Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по творческой работе**

Темы: “ Калькулятор. Градусная система мер.

Решение задачи коммивояжёра”

Вариант 16

Выполнил:

Студент группы ИВТ-20-2Б Тедеев Алесандр Зурабович

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

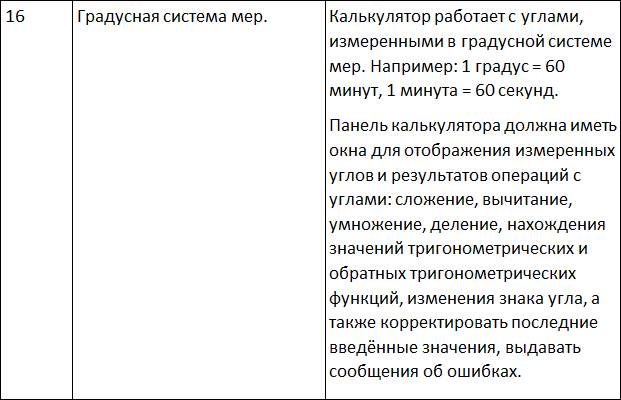
Пермь, 2021

**Постановка задачи**

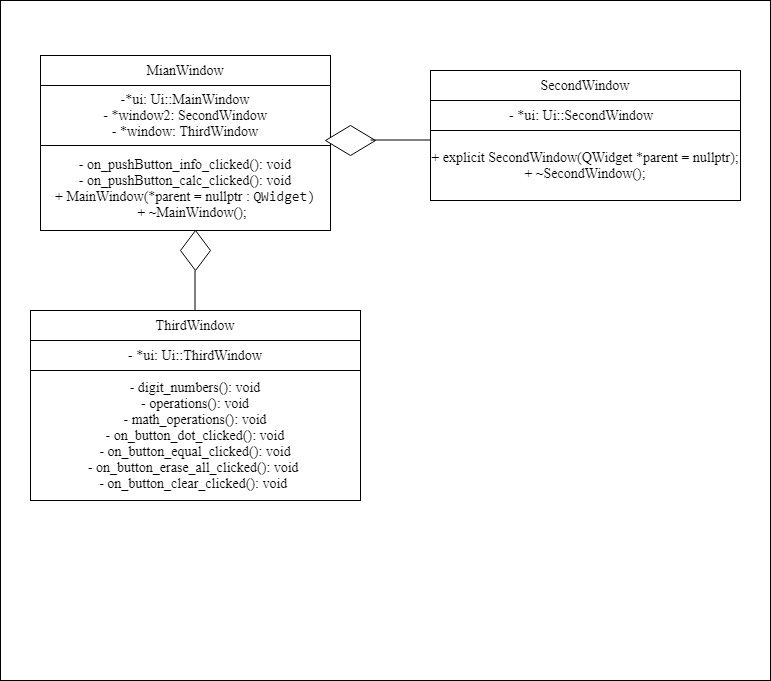
1. Разработать алгоритм калькулятора в соответствии с заданием.

2. Реализовать алгоритм в виде программы на алгоритмическом языке С++.

3. Разработать интерфейс средствами Qt.



**Диаграмма классов**

****

**Анализ задачи**

**1.** Для решения задач необходимо:

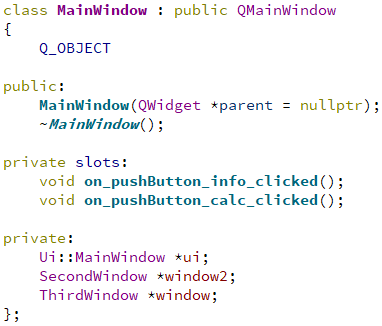
**1.1.** Организовать класс MainWindow, окно которого будет являться меню для запуска калькулятора, либо открытия постановки задачи

**1.2.** Организовать класс SecondWindow, окно которого будет содержать постановку задачи либо в текстовом формате, либо как фоновая картинка.

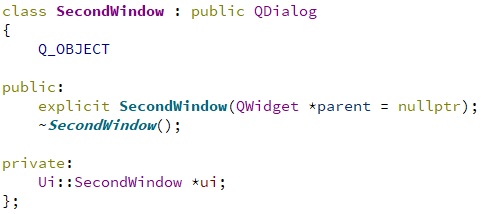
**1.3.** Организовать класс ThirdWindow, окно которого будет представлять из себя калькулятор.

**2.** В ходе работы были использованы следующие типы данных:

**2.1.** Класс MainWindow с двумя слотами под кнопку запуска окна с постановкой задачи – on\_pushButton\_info\_clicked(), и калькулятора - on\_pushButton\_calc\_clicked() и указателями window2 и window для запуска данных окон.



**2.2.** Класс SecondWindow для вывода постановки задачи.

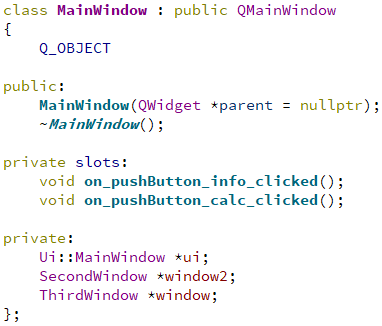


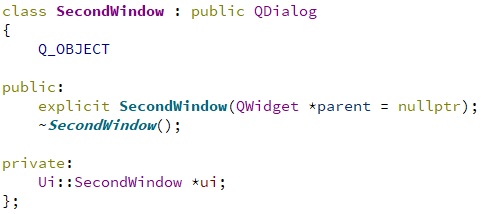
**2.3.** Класс ThirdWindow для вывода окна калькулятора с необходимыми для него слотами.



**3.** Для решения задачи данные были представлены в следующем виде:

**3.1.** Под каждое окно используются свои классы и формы

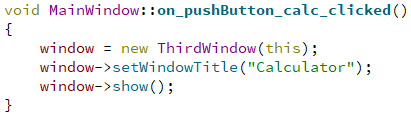






**4.** Для операций ввода и вывода использовались следующие операторы и функции:

**4.1.** Для запуска калькулятора используется слот on\_pushButton\_calc\_clicked().



**4.2.** К данному слоту подключаются кнопки ввода из калькулятора для отдельных функций обработки действий в окне.



**4.3.** Для вывода данных в поля с числами используются команды.

Так как поля заполняются в текстовом формате, для отправки чисел в данные поля необходимо преобразовать числа в формат QString, а затем использовать метод setText() для вывода значений в полях окна.

ui->degree\_result->setText(degrees);

ui->radians\_result->setText(rad);

ui->minutes\_result->setText(mins);

ui->seconds\_result->setText(secs);

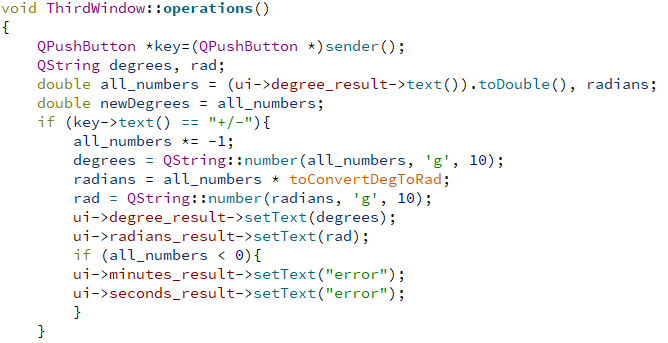
**5.** Поставленные задачи будут решены следующими действиями:

**5.1.** Ввод через кнопки калькулятора обеспечивается функцией digit\_numbers().

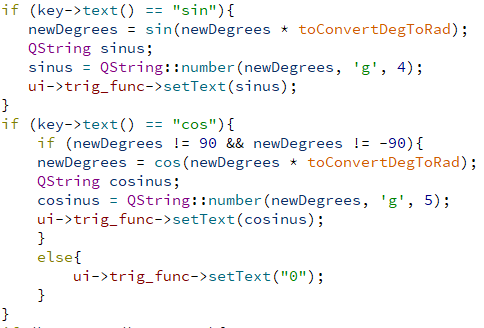


В указатель key поступает данное с нажатой кнопки, в переменной all\_numbers, в которой значение из поля переводится в число через метод toDouble(). В переменных minutes и seconds производится перевод в минуты и секунды градусов, в переменных degrees, secs, mins производится преобразование в число и через метод setText().

