); // нижний правый
        угол кнопки
        glVertex2i(width - 50, height - height / 10 - 20); // верхний правый
        угол кнопки
    }
    glEnd();
    string name = "TSP"; // имя кнопки
    glRasterPos2i(width - 130, 0.83 * height); // определение позиции для
    текста на кнопке
    for (int i = 0; i < name.length(); i++) {
        glutBitmapCharacter(GLUT_BITMAP_8_BY_13, name[i]); //
        рисовка текста на кнопке
    }
}

void drawButton2() { // отрисовка 2 кнопки
    if (buttonBacklight == 2) { // проверка состояния подсветки кнопки
        glColor3f(245.0f / 255.0f, 222.0f / 255.0f, 179.0f / 255.0f); //
        установка цвета подсветки
    }
    else {
        glColor3f(0.9, 0.8, 0.9); // установка цвета кнопки
    }
    glBegin(GL_QUADS); { // рисовка кнопки

```



```

        glVertex2i(width - width / 6, height - 2 * (height / 10) - 20); //
верхний левый угол кнопки
        glVertex2i(width - width / 6, height - 3 * (height / 10)); // нижний
левый угол кнопки
        glVertex2i(width - 50, height - 3 * (height / 10)); // нижний правый
угол кнопки
        glVertex2i(width - 50, height - 2 * (height / 10) - 20); // верхний
правый угол кнопки
    }
    glEnd();
    glColor3f(0.0f, 0.0f, 0.0f); // установка цвета для границ кнопки
    glBegin(GL_LINE_LOOP); { // граница вокруг кнопки
        glVertex2i(width - width / 6, height - 2 * (height / 10) - 20); //
верхний левый угол кнопки
        glVertex2i(width - width / 6, height - 3 * (height / 10)); // нижний
левый угол кнопки
        glVertex2i(width - 50, height - 3 * (height / 10)); // нижний правый
угол кнопки
        glVertex2i(width - 50, height - 2 * (height / 10) - 20); // верхний
правый угол кнопки
    }
    glEnd();
    string name = "print matrix"; // имя кнопки
    glRasterPos2i(width - 160, 0.73 * height); // определение позиции для
текста на кнопке
    for (int i = 0; i < name.length(); i++)
        glutBitmapCharacter(GLUT_BITMAP_8_BY_13, name[i]); //
рисовка текста на кнопке
}

void drawButton3() { // отрисовка 3 кнопки

```

```

        if (buttonBacklight == 3) { // проверка состояния подсветки кнопки
            glColor3f(245.0f / 255.0f, 222.0f / 255.0f, 179.0f / 255.0f); //
установка цвета подсветки
        }
        else {
            glColor3f(0.9, 0.8, 0.9); // установка цвета кнопки
        }

        glBegin(GL_QUADS); { // рисовка кнопки
            glVertex2i(width - width / 6, height - 3 * (height / 10) - 20); //
верхний левый угол кнопки
            glVertex2i(width - width / 6, height - 4 * (height / 10)); // нижний
левый угол кнопки
            glVertex2i(width - 50, height - 4 * (height / 10)); // нижний правый
угол кнопки
            glVertex2i(width - 50, height - 3 * (height / 10) - 20); // верхний
правый угол кнопки
        }

        glEnd();

        glColor3f(0.0f, 0.0f, 0.0f); // установка цвета для границ кнопки
        glBegin(GL_LINE_LOOP); { // граница вокруг кнопки
            glVertex2i(width - width / 6, height - 3 * (height / 10) - 20); //
верхний левый угол кнопки
            glVertex2i(width - width / 6, height - 4 * (height / 10)); // нижний
левый угол кнопки
            glVertex2i(width - 50, height - 4 * (height / 10)); // нижний правый
угол кнопки
            glVertex2i(width - 50, height - 3 * (height / 10) - 20); // верхний
правый угол кнопки
        }

        glEnd();

```

```

    string name = "add vertex"; // имя кнопки

    glRasterPos2i(width - 160, 0.63 * height); // определение позиции для
текста на кнопке

    for (int i = 0; i < name.length(); i++)
        glutBitmapCharacter(GLUT_BITMAP_8_BY_13, name[i]); //
рисовка текста на кнопке

}

void drawButton4() { // отрисовка 4 кнопки

    if (buttonBacklight == 4) { // проверка состояния подсветки кнопки
        glColor3f(245.0f / 255.0f, 222.0f / 255.0f, 179.0f / 255.0f); //
установка цвета подсветки
    }
    else {
        glColor3f(0.9, 0.8, 0.9); // установка цвета кнопки
    }

    glBegin(GL_QUADS); { // рисовка кнопки
        glVertex2i(width - width / 6, height - 4 * (height / 10) - 20); //
верхний левый угол кнопки
        glVertex2i(width - width / 6, height - 5 * (height / 10)); // нижний
левый угол кнопки
        glVertex2i(width - 50, height - 5 * (height / 10)); // нижний правый
угол кнопки
        glVertex2i(width - 50, height - 4 * (height / 10) - 20); // верхний
правый угол кнопки
    }

    glEnd();

    glColor3f(0.0f, 0.0f, 0.0f); // установка цвета для границ кнопки
    glBegin(GL_LINE_LOOP); { // граница вокруг кнопки

```