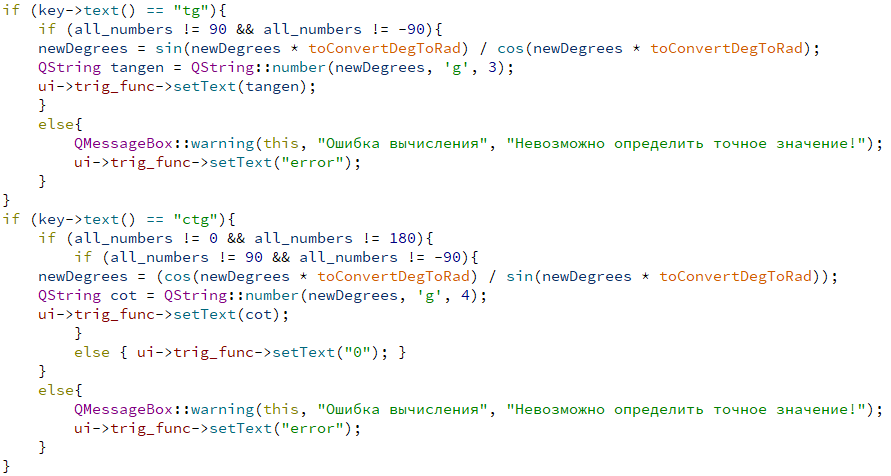
**5.2.** В слоте operations() реализованы операция смены знака, использование тригонометрических и обратных тригонометрических функций.



Для функций синуса и косинуса значение из поля градусов преобразутеся в радианы с помощью глобальной переменной toConvertDegToRad.



Для функций тангенс и котангенс, а также обратных тригонометрических функций, реализованы проверки в соответствии с ограничениями данных функций. В случае попадания за область определения функций выводится ошибка через класс QMessageBox (рисунок 1)



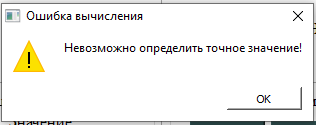


Рисунок 1

if (key->text() == "arcsin"){

if (all\_numbers >= -90 && all\_numbers <= 90){

newDegrees = asin(newDegrees \* toConvertDegToRad);

QString arcsin = QString::number(newDegrees, 'g', 4);

if (all\_numbers > 57){

QMessageBox::warning(this, "Ошибка вычисления", "Почему-то встроенная функция не считает эту фунцию для углов, больших 57");

}

ui->obr\_trig\_func->setText(arcsin);

}

else{

QMessageBox::warning(this, "Ошибка вычисления", "Невозможно определить точное значение!");

ui->obr\_trig\_func->setText("error");

}

}

if (key->text() == "arccos"){

if (all\_numbers >= 0 && all\_numbers <= 180){

newDegrees = acos(newDegrees \* toConvertDegToRad);

QString arccos = QString::number(newDegrees, 'g', 4);

if (all\_numbers > 57){

QMessageBox::warning(this, "Ошибка вычисления", "Почему-то встроенная функция не считает эту фунцию для углов, больших 57");

}

ui->obr\_trig\_func->setText(arccos);

}

else{

QMessageBox::warning(this, "Ошибка вычисления", "Невозможно определить точное значение!");

ui->obr\_trig\_func->setText("error");

}

}

if (key->text()== "arctg"){

if (all\_numbers > -90 && all\_numbers < 90){

newDegrees = atan(newDegrees \* toConvertDegToRad);

QString arctan = QString::number(newDegrees, 'g', 4);

ui->obr\_trig\_func->setText(arctan);

}

else{

QMessageBox::warning(this, "Ошибка вычисления", "Невозможно определить точное значение!");

ui->obr\_trig\_func->setText("error");

}

}

if (key->text() == "arcctg"){

if (all\_numbers > 0 && all\_numbers < 180){

newDegrees = atan(1 / (newDegrees \* toConvertDegToRad));

QString arccot = QString::number(newDegrees, 'g', 4);

ui->obr\_trig\_func->setText(arccot);

}

else{

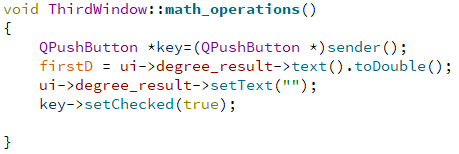
QMessageBox::warning(this, "Ошибка вычисления", "Невозможно определить точное значение!");

ui->obr\_trig\_func->setText("error");

}

}

**5.3.** В методе math\_operations() запоминается операция, кнопку которой нажали: если была выбрана операция сложения, выставляется флажок, который необходим в другом слоте, а также в переменную firstD запоминается число, над которым необходимо произвести действие.



**5.4.** В методе on\_button\_equal\_clicked() осуществляются вычтсления: сложение, вычитание, деление, умножение.

void ThirdWindow::on\_button\_equal\_clicked()

{

double result, secondD, radians, minutes, seconds;

QString degrees, rad, mins, secs;

secondD = ui->degree\_result->text().toDouble();

if (ui->button\_division->isChecked()){

if (secondD != 0){

result = firstD / secondD;

degrees = QString::number(result, 'g', 10);

radians = result \* toConvertDegToRad;

minutes = result \* 60;

seconds = minutes \* 60;

mins = QString::number(minutes, 'g', 5);

secs = QString::number(seconds, 'g', 5);

rad = QString::number(radians, 'g', 10);

ui->minutes\_result->setText(mins);

ui->seconds\_result->setText(secs);

ui->degree\_result->setText(degrees);

ui->radians\_result->setText(rad);

}else{

QMessageBox::warning(this, "error", "На нуль лучше не делить");

}

ui->button\_division->setChecked(false);

} else if (ui->button\_multiplication->isChecked()){

result = firstD \* secondD;

degrees = QString::number(result, 'g', 10);

radians = result \* toConvertDegToRad;

rad = QString::number(radians, 'g', 10);

ui->degree\_result->setText(degrees);

ui->radians\_result->setText(rad);

minutes = result \* 60;

seconds = minutes \* 60;

mins = QString::number(minutes, 'g', 5);

secs = QString::number(seconds, 'g', 5);

ui->minutes\_result->setText(mins);

ui->seconds\_result->setText(secs);

ui->button\_multiplication->setChecked(false);

} else if (ui->button\_minus->isChecked()){

result = firstD - secondD;

degrees = QString::number(result, 'g', 10);

radians = result \* toConvertDegToRad;

rad = QString::number(radians, 'g', 10);

ui->degree\_result->setText(degrees);

ui->radians\_result->setText(rad);

ui->button\_minus->setChecked(false);

minutes = result \* 60;

seconds = minutes \* 60;

mins = QString::number(minutes, 'g', 5);

secs = QString::number(seconds, 'g', 5);

ui->minutes\_result->setText(mins);

ui->seconds\_result->setText(secs);

} else if (ui->button\_plus->isChecked()){

result = firstD + secondD;

degrees = QString::number(result, 'g', 10);

radians = result \* toConvertDegToRad;

rad = QString::number(radians, 'g', 10);

minutes = result \* 60;

seconds = minutes \* 60;

mins = QString::number(minutes, 'g', 5);

secs = QString::number(seconds, 'g', 5);

rad = QString::number(radians, 'g', 10);

ui->minutes\_result->setText(mins);

ui->seconds\_result->setText(secs);

ui->degree\_result->setText(degrees);

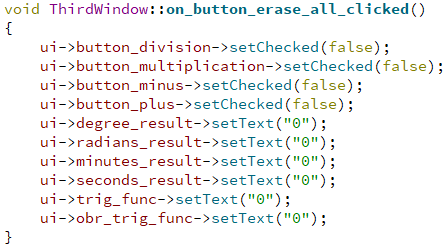
ui->radians\_result->setText(rad);

}

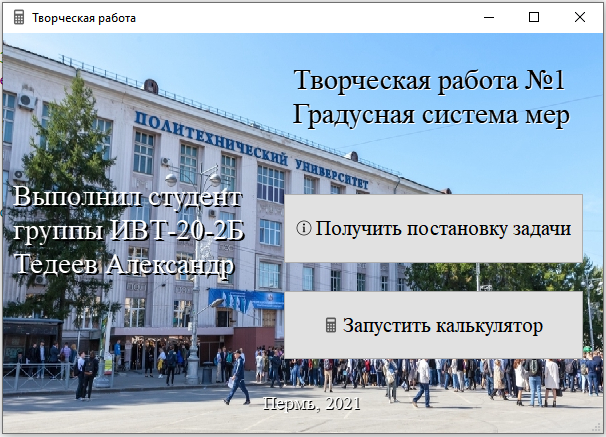
}

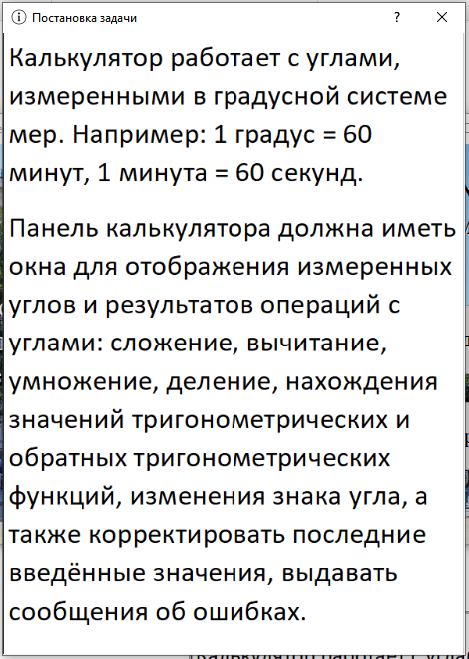
Вычисления происходят по одному алгоритму: в переменную secondD запоминается второе введённое число, в переменную result запоминается выичсленное число, оно переводится в минуты, секунды, радианы. Значения выводятся в нужные поля, и флажок, отвечающий за данную операцию.

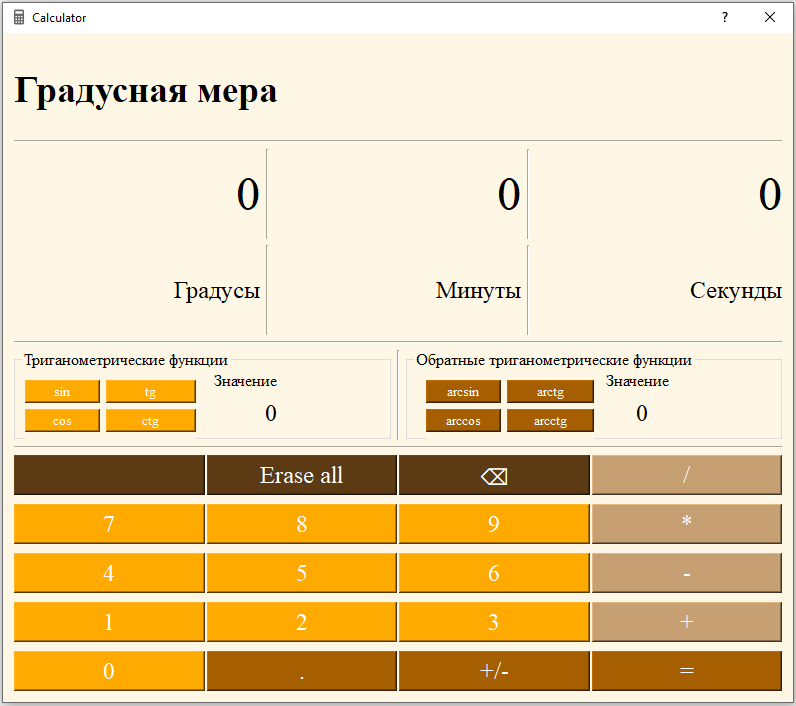
**5.5.** Метод on\_button\_erase\_all\_clicked() используются при нажатии кнопки Erase\_all для очищения всех полей от чисел. Кроме того, в данной функции все флажки операций.



**Скриншоты**







Код

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "secondwindow.h"

#include "thirdwindow.h"

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_pushButton\_info\_clicked();

void on\_pushButton\_calc\_clicked();

private:

Ui::MainWindow \*ui;

SecondWindow \*window2;

ThirdWindow \*window;

};

#endif // MAINWINDOW\_H

#ifndef SECONDWINDOW\_H

#define SECONDWINDOW\_H

#include <QDialog>

#include <QPixmap>

namespace Ui {

class SecondWindow;

}

class SecondWindow : public QDialog

{

Q\_OBJECT

public:

explicit SecondWindow(QWidget \*parent = nullptr);

~SecondWindow();

private:

Ui::SecondWindow \*ui;

};

#endif // SECONDWINDOW\_H

#ifndef THIRDWINDOW\_H

#define THIRDWINDOW\_H

#include <QDialog>

#include <QMessageBox>

#include <QString>

#include <QPushButton>

#include <QMessageBox>

#include <QPalette>

#include <QPixmap>

namespace Ui {

class ThirdWindow;

}

class ThirdWindow : public QDialog

{

Q\_OBJECT

public:

explicit ThirdWindow(QWidget \*parent = nullptr);

~ThirdWindow();

private:

Ui::ThirdWindow \*ui;

private slots:

void digit\_numbers();

void operations();

void math\_operations();

void on\_button\_dot\_clicked();

void on\_button\_equal\_clicked();

void on\_button\_erase\_all\_clicked();

void on\_button\_clear\_clicked();

};

#endif // THIRDWINDOW\_H

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.setWindowTitle("Творческая работа");

w.show();

return a.exec();

}

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QMessageBox>

#include <QGraphicsDropShadowEffect>

#include <QIcon>

#include <QPixmap>

#include <QPalette>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

QPixmap agnles(":/img/img/1050\_16x9-1618578324bDO3AETSRnd1g2D3lle6OJ3ZQwNPKEwiEGoQIdV4.jpeg");

agnles = agnles.scaled(this->size(), Qt::IgnoreAspectRatio);

QPalette palette;

palette.setBrush(QPalette::Background, agnles);

this->setPalette(palette);

QGraphicsDropShadowEffect \*eff = new QGraphicsDropShadowEffect(this);

eff->setOffset(2, 2);

eff->setBlurRadius(0.5);

eff->setColor(Qt::black);

ui->sign->setGraphicsEffect(eff);

QGraphicsDropShadowEffect \*eff2 = new QGraphicsDropShadowEffect(this);

eff2->setOffset(1, 1);

eff2->setBlurRadius(0.5);

eff2->setColor(Qt::black);

ui->place\_year->setGraphicsEffect(eff2);

QGraphicsDropShadowEffect \*eff3 = new QGraphicsDropShadowEffect(this);

eff3->setOffset(1, 1);

eff3->setBlurRadius(1);

eff3->setColor(Qt::white);

ui->title->setGraphicsEffect(eff3);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_pushButton\_info\_clicked()

{

window2 = new SecondWindow(this);

window2->setWindowTitle("Постановка задачи");

window2->show();

}

void MainWindow::on\_pushButton\_calc\_clicked()

{

window = new ThirdWindow(this);

window->setWindowTitle("Calculator");

window->show();

}

#include "secondwindow.h"

#include "ui\_secondwindow.h"

SecondWindow::SecondWindow(QWidget \*parent) :

QDialog(parent),

ui(new Ui::SecondWindow)

{

ui->setupUi(this);

QPixmap secondpic(":/img/img/s.PNG");

int w = ui->background2->width();

int h = ui->background2->height();

ui->background2->setPixmap(secondpic.scaled(w,h,Qt::KeepAspectRatio));

}

SecondWindow::~SecondWindow()

{

delete ui;

}

#include "thirdwindow.h"

#include "ui\_thirdwindow.h"

#include <cmath>

double firstD;

const double toConvertDegToRad = 0.01745329251;

ThirdWindow::ThirdWindow(QWidget \*parent) :

QDialog(parent),

ui(new Ui::ThirdWindow)

{

ui->setupUi(this);

connect(ui->button\_null, SIGNAL(clicked()), this, SLOT(digit\_numbers()));

connect(ui->button\_one, SIGNAL(clicked()), this, SLOT(digit\_numbers()));

connect(ui->button\_two, SIGNAL(clicked()), this, SLOT(digit\_numbers()));

connect(ui->button\_three, SIGNAL(clicked()), this, SLOT(digit\_numbers()));

connect(ui->button\_four, SIGNAL(clicked()), this, SLOT(digit\_numbers()));

connect(ui->button\_five, SIGNAL(clicked()), this, SLOT(digit\_numbers()));

connect(ui->button\_six, SIGNAL(clicked()), this, SLOT(digit\_numbers()));

connect(ui->button\_seven, SIGNAL(clicked()), this, SLOT(digit\_numbers()));

connect(ui->button\_eight, SIGNAL(clicked()), this, SLOT(digit\_numbers()));

connect(ui->button\_nine, SIGNAL(clicked()), this, SLOT(digit\_numbers()));

connect(ui->button\_plus\_minus, SIGNAL(clicked()), this, SLOT(operations()));

connect(ui->button\_division, SIGNAL(clicked()), this, SLOT(math\_operations()));

connect(ui->button\_multiplication, SIGNAL(clicked()), this, SLOT(math\_operations()));

connect(ui->button\_minus, SIGNAL(clicked()), this, SLOT(math\_operations()));

connect(ui->button\_plus, SIGNAL(clicked()), this, SLOT(math\_operations()));

ui->button\_division->setCheckable(true);

ui->button\_multiplication->setCheckable(true);

ui->button\_minus->setCheckable(true);

ui->button\_plus->setCheckable(true);

connect(ui->button\_sin, SIGNAL(clicked()), this, SLOT(operations()));

connect(ui->button\_cos, SIGNAL(clicked()), this, SLOT(operations()));

connect(ui->button\_tan, SIGNAL(clicked()), this, SLOT(operations()));

connect(ui->button\_cot, SIGNAL(clicked()), this, SLOT(operations()));

connect(ui->button\_arcsin, SIGNAL(clicked()), this, SLOT(operations()));

connect(ui->button\_arccos, SIGNAL(clicked()), this, SLOT(operations()));

connect(ui->button\_arctg, SIGNAL(clicked()), this, SLOT(operations()));

connect(ui->button\_arcctg, SIGNAL(clicked()), this, SLOT(operations()));