```

        glVertex2i(width - width / 6, height - 4 * (height / 10) - 20); //
верхний левый угол кнопки
        glVertex2i(width - width / 6, height - 5 * (height / 10)); // нижний
левый угол кнопки
        glVertex2i(width - 50, height - 5 * (height / 10)); // нижний правый
угол кнопки
        glVertex2i(width - 50, height - 4 * (height / 10) - 20); // верхний
правый угол кнопки
    }
    glEnd();
    string name = "delete vertex"; // имя кнопки
    glRasterPos2i(width - 163, 0.53 * height); // определение позиции для
текста на кнопке
    for (int i = 0; i < name.length(); i++)
        glutBitmapCharacter(GLUT_BITMAP_8_BY_13, name[i]); //
рисовка текста на кнопке
}

void drawButton5() { // отрисовка 5 кнопки
    if (buttonBacklight == 5) { // проверка состояния подсветки кнопки
        glColor3f(245.0f / 255.0f, 222.0f / 255.0f, 179.0f / 255.0f); //
установка цвета подсветки
    }
    else {
        glColor3f(0.9, 0.8, 0.9); // установка цвета кнопки
    }

    glBegin(GL_QUADS); { // рисовка кнопки
        glVertex2i(width - width / 6, height - 5 * (height / 10) - 20); //
верхний левый угол кнопки
        glVertex2i(width - width / 6, height - 6 * (height / 10)); // нижний
левый угол кнопки

```

```

        glVertex2i(width - 50, height - 6 * (height / 10)); // нижний правый
угол кнопки
        glVertex2i(width - 50, height - 5 * (height / 10) - 20); // верхний
правый угол кнопки
    }
    glEnd();
    glColor3f(0.0f, 0.0f, 0.0f); // установка цвета для границ кнопки
    glBegin(GL_LINE_LOOP); { // граница вокруг кнопки
        glVertex2i(width - width / 6, height - 5 * (height / 10) - 20); //
верхний левый угол кнопки
        glVertex2i(width - width / 6, height - 6 * (height / 10)); // нижний
левый угол кнопки
        glVertex2i(width - 50, height - 6 * (height / 10)); // нижний правый
угол кнопки
        glVertex2i(width - 50, height - 5 * (height / 10) - 20); // верхний
правый угол кнопки
    }
    glEnd();
    string name = "add edge"; // имя кнопки
    glRasterPos2i(width - 160, 0.43 * height); // определение позиции для
текста на кнопке
    for (int i = 0; i < name.length(); i++)
        glutBitmapCharacter(GLUT_BITMAP_8_BY_13, name[i]); //
рисовка текста на кнопке
}

void drawButton6() { // отрисовка 6 кнопки
    if (buttonBacklight == 6) { // проверка состояния подсветки кнопки
        glColor3f(245.0f / 255.0f, 222.0f / 255.0f, 179.0f / 255.0f); //
установка цвета подсветки
    }
}

```

```

else {
    glColor3f(0.9, 0.8, 0.9); // установка цвета кнопки
}

glBegin(GL_QUADS); { // рисовка кнопки
    glVertex2i(width - width / 6, height - 6 * (height / 10) - 20); //
верхний левый угол кнопки
    glVertex2i(width - width / 6, height - 7 * (height / 10)); // нижний
левый угол кнопки
    glVertex2i(width - 50, height - 7 * (height / 10)); // нижний правый
угол кнопки
    glVertex2i(width - 50, height - 6 * (height / 10) - 20); // верхний
правый угол кнопки
}
glEnd();
glColor3f(0.0f, 0.0f, 0.0f); // установка цвета для границ кнопки
glBegin(GL_LINE_LOOP); { // граница вокруг кнопки
    glVertex2i(width - width / 6, height - 6 * (height / 10) - 20); //
верхний левый угол кнопки
    glVertex2i(width - width / 6, height - 7 * (height / 10)); // нижний
левый угол кнопки
    glVertex2i(width - 50, height - 7 * (height / 10)); // нижний правый
угол кнопки
    glVertex2i(width - 50, height - 6 * (height / 10) - 20); // верхний
правый угол кнопки
}
glEnd();
string name = "delete edge"; // имя кнопки
glRasterPos2i(width - 160, 0.33 * height); // определение позиции для
текста на кнопке
for (int i = 0; i < name.length(); i++)

```

```
glutBitmapCharacter(GLUT_BITMAP_8_BY_13, name[i]); //
```

рисовка текста на кнопке

```
}
```

```
void hoverCheckButton(int x, int y) { // проверка, курсор находится на кнопке  
или нет
```

```
    if (x > (width - width / 6) && x < (width - 50) && y > (height - 2 * (height /  
10)) && y < (height - (height / 10) - 20)) {
```

```
        buttonBacklight = 1; // если курсор находится над первой кнопкой,  
то подсветка кнопки устанавливается в значение 1
```

```
    }
```

```
    else if (x > (width - width / 6) && x < (width - 50) && y > (height - 3 *  
(height / 10)) && y < (height - 2 * (height / 10) - 20)) {
```

```
        buttonBacklight = 2; // если курсор находится над второй кнопкой,  
то подсветка кнопки устанавливается в значение 2
```

```
    }
```

```
    else if (x > (width - width / 6) && x < (width - 50) && y > (height - 4 *  
(height / 10)) && y < (height - 3 * (height / 10) - 20)) {
```

```
        buttonBacklight = 3; // Если курсор находится над третьей  
кнопкой, то подсветка кнопки устанавливается в значение 3
```

```
    }
```

```
    else if (x > (width - width / 6) && x < (width - 50) && y > (height - 5 *  
(height / 10)) && y < (height - 4 * (height / 10) - 20)) {
```

```
        buttonBacklight = 4; // если курсор находится над четвёртой  
кнопкой, то подсветка кнопки устанавливается в значение 4
```

```
    }
```

```
    else if (x > (width - width / 6) && x < (width - 50) && y > (height - 6 *  
(height / 10)) && y < (height - 5 * (height / 10) - 20)) {
```

```
        buttonBacklight = 5; // если курсор находится над пятой кнопкой,  
то подсветка кнопки устанавливается в значение 5
```

```

    }
    else if (x > (width - width / 6) && x < (width - 50) && y > (height - 7 *
(height / 10)) && y < (height - 6 * (height / 10) - 20)) {
        buttonBacklight = 6; // если курсор находится над шестой кнопкой,
то подсветка кнопки устанавливается в значение 6
    }
    else {
        buttonBacklight = 0; // если курсор не находится ни над одной из
кнопок, то подсветка кнопки устанавливается в значение 0
    }
}

```

```

void mouseMove(int x, int y) { // перемещение мыши по экрану
    y = height - y; // переворот оси Y для корректного отображения
    hoverCheckButton(x, y); // проверка на какой кнопкой находится курсор
    glutPostRedisplay(); // обновление отображения окна
}

```

```

void mouseClicked(int button, int state, int x, int y) { // действия при нажатии на
кнопки
    if (x > (width - width / 6) && x < (width - 50) && y > 93 && y < 150) { //
подходят ли текущие координаты под кнопку 1
        if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
            // нажата ли кнопка
            newIndexes.clear(); // очищение вектора
            indexes.clear(); // очищение вектора
            cout << ": " << endl;
            int** matrix = modificaMatrix(); // получение
редуцированной матрицы

```



```

bool checker = checkData(matrix); // проверка, подходят ли
данные
if (!checker) {
    cout << "Неподходящие данные для решения задачи
коммивояжера!" << endl;
    return;
}
int n = graph.amountVertices(); // количество вершин
while (indexes.size() < n)
    matrix = findZero(matrix); // нахождение измененной
матрицы

cout << endl;
printAnswer(); // вывод ответа на задачу коммивояжера
return;
}
}

if (x > (width - width / 6) && x < (width - 50) && y > 175 && y < 233) { //
подходят ли текущие координаты под кнопку 2
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
// нажата ли кнопка
        graph.print(); // печать матрицы смежности
        return;
    }
}

if (x > (width - width / 6) && x < (width - 50) && y > 253 && y < 312) { //
подходят ли текущие координаты под кнопку 3
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
// нажата ли кнопка
        int cur = graph.amountVertices(); // количество вершин
        graph.insertVertex(cur + 1); // вставка вершины

```

```

        coordinates[cur].x = width / 2; // обновление координат
        coordinates[cur].y = height / 2; // обновление координат
        return;
    }
}

if (x > (width - width / 6) && x < (width - 50) && y > 330 && y < 390) { //
подходят ли текущие координаты под кнопку 4
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
// нажата ли кнопка
        graph.eraseVertex(); // удаление вершины
        return;
    }
}

if (x > (width - width / 6) && x < (width - 50) && y > 408 && y < 466) { //
подходят ли текущие координаты под кнопку 5
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
// нажата ли кнопка
        int firstVertex, secondVertex, Weight;
        cout << "Введите первую вершину для ребра: ";
        cin >> firstVertex;
        cout << "Введите вторую вершину для ребра: ";
        cin >> secondVertex;
        cout << "Введите вес нового ребра: ";
        cin >> Weight;
        cout << endl;
        graph.insertEdge(firstVertex, secondVertex, Weight); // вставка
заданного ребра
        return;
    }
}
}

```