}

void ThirdWindow::SetFunction(double result, QString degrees)

{

degrees = QString::number(result, 'g', 5);

//radians = result \* toConvertDegToRad;

//rad = QString::number(radians, 'g', 10);

//degrees += "°";

ui->degree\_result->setText(degrees);

//ui->radians\_result->setText(rad);

double minutes = result \* 60;

double seconds = minutes \* 60;

QString mins = QString::number(minutes, 'g', 5);

QString secs = QString::number(seconds, 'g', 7);

//ui->radians\_result->setText(rad);

//mins += "'";

//secs += "''";

ui->minutes\_result->setText(mins);

ui->seconds\_result->setText(secs);

//ui->degree\_result->setText(ui->degree\_result->text() + tr("°"));

}

ThirdWindow::~ThirdWindow()

{

delete ui;

}

void ThirdWindow::digit\_numbers()

{

QPushButton \*key=(QPushButton \*)sender();

QString degrees;

double all\_numbers;

if ((ui->degree\_result->text() + key->text()).toDouble() >= 361)

return;

if (ui->degree\_result->text().contains(".") && key->text() == "0"){

degrees = ui->degree\_result->text() + key->text();

ui->degree\_result->setText(degrees);

}

else{

all\_numbers = (ui->degree\_result->text() + key->text()).toDouble();

SetFunction(all\_numbers, degrees);

}

}

void ThirdWindow::operations()

{

QPushButton \*key=(QPushButton \*)sender();

QString degrees, rad;

double all\_numbers = (ui->degree\_result->text()).toDouble();

double newDegrees = all\_numbers;

if (key->text() == "+/-"){

all\_numbers \*= -1;

if (all\_numbers < 0){

degrees = QString::number(all\_numbers, 'g', 5);

ui->degree\_result->setText(degrees);

ui->minutes\_result->setText("error");

ui->seconds\_result->setText("error");

}

else{

SetFunction(all\_numbers, degrees);

}

}

if (key->text() == "sin"){

newDegrees = sin(newDegrees \* toConvertDegToRad);

QString sinus;

sinus = QString::number(newDegrees, 'g', 4);

ui->trig\_func->setText(sinus);

}

if (key->text() == "cos"){

if (newDegrees != 90 && newDegrees != -90){

newDegrees = cos(newDegrees \* toConvertDegToRad);

QString cosinus;

cosinus = QString::number(newDegrees, 'g', 5);

ui->trig\_func->setText(cosinus);

}

else{

ui->trig\_func->setText("0");

}

}

if (key->text() == "tg"){

if (all\_numbers != 90 && all\_numbers != -90){

newDegrees = sin(newDegrees \* toConvertDegToRad) / cos(newDegrees \* toConvertDegToRad);

QString tangen = QString::number(newDegrees, 'g', 3);

ui->trig\_func->setText(tangen);

}

else{

QMessageBox::warning(this, "Ошибка вычисления", "Невозможно определить точное значение!");

ui->trig\_func->setText("error");

}

}

if (key->text() == "ctg"){

if (all\_numbers != 0 && all\_numbers != 180){

if (all\_numbers != 90 && all\_numbers != -90){

newDegrees = (cos(newDegrees \* toConvertDegToRad) / sin(newDegrees \* toConvertDegToRad));

QString cot = QString::number(newDegrees, 'g', 4);

ui->trig\_func->setText(cot);

}

else { ui->trig\_func->setText("0"); }

}

else{

QMessageBox::warning(this, "Ошибка вычисления", "Невозможно определить точное значение!");

ui->trig\_func->setText("Error");

}

}

if (key->text() == "arcsin"){

if (all\_numbers >= -90 && all\_numbers <= 90){

newDegrees = asin(newDegrees \* toConvertDegToRad);

QString arcsin = QString::number(newDegrees, 'g', 4);

if (all\_numbers > 57){

QMessageBox::warning(this, "Ошибка вычисления", "Почему-то встроенная функция не считает эту фунцию для углов, больших 57");

}

ui->obr\_trig\_func->setText(arcsin);

}

else{

QMessageBox::warning(this, "Ошибка вычисления", "Невозможно определить точное значение!");

ui->obr\_trig\_func->setText("Error");

}

}

if (key->text() == "arccos"){

if (all\_numbers >= 0 && all\_numbers <= 180){

newDegrees = acos(newDegrees \* toConvertDegToRad);

QString arccos = QString::number(newDegrees, 'g', 4);

if (all\_numbers > 57){

QMessageBox::warning(this, "Ошибка вычисления", "Почему-то встроенная функция не считает эту фунцию для углов, больших 57");

}

ui->obr\_trig\_func->setText(arccos);

}

else{

QMessageBox::warning(this, "Ошибка вычисления", "Невозможно определить точное значение!");

ui->obr\_trig\_func->setText("Error");

}

}

if (key->text()== "arctg"){

if (all\_numbers > -90 && all\_numbers < 90){

newDegrees = atan(newDegrees \* toConvertDegToRad);

QString arctan = QString::number(newDegrees, 'g', 4);

ui->obr\_trig\_func->setText(arctan);

}

else{

QMessageBox::warning(this, "Ошибка вычисления", "Невозможно определить точное значение!");

ui->obr\_trig\_func->setText("Error");

}

}

if (key->text() == "arcctg"){

if (all\_numbers > 0 && all\_numbers < 180){

newDegrees = atan(1 / (newDegrees \* toConvertDegToRad));

QString arccot = QString::number(newDegrees, 'g', 4);

ui->obr\_trig\_func->setText(arccot);

}

else{

QMessageBox::warning(this, "Ошибка вычисления", "Невозможно определить точное значение!");

ui->obr\_trig\_func->setText("Error");

}

}

}

void ThirdWindow::math\_operations()

{

QPushButton \*key=(QPushButton \*)sender();

firstD = ui->degree\_result->text().toDouble();

ui->degree\_result->setText("");

key->setChecked(true);

}

void ThirdWindow::on\_button\_dot\_clicked()

{

if (!(ui->degree\_result->text().contains('.'))){

ui->degree\_result->setText(ui->degree\_result->text() + tr("."));

}

}

void ThirdWindow::on\_button\_equal\_clicked()

{

double result, secondD;

QString degrees;

secondD = ui->degree\_result->text().toDouble();

if (ui->button\_division->isChecked()){

if (secondD != 0){

result = firstD / secondD;

SetFunction(result, degrees);

}else{

QMessageBox::warning(this, "Error", "На нуль лучше не делить");

}

ui->button\_division->setChecked(false);

} else if (ui->button\_multiplication->isChecked()){

result = firstD \* secondD;

SetFunction(result, degrees);

ui->button\_multiplication->setChecked(false);

} else if (ui->button\_minus->isChecked()){

result = firstD - secondD;

SetFunction(result, degrees);

ui->button\_minus->setChecked(false);

} else if (ui->button\_plus->isChecked()){

result = firstD + secondD;

SetFunction(result, degrees);

ui->button\_plus->setChecked(false);

}

}

void ThirdWindow::on\_button\_erase\_all\_clicked()

{

ui->button\_division->setChecked(false);

ui->button\_multiplication->setChecked(false);

ui->button\_minus->setChecked(false);

ui->button\_plus->setChecked(false);

ui->degree\_result->setText("0");

ui->minutes\_result->setText("0");

ui->seconds\_result->setText("0");

ui->trig\_func->setText("0");

ui->obr\_trig\_func->setText("0");

}

void ThirdWindow::on\_button\_clear\_clicked()

{

ui->button\_division->setChecked(false);

ui->button\_multiplication->setChecked(false);

ui->button\_minus->setChecked(false);

ui->button\_plus->setChecked(false);

QString degrees = ui->degree\_result->text();

degrees.chop(1);

double result = degrees.toDouble();

SetFunction(result, degrees);

}

**Анализ решения задачи коммивояжёра**

**1.** Для решения задач необходимо:

**1.1.** Использовать класс MainWindow для отображения окна с матрицей.

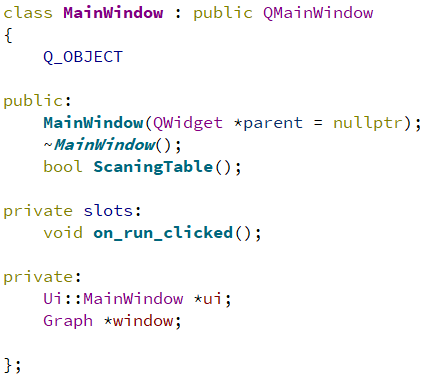
**1.2.** Использовать класс Graph, который будет отвечать за отрисовку связей, узлов, надписей весов на связях, а также за общую работу алгоритма решения задачи коммивояжёра.

**2.** В ходе работы были использованы следующие типы данных:

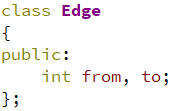
**2.1.** Класс Graph, который будет отвечать за отрисовку связей, узлов, надписей весов на связях, а также за общую работу алгоритма решения задачи коммивояжёра.



**2.2.** Класс MainWindow для отображения окна с матрицей.

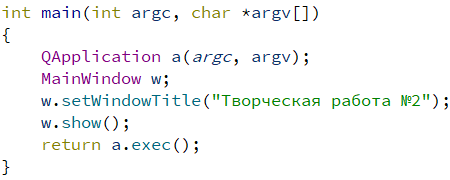


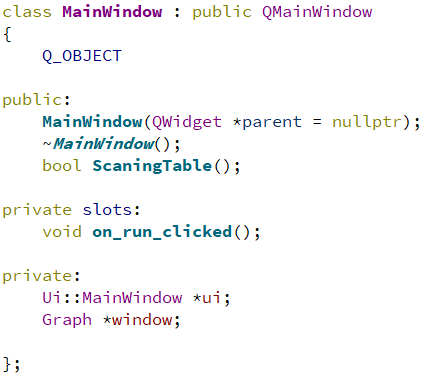
**2.3.** Структура Edge, необходимая для хранения пары вершин оптимального пути.



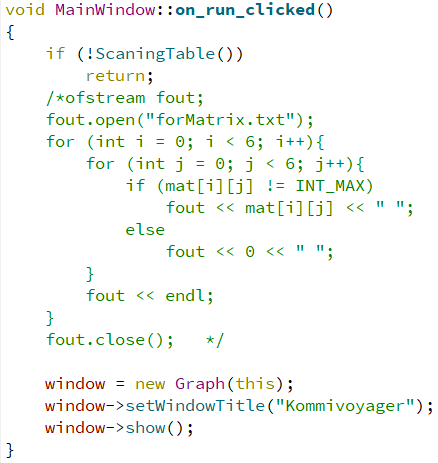
**3.** Для решения задачи данные были представлены в следующем виде:

**3.1.** В функции main() создаётся объект класса MainWindow для отображения титульного окна, в котором должна использоваться матрица смежности для ввода графа с 6 вершинами.





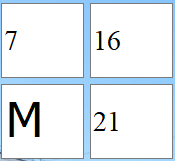
**3.2.** Функция on\_run\_clicked() – событие, отвечающее за нажатие кнопки «Calculate».



Закомментированная область – заполнение введённой матрицы пользователем в текстовый документ с помощью потокового класса ofstream.

**4.** Для операций ввода и вывода использовались следующие операторы и функции:

**4.1.** Ввод данных в матрицу осуществим с помощью множества объектов класса QLineEdit.



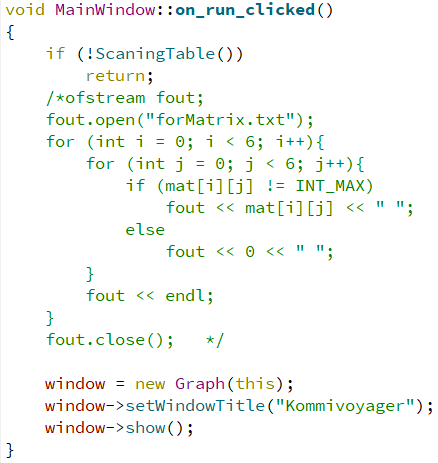
**4.2.** Функция ScanningTable() необходима для проверки матрицы на случай, если хотя бы одно поле не является заполненным. Если матрица полностью заполнена, то возвращается значение 1, иначе – 0.



В переменные класса QString собираются данные с обхектов класса QLineEdit. В случае, если хотя бы одно поле является пустым, метод isEmpty() вернёт значение 1 и функция вернёт значение 0.

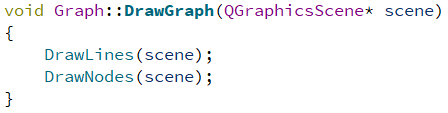
В другие переменные типа int преобразуются значения из класса QString и происходит заполнение матрицы.

**4.3.** При нажатии кнопки Calculate выполняется событие on\_run\_clicked().



Выполняется проверка на значение функции ScanningTable(). В случае, если матрица была заполнена полностью выполняется её копирование в текстовый файл для последующего пользования, а также вызов класса Graph, с новым окном.

**4.4.** Для печати графа в окно используется функция DrawGraph().



Для запуска данной функции необходимо создать объект класса QGraphicsScene, который создаётся в конструкторе класса.

scene = new QGraphicsScene();

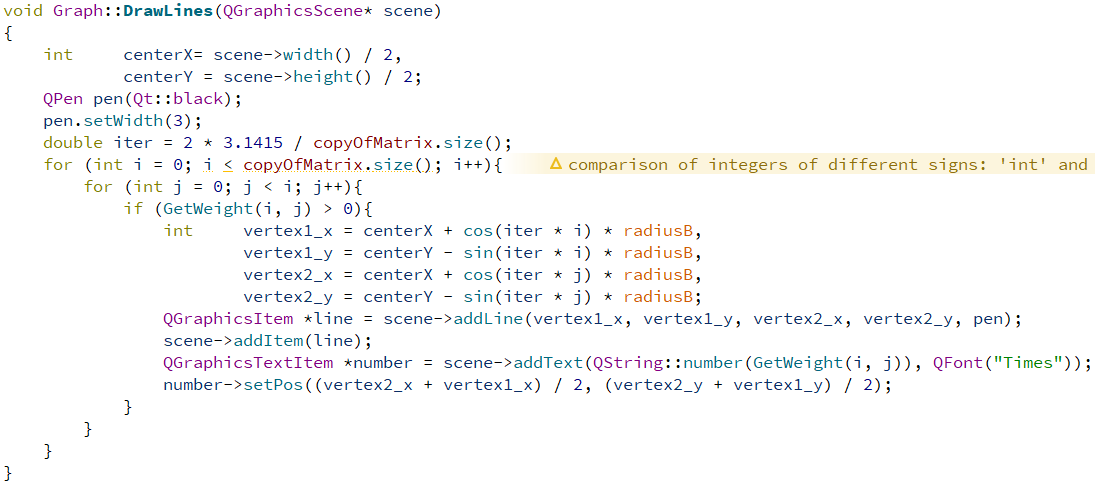
scene->setSceneRect(0, 0, ui->graphicsView->width(), ui->graphicsView->height());

ui->graphicsView->setScene(scene);

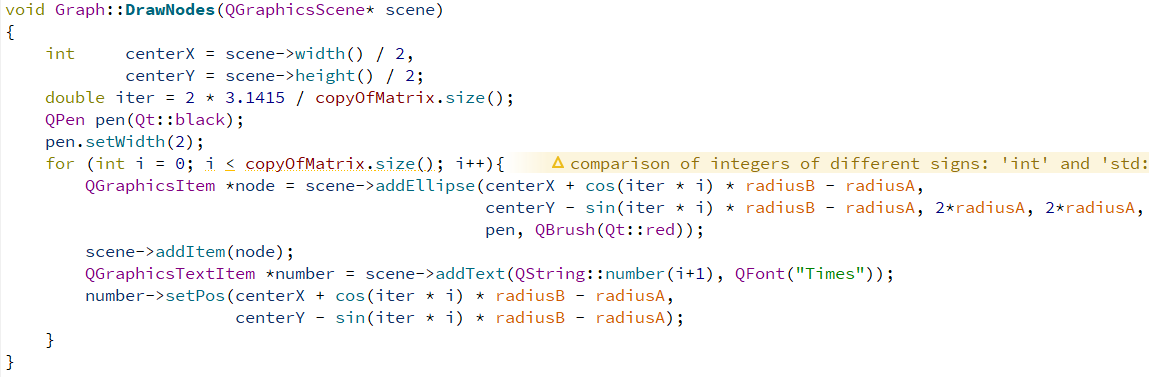
scene->clear();

DrawGraph(scene);

**4.5.** Функция DrawLines() необходима для вычисления координат для отрисовки связей между узлами. Также функция отрисовывает веса рёбер.