```

        if (x > 899 && x < 1030 && y <= 546 && y >= 486) { // подходят ли
текущие координаты под кнопку 6
            if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
// нажата ли кнопка
                int firstVertex, secondVertex;
                cout << "Введите первую вершину для ребра: ";
                cin >> firstVertex;
                cout << "Введите вторую вершину для ребра: ";
                cin >> secondVertex;
                cout << endl;
                graph.eraseEdge(firstVertex, secondVertex); // удаление
заданного ребра
                return;
            }
        }
    }
}

```

```

void drawTitle() { // вывод заголовка
    string title = "ELECTIVE ACTIONS"; // имя заголовка
    glRasterPos2i(900, height-60); // координаты заголовка
    for (int i = 0; i < title.length(); i++) // отрисовка по букве
        glutBitmapCharacter(GLUT_BITMAP_9_BY_15, title[i]);
}

```

```

void reshape(int w, int h) { // ресайз окна OpenGL
    width = w; // ширину нового окна
    height = h; // высоту нового окна
    glViewport(0, 0, (GLsizei)width, (GLsizei)height); // область просмотра
для нового размера окна
}

```

```

    glMatrixMode(GL_PROJECTION); // переключение на матрицу
    проекции
    glLoadIdentity(); // очистка текущей матрицы
    gluOrtho2D(0, (GLdouble)width, 0, (GLdouble)height); // определение
    проекции с центром в начале координат и размерами окна
    glutPostRedisplay(); // обновление отображения окна
}

```

```

void display() { // основной цикл отрисовки
    glMatrixMode(GL_PROJECTION); // переключение на матрицу
    проекции
    glLoadIdentity(); // очистка текущей матрицы
    gluOrtho2D(0, width, 0, height); // определение проекции с центром в
    начале координат и размерами окна
    glViewport(0, 0, width, height); // область просмотра для текущего
    размера окна
    glClearColor(1.0, 0.8, 0.8, 0.8); // цвет фона
    glClear(GL_COLOR_BUFFER_BIT); // очистка экрана
    drawTitle(); // рисовка заголовка
    drawButton1(); // рисовка первой кнопки
    drawButton2(); // рисовка второй кнопки
    drawButton3(); // рисовка третьей кнопки
    drawButton4(); // рисовка четвертой кнопки
    drawButton5(); // рисовка пятой кнопки
    drawButton6(); // рисовка шестой кнопки
    graph.drawGraph(); // рисовка графика
    glutSwapBuffers(); // смена буферов
}

```

Файл main.cpp:

```

#include "Graph.h" // подключение файла с определением структуры данных

```

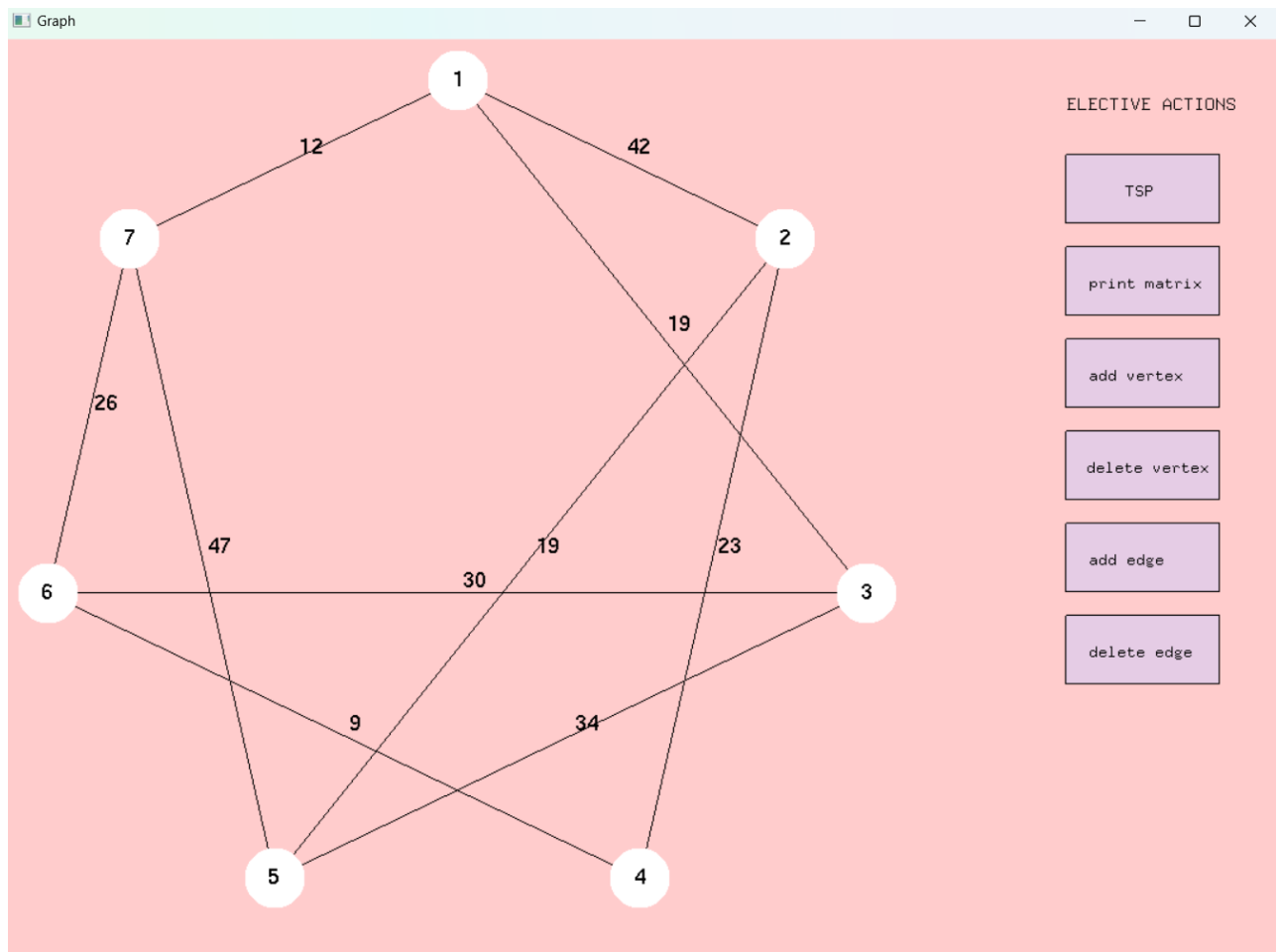
```
#include <gl/glut.h> // подключение заголовочного файла библиотеки OpenGL
#include <iostream> // подключение заголовочного файла, который позволяет
использовать потоковый ввод-вывод
using namespace std; // разрешает использование пространства имен
стандартной библиотеки