**4.6.** Функция DrawNodes() необходима для вычисления координат узлов.



**4.7.** Функция GetWeight() необходима для возвращения веса между указанными узлами. Глобальные переменные отвечают за радиус каждого узла (radiusA) и расстояние узла от центра сцены.

int radiusA = 18,

radiusB = 170;

int Graph::GetWeight(const int vertex1, const int vertex2)

{

if (vertex1 >= 0 && vertex1 < copyOfMatrix.size() && vertex2 >=0 && vertex2 < copyOfMatrix.size())

return copyOfMatrix[vertex1][vertex2];

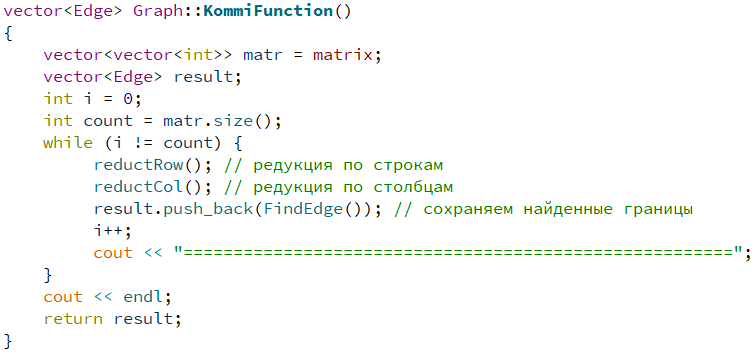
else

return 0;

}

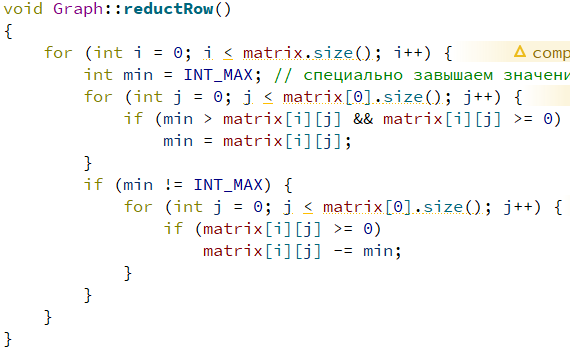
**5.** Поставленные задачи будут решены следующими действиями:

**5.1.** Для решения задачи коммивояжёра организовать функцию KommiFunction(), которая возвращает вектор структур Edge (массив, содержащий пары вершин).

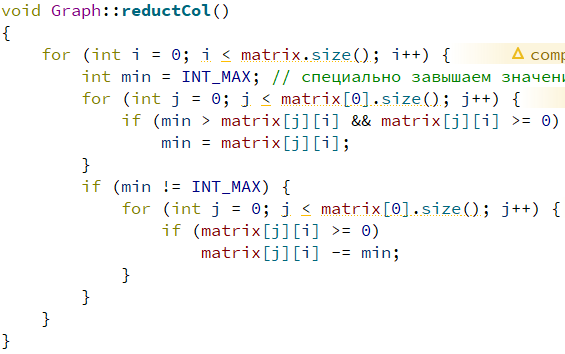


В цикле происходит редукция строк (функция reductRow()), редукция столбцов (функция reductCol()), а затем в вектор типа Edge заносятся вершины из функции FindEdge(). Цикл прекращается, когда матрица сокращается до 1 числа.

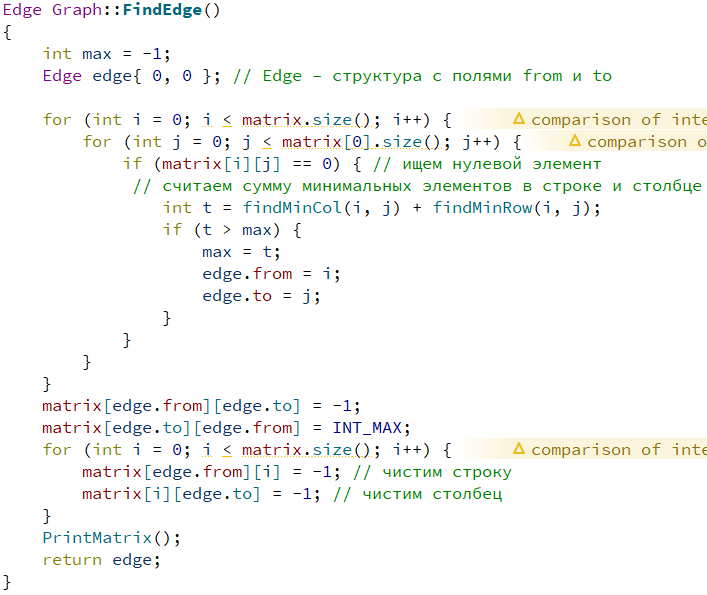
**5.2.** В функции reductRow() происходит редукция строк: находится минимальный элемент в каждой строке матрицы, а за тем вычитание из каждого элемента данной строки этого минимума.



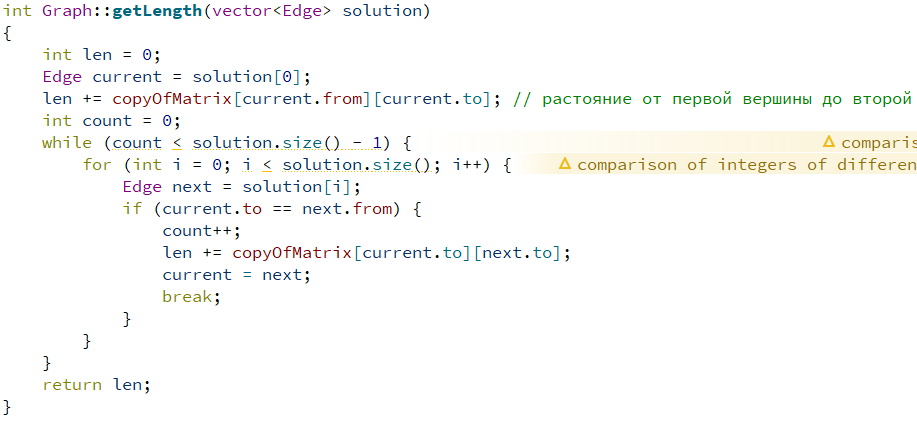
**5.3.** В функции reductCol() происходит редукция столбцов матрицы: находится минимальный элемент в каждом столбце матрицы, а за тем вычитание из каждого элемента данного столбца этого минимума (рисунок 2).



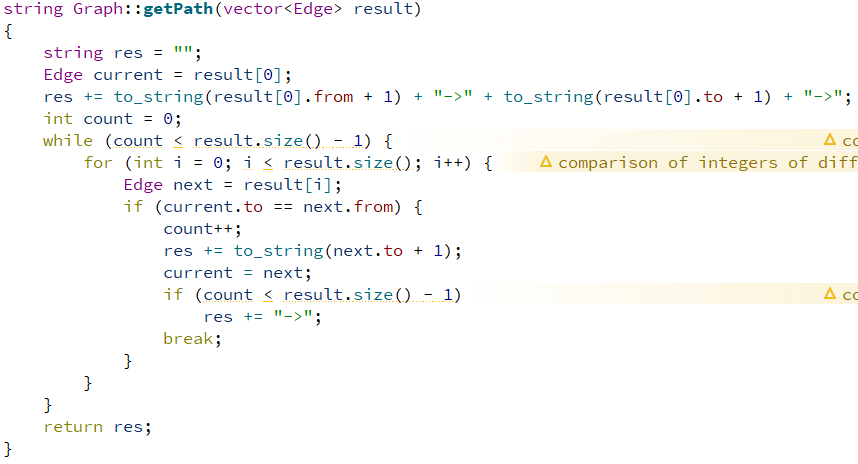
**5.4.** В функции FindEdge() происходит оценка нулевых элементов матрицы, а затем удаление столбцов и строк матрицы с максимальной оценкой, так как в моей реализации происходит включение найденного пути между узлами сразу.



**5.5.** В функции getLength() происходит подсчёт длины пути между вершинами, которые содержатся в векторе структур Edge.



**5.6.** В функции getPath() формируется путь между вершинами так, чтобы он был удобно читаемым.

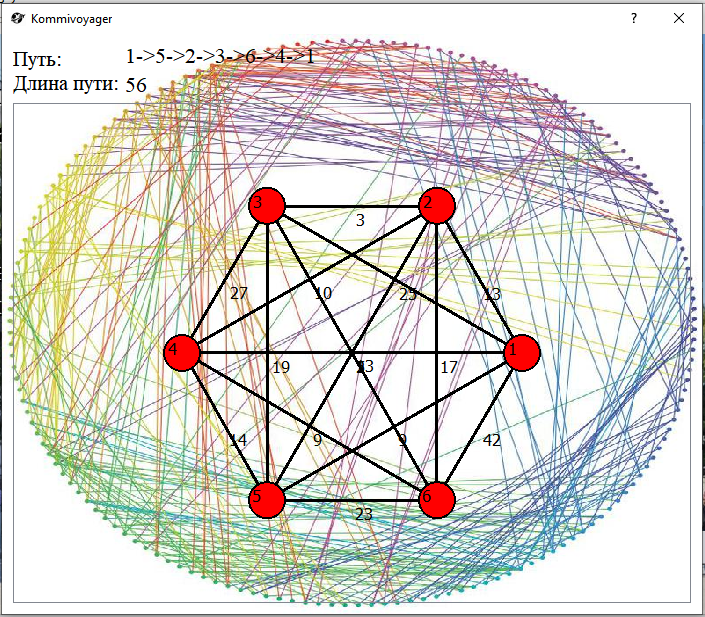


**5.7.** В конструкторе класса Graph происходит вызов функций для формирования вектора вершин, получения пути, получения длины пути для того, чтобы в сцене вывести полученные данные через методы setText() для объектов класса QLabel, а также рисование самого графа.



**Скриншоты**





**Код**

#ifndef EDGE\_H

#define EDGE\_H

class Edge

{

public:

int from, to;

};

#endif // EDGE\_H

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QGraphicsDropShadowEffect>

#include "graph.h"

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

bool ScaningTable();

private slots:

void on\_run\_clicked();

private:

Ui::MainWindow \*ui;

Graph \*window;

};

#endif // MAINWINDOW\_H

#ifndef GRAPH\_H

#define GRAPH\_H

#include <QDialog>

#include "Edge.h"

#include <iostream>

#include <QGraphicsScene>

#include <QGraphicsItem>

#include <QPen>

#include <QBrush>

#include <QFont>

#include <QString>

#include <QGraphicsTextItem>

#include <vector>

#include <string>

#include <fstream>

#include <QCoreApplication>

#include <QFile>

#include <QTextStream>

namespace Ui {

class Graph;

}

class Graph : public QDialog

{

Q\_OBJECT

public:

explicit Graph(QWidget \*parent = nullptr);

Graph(std::vector<std::vector<int>> matr);

~Graph();

void PrintMatrix();

void reductRow();

void reductCol() ;

int findMinCol(int a, int b);

int findMinRow(int a, int b);

int getSize() const;

Edge FindEdge();

std::string getPath(std::vector<Edge> result);

int getLength(std::vector<Edge> solution);

QString way;

std::vector<Edge> KommiFunction();

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

void DrawLines(QGraphicsScene\* scene);

void DrawNodes(QGraphicsScene\* scene);

void DrawGraph(QGraphicsScene\* scene);

private:

Ui::Graph \*ui;

std::vector<std::vector<int>> matrix;

std::vector<std::vector<int>> copyOfMatrix;

QGraphicsScene \*scene;

void ReadFunction();

};

#endif // GRAPH\_H

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.setWindowTitle("Творческая работа №2");

w.show();

return a.exec();

}

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "graph.h"

#include <string>

using namespace std;

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

QPixmap backgr(":/prefix1/pics/backgr.jpeg");

backgr = backgr.scaled(this->size(), Qt::IgnoreAspectRatio);

QPalette palette;

palette.setBrush(QPalette::Background, backgr);

this->setPalette(palette);

QGraphicsDropShadowEffect \*eff = new QGraphicsDropShadowEffect(this);

eff->setOffset(2, 2);

eff->setBlurRadius(0.5);

eff->setColor(Qt::black);

ui->sign->setGraphicsEffect(eff);

QGraphicsDropShadowEffect \*eff2 = new QGraphicsDropShadowEffect(this);

eff2->setOffset(1, 1);

eff2->setBlurRadius(0.5);

eff2->setColor(Qt::black);

ui->date\_year->setGraphicsEffect(eff2);

QGraphicsDropShadowEffect \*eff3 = new QGraphicsDropShadowEffect(this);

eff3->setOffset(2, 2);

eff3->setBlurRadius(1);

eff3->setColor(Qt::white);

ui->title->setGraphicsEffect(eff3);

}

vector<vector<int>> mat;

bool MainWindow::ScaningTable()

{

QString inp1 = ui->sec\_fir->text(), inp2 = ui->th\_fir->text(), inp3 = ui->four\_fir->text(), inp4 = ui->fith\_fir->text(), inp5 = ui->six\_fir->text(),

inp6 = ui->fir\_sec->text(), inp7 = ui->th\_sec->text(), inp8 = ui->four\_sec->text(), inp9 = ui->fith\_sec->text(), inp10 = ui->six\_sec->text(),

inp11 = ui->fir\_th->text(), inp12 = ui->sec\_th->text(), inp13 = ui->four\_th->text(), inp14 = ui->fith\_th->text(), inp15 = ui->six\_th->text(),

inp16 = ui->fir\_four->text(), inp17 = ui->sec\_four->text(), inp18 = ui->th\_four->text(), inp19 = ui->fith\_four->text(), inp20 = ui->six\_four->text(),

inp21 = ui->fir\_fith->text(), inp22 = ui->sec\_fith->text(), inp23 = ui->th\_fith->text(), inp24 = ui->four\_fith->text(), inp25 = ui->six\_fith->text(),

inp26 = ui->fir\_six->text(), inp27 = ui->sec\_six->text(), inp28 = ui->th\_six->text(), inp29 = ui->four\_six->text(), inp30 = ui->fith\_six->text();

if (inp1.isEmpty() || inp2.isEmpty() || inp3.isEmpty() || inp3.isEmpty() || inp4.isEmpty() || inp5.isEmpty() ||

inp6.isEmpty() || inp7.isEmpty() || inp8.isEmpty() || inp9.isEmpty() || inp10.isEmpty() || inp11.isEmpty() ||

inp12.isEmpty() || inp13.isEmpty() || inp14.isEmpty() || inp15.isEmpty() || inp16.isEmpty() || inp17.isEmpty() ||

inp18.isEmpty() || inp19.isEmpty() || inp20.isEmpty() || inp21.isEmpty() || inp22.isEmpty() || inp23.isEmpty() ||

inp24.isEmpty() || inp25.isEmpty() || inp26.isEmpty() || inp27.isEmpty() || inp28.isEmpty() || inp29.isEmpty() || inp30.isEmpty())

return 0;

int in1 = inp1.toInt(), in2 = inp2.toInt(), in3 = inp3.toInt(), in4 = inp4.toInt(), in5 = inp5.toInt(), in6 = inp6.toInt(), in7 = inp7.toInt(),

in8 = inp8.toInt(), in9 = inp9.toInt(), in10 = inp10.toInt(), in11 = inp11.toInt(), in12 = inp12.toInt(), in13 = inp13.toInt(),

in14 = inp14.toInt(), in15 = inp15.toInt(), in16 = inp16.toInt(), in17 = inp17.toInt(), in18 = inp18.toInt(), in19 = inp19.toInt(),

in20 = inp20.toInt(), in21 = inp21.toInt(), in22 = inp22.toInt(), in23 = inp23.toInt(), in24 = inp24.toInt(), in25 = inp25.toInt(),

in26 = inp26.toInt(), in27 = inp27.toInt(), in28 = inp28.toInt(), in29 = inp29.toInt(), in30 = inp30.toInt();

mat = {{ INT\_MAX, in1, in2, in3, in4, in5 },

{ in6, INT\_MAX, in7, in8, in9, in10 },

{ in11, in12, INT\_MAX, in13, in14, in15 },

{ in16, in17, in18, INT\_MAX, in19, in20 },

{ in21, in22, in23, in24, INT\_MAX, in25 },

{ in26, in27, in28, in29, in30, INT\_MAX}};

return 1;

}

MainWindow::~MainWindow()

{

delete ui;

delete window;

}

void MainWindow::on\_run\_clicked()

{

if (!ScaningTable())

return;