int main(int argc, char** argv) {
    system("chcp 1251"); // установление кодовой страницы для
корректного отображения русских символов
    glutInit(&argc, argv); // инициализация GLUT
    createGraph(); // вызов функции создания графа
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA); // задание
режима отображения окна
    glutInitWindowSize(width, height); // задание размера окна
    glutCreateWindow("Graph"); // создание окна с заголовком
    glutDisplayFunc(display); // задание функции для отображения
содержимого окна
    glutReshapeFunc(reshape); // задание функции, которая вызывается при
изменении размеров окна
    glutMouseFunc(mouseClick); // задание функции, которая будет
вызываться при клике мыши
    glutPassiveMotionFunc(mouseMove); // задание функции, которая будет
вызываться при движении курсора мыши
    glutMainLoop(); // запуск главного цикла обработки событий GLUT
    return 0; // завершение выполнения программы
}
```

5. Результат работы программы:

Граф взят из лабораторной работы “Введение в теорию графов.

Алгоритмы Дейкстры и Флойда” 12 варианта.



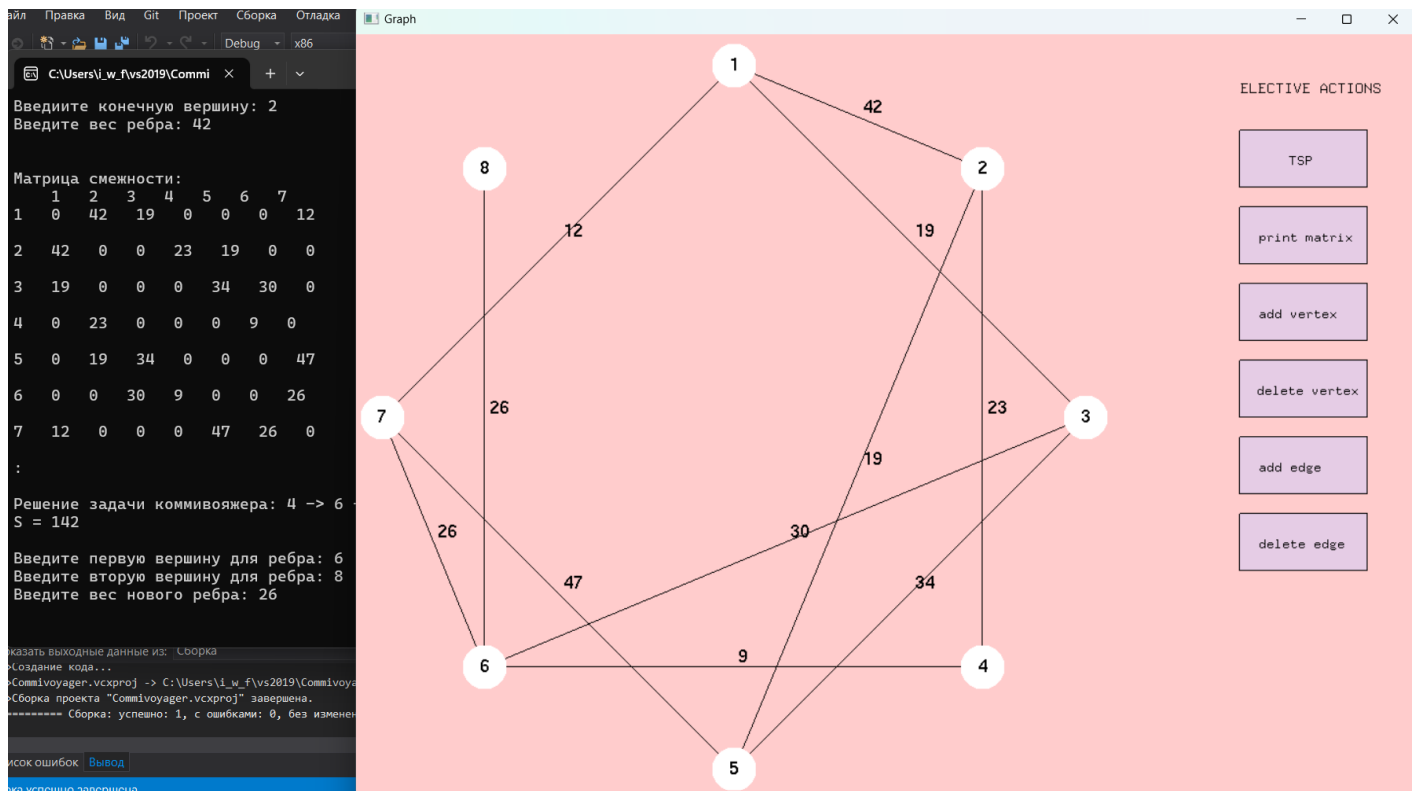
Матрица смежности графа.

Матрица смежности:								
	1	2	3	4	5	6	7	
1	0	42	19	0	0	0	0	12
2	42	0	0	23	19	0	0	
3	19	0	0	0	34	30	0	
4	0	23	0	0	0	9	0	
5	0	19	34	0	0	0	47	
6	0	0	30	9	0	0	26	
7	12	0	0	0	47	26	0	

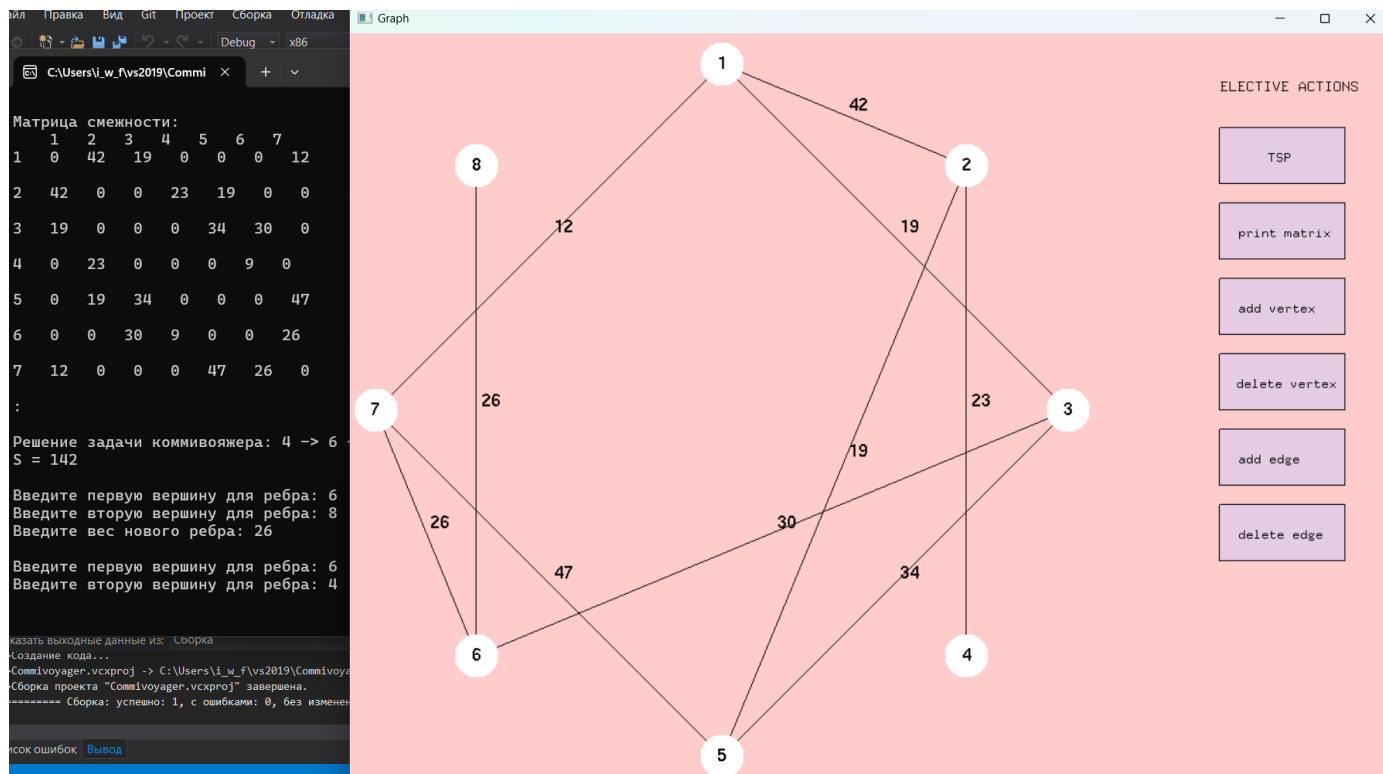
Решение задачи коммивояжера для графа.

Решение задачи коммивояжера: 4 -> 6 -> 7 -> 1 -> 3 -> 5 -> 2 -> 4
S = 142

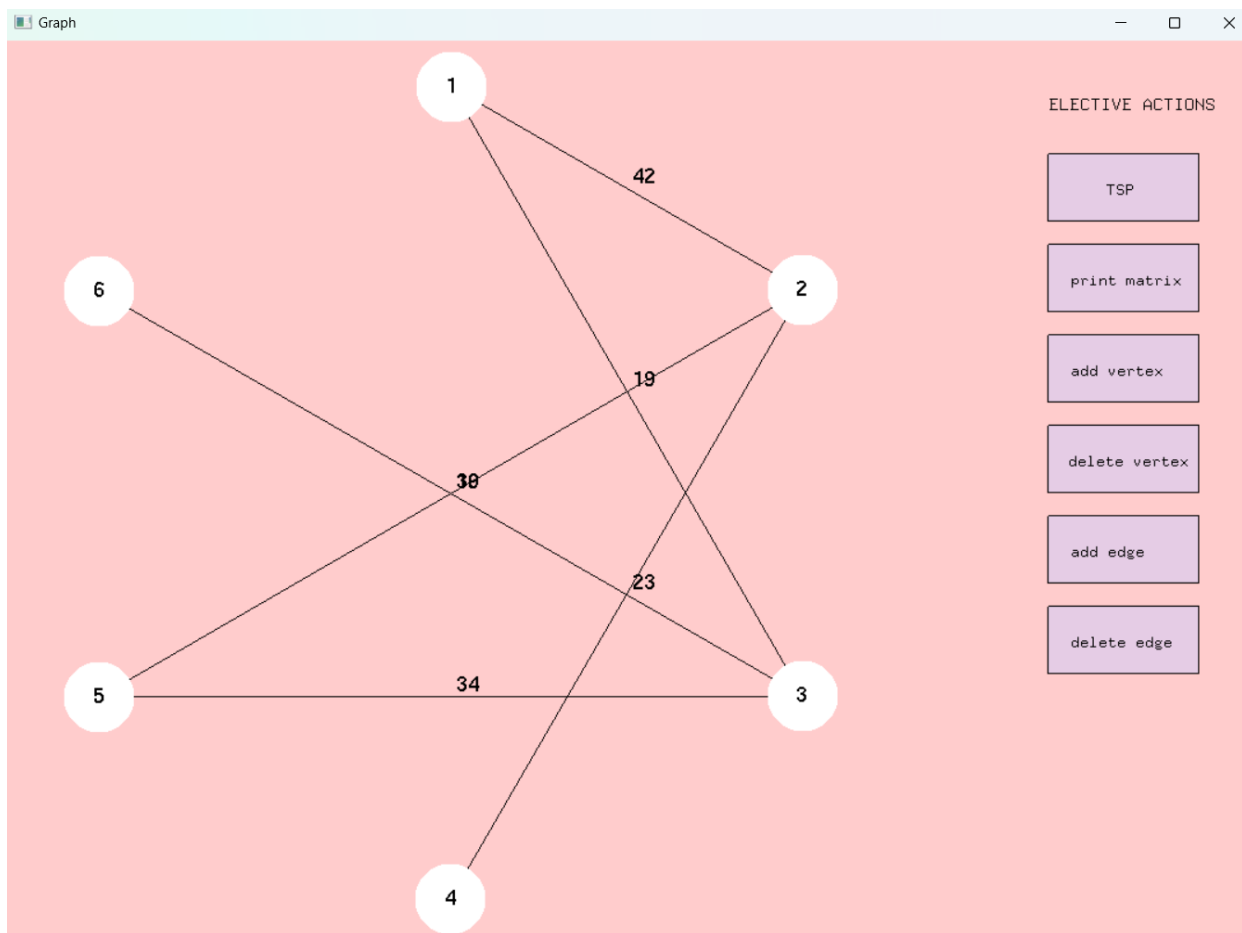
Вставка новой вершины (8) и добавление нового ребра (между 8 и 6 с весом 26).



Удаление ребра (между 6 и 4).



Удаление двух вершин (8 и 7).

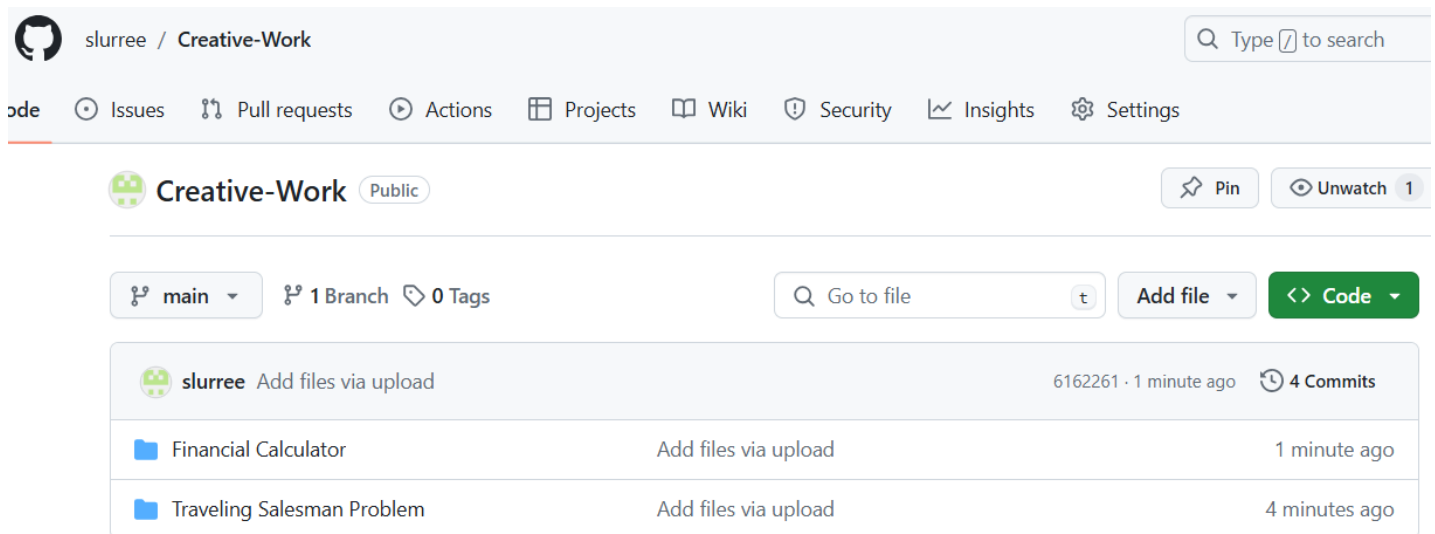


Выводы: Программа финансового калькулятора представляет собой простое и удобное приложение, является полезным инструментом, позволяющим быстро и точно рассчитать основные финансовые показатели. Вторая программа представляет собой эффективный инструмент для решения задачи коммивояжера с использованием метода ветвей и границ. Она позволяет пользователю визуализировать графы, редактировать и решать TSP.

Ссылка на видеоролик в YouTube с демонстрацией работы:

https://youtu.be/UI7_gvx3C3k

Ссылка на работу в GitHub: <https://github.com/slurree/Creative-Work.git>



The screenshot shows the GitHub interface for the repository 'Creative-Work' by user 'slurree'. The repository is public and has 1 branch (main) and 0 tags. The file list shows two folders: 'Financial Calculator' and 'Traveling Salesman Problem', both added via upload. The repository has 4 commits and was last updated 1 minute ago.

File/Folder	Action	Time
Financial Calculator	Add files via upload	1 minute ago
Traveling Salesman Problem	Add files via upload	4 minutes ago