/\*ofstream fout;

fout.open("forMatrix.txt");

for (int i = 0; i < 6; i++){

for (int j = 0; j < 6; j++){

if (mat[i][j] != INT\_MAX)

fout << mat[i][j] << " ";

else

fout << 0 << " ";

}

fout << endl;

}

fout.close(); \*/

window = new Graph(this);

window->setWindowTitle("Kommivoyager");

window->show();

}

#include "graph.h"

#include "ui\_graph.h"

#include <cmath>

using namespace std;

/\* CALCULATING COORDS FOR PRINTING \*/

//

int radiusA = 18,

radiusB = 170;

int Graph::GetWeight(const int vertex1, const int vertex2)

{

if (vertex1 >= 0 && vertex1 < copyOfMatrix.size() && vertex2 >=0 && vertex2 < copyOfMatrix.size())

return copyOfMatrix[vertex1][vertex2];

else

return 0;

}

void Graph::DrawLines(QGraphicsScene\* scene)

{

int centerX= scene->width() / 2,

centerY = scene->height() / 2;

QPen pen(Qt::black);

pen.setWidth(3);

double iter = 2 \* 3.1415 / copyOfMatrix.size();

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

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

if (GetWeight(i, j) > 0){

int vertex1\_x = centerX + cos(iter \* i) \* radiusB,

vertex1\_y = centerY - sin(iter \* i) \* radiusB,

vertex2\_x = centerX + cos(iter \* j) \* radiusB,

vertex2\_y = centerY - sin(iter \* j) \* radiusB;

QGraphicsItem \*line = scene->addLine(vertex1\_x, vertex1\_y, vertex2\_x, vertex2\_y, pen);

scene->addItem(line);

QGraphicsTextItem \*number = scene->addText(QString::number(GetWeight(i, j)), QFont("Times"));

number->setPos((vertex2\_x + vertex1\_x) / 2, (vertex2\_y + vertex1\_y) / 2);

}

}

}

}

void Graph::DrawNodes(QGraphicsScene\* scene)

{

int centerX = scene->width() / 2,

centerY = scene->height() / 2;

double iter = 2 \* 3.1415 / copyOfMatrix.size();

QPen pen(Qt::black);

pen.setWidth(2);

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

QGraphicsItem \*node = scene->addEllipse(centerX + cos(iter \* i) \* radiusB - radiusA,

centerY - sin(iter \* i) \* radiusB - radiusA, 2\*radiusA, 2\*radiusA,

pen, QBrush(Qt::red));

scene->addItem(node);

QGraphicsTextItem \*number = scene->addText(QString::number(i+1), QFont("Times"));

number->setPos(centerX + cos(iter \* i) \* radiusB - radiusA,

centerY - sin(iter \* i) \* radiusB - radiusA);

}

}

void Graph::DrawGraph(QGraphicsScene\* scene)

{

DrawLines(scene);

DrawNodes(scene);

}

//

void Graph::PrintMatrix()

{

cout << endl;

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

for (int j = 0; j < matrix[0].size(); j++) {

cout << "\t" <<matrix[i][j] << " ";

}

cout << endl;

}

cout << endl;

}

void Graph::reductRow()

{

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

int min = INT\_MAX; // специально завышаем значение до максимума

for (int j = 0; j < matrix[0].size(); j++) {

if (min > matrix[i][j] && matrix[i][j] >= 0)

min = matrix[i][j];

}

if (min != INT\_MAX) {

for (int j = 0; j < matrix[0].size(); j++) {

if (matrix[i][j] >= 0)

matrix[i][j] -= min;

}

}

}

}

void Graph::reductCol()

{

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

int min = INT\_MAX; // специально завышаем значение до максимума

for (int j = 0; j < matrix[0].size(); j++) {

if (min > matrix[j][i] && matrix[j][i] >= 0)

min = matrix[j][i];

}

if (min != INT\_MAX) {

for (int j = 0; j < matrix[0].size(); j++) {

if (matrix[j][i] >= 0)

matrix[j][i] -= min;

}

}

}

}

int Graph::findMinCol(int a, int b)

{

int min = INT\_MAX;

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

if (min > matrix[i][b] && matrix[i][b] >= 0 && i != a) {

min = matrix[i][b];

}

}

if (min == INT\_MAX) return 0;

return min;

}

int Graph::findMinRow(int a, int b)

{

int min = INT\_MAX;

for (int j = 0; j < matrix[0].size(); j++) {

if (min > matrix[a][j] && matrix[a][j] >= 0 && b != j)

min = matrix[a][j];

}

if (min == INT\_MAX) return 0;

return min;

}

Edge Graph::FindEdge()

{

int max = -1;

Edge edge{ 0, 0 }; // Edge – структура с полями from и to

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

for (int j = 0; j < matrix[0].size(); j++) {

if (matrix[i][j] == 0) { // ищем нулевой элемент

// считаем сумму минимальных элементов в строке и столбце

int t = findMinCol(i, j) + findMinRow(i, j);

if (t > max) {

max = t;

edge.from = i;

edge.to = j;

}

}

}

}

matrix[edge.from][edge.to] = -1;

matrix[edge.to][edge.from] = INT\_MAX;

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

matrix[edge.from][i] = -1; // чистим строку

matrix[i][edge.to] = -1; // чистим столбец

}

PrintMatrix();

return edge;

}

string Graph::getPath(vector<Edge> result)

{

string res = "";

Edge current = result[0];

res += to\_string(result[0].from + 1) + "->" + to\_string(result[0].to + 1) + "->";

int count = 0;

while (count < result.size() - 1) {

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

Edge next = result[i];

if (current.to == next.from) {

count++;

res += to\_string(next.to + 1);

current = next;

if (count < result.size() - 1)

res += "->";

break;

}

}

}

return res;

}

int Graph::getLength(vector<Edge> solution)

{

int len = 0;

Edge current = solution[0];

len += copyOfMatrix[current.from][current.to]; // растояние от первой вершины до второй

int count = 0;

while (count < solution.size() - 1) {

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

Edge next = solution[i];

if (current.to == next.from) {

count++;

len += copyOfMatrix[current.to][next.to];

current = next;

break;

}

}

}

return len;

}

void Graph::ReadFunction()

{

string row, tmpRow;

int j = 0, k = 0;

ifstream fin;

fin.open("forMatrix.txt");

while (getline(fin, row)){

for (int i = 0; i < row.length(); i++){

if (row[i] != ' ')

tmpRow += row[i];

else{

copyOfMatrix[j][k] = stoi(tmpRow);

k++;

}

}

j++;

}

fin.close();

}

vector<Edge> Graph::KommiFunction()

{

vector<vector<int>> matr = matrix;

vector<Edge> result;

int i = 0;

int count = matr.size();

while (i != count) {

reductRow(); // редукция по строкам

reductCol(); // редукция по cтолбцам

result.push\_back(FindEdge()); // сохраняем найденные границы

i++;

cout << "=======================================================";

}

cout << endl;

return result;

}

Graph::Graph(QWidget \*parent) :

QDialog(parent),

ui(new Ui::Graph)

{

ui->setupUi(this);

copyOfMatrix = { {0, 7, 16, 21, 2, 17 },

{13, 0, 21, 15, 43, 23 },

{ 25, 3, 0, 31, 17, 9 },

{ 13, 10, 27, 0, 33, 12 },

{ 9, 2, 19, 14, 0, 51 },

{ 42, 17, 5, 9, 23, 0 } };

matrix = { {INT\_MAX, 7, 16, 21, 2, 17 },

{13, INT\_MAX, 21, 15, 43, 23 },

{ 25, 3, INT\_MAX, 31, 17, 9 },

{ 13, 10, 27, INT\_MAX, 33, 12 },

{ 9, 2, 19, 14, INT\_MAX, 51 },

{ 42, 17, 5, 9, 23, INT\_MAX } };

vector<Edge> paths = KommiFunction();

string path = getPath(paths);

int len = getLength(paths);

QPixmap backgR(":/prefix1/pics/188f46b234ec79b3823b31ac77c0fa52.jpg");

backgR = backgR.scaled(this->size(), Qt::IgnoreAspectRatio);

QPalette palette;

palette.setBrush(QPalette::Background, backgR);

this->setPalette(palette);

scene = new QGraphicsScene();

scene->setSceneRect(0, 0, ui->graphicsView->width(), ui->graphicsView->height());

ui->graphicsView->setScene(scene);

scene->clear();

QString p = QString::fromStdString(path);

QString l = QString::number(len);

ui->len->setText(l);

ui->path->setText(p);

DrawGraph(scene);

}

Graph::~Graph()

{

delete ui;

delete scene;

}