Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

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

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по творческой работе

Тема: «Решение задачи Коммивояжера и АРМ Трейдера»

Выполнили

Студент группы РИС-22-2б

Максимов А.А.

Проверил доц. Кафедры ИТАС

Полякова О.А.

Пермь 2023

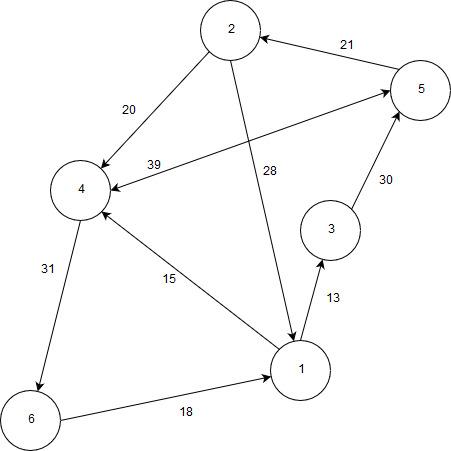
# Постановка задачи

Визуализировать решение задачи Коммивояжера с помощью языка программирования C++.

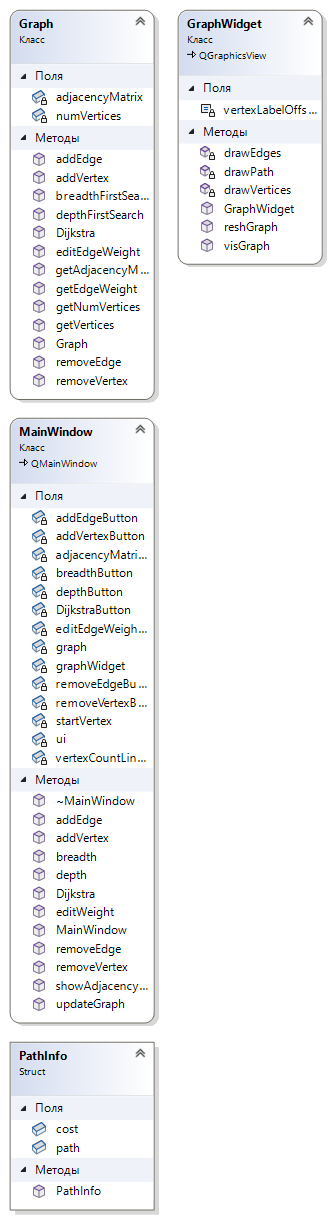
ВАРИАНТ 22:

Реализовать граф, построив все необходимые матрицы.

Выполнение начать с вершины 5.



UML – диаграмма



Используемые инструменты.

Для реализации решения задачи Коммивояжера был использован язык программирования C++, а также фреймворк Qt, с собственной средой разработки Qt Creator. Для записи видео с экрана компьютера была использована программа OBS.

Библиотеки.

Для визуализации решения задачи Коммивояжера были использованы следующие классы, входящие в набор разработчика Qt: QMainWindow, QLineEdit, QMessageBox, QString, QObject, QWidget, QGraphicsView, QGraphicsScene, QGraphicsItem, QGraphicsLineItem, QPen, QBrush, QColor, QPointF, QRectF, QGraphicsTextItem

Код программы

**graph.h**

#ifndef GRAPH\_H

#define GRAPH\_H

#include <vector>

class Graph {

private:

int numVertices;

std::vector<std::vector<int>> adjacencyMatrix;

public:

Graph(int vertices);

void addVertex();

void removeVertex(int vertex);

void removeEdge(int v1, int v2);

void editEdgeWeight(int v1, int v2, int weight);

int getNumVertices() const;

int getEdgeWeight(int v1, int v2) const;

void addEdge(int v1, int v2, int weight);

std::vector<int> getVertices() const;

std::vector<std::vector<int>> getAdjacencyMatrix() const;

};

#endif // GRAPH\_H

**graph.cpp**

#include "graph.h"

#include <QMessageBox>

using namespace std;

Graph::Graph(int vertices) {

numVertices = vertices;

adjacencyMatrix.resize(numVertices, vector<int>(numVertices, 0));

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

adjacencyMatrix[i][i] = 0;

}

}

int Graph::getNumVertices() const {

return numVertices;

}

int Graph::getEdgeWeight(int v1, int v2) const {

return adjacencyMatrix[v1][v2];

}

void Graph::addEdge(int v1, int v2, int weight) {

adjacencyMatrix[v1][v2] = weight;

}

vector<int> Graph::getVertices() const {

vector<int> vertices(numVertices);

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

vertices[i] = i;

}

return vertices;

}

void Graph::addVertex() {

numVertices++;

adjacencyMatrix.resize(numVertices, vector<int>(numVertices, 0));

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

adjacencyMatrix[i].resize(numVertices);

}

}

void Graph::removeVertex(int vertex) {

if (vertex < 0 || vertex >= numVertices) {

QMessageBox::critical(nullptr, "Ошибка", "Неверный номер вершины");

return;

}

numVertices--;

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

adjacencyMatrix[i].erase(adjacencyMatrix[i].begin() + vertex);

}

adjacencyMatrix.erase(adjacencyMatrix.begin() + vertex);

}

vector<std::vector<int>> Graph::getAdjacencyMatrix() const {

return adjacencyMatrix;

}

void Graph::removeEdge(int v1, int v2) {

if (v1 < 0 || v1 >= numVertices || v2 < 0 || v2 >= numVertices) {

QMessageBox::critical(nullptr, "Ошибка", "Неверные номера вершин");

return;

}

adjacencyMatrix[v1][v2] = 0;

adjacencyMatrix[v2][v1] = 0;

}

void Graph::editEdgeWeight(int v1, int v2, int weight) {

if (v1 < 0 || v1 >= numVertices || v2 < 0 || v2 >= numVertices) {

QMessageBox::critical(nullptr, "Ошибка", "Неверные номера вершин");

return;

}

adjacencyMatrix[v1][v2] = weight;

}

**graphwidget.h**

#ifndef GRAPHWIDGET\_H

#define GRAPHWIDGET\_H

#include <QMainWindow>

#include <QObject>

#include <QWidget>

#include <QGraphicsView>

#include <QGraphicsScene>

#include <QGraphicsItem>

#include <QGraphicsLineItem>

#include <QPen>

#include <QBrush>

#include <QColor>

#include <QPointF>

#include <QRectF>

#include <QGraphicsTextItem>

#include "graph.h"

using namespace std;

struct PathInfo {

std::vector<int> path;

int cost;

PathInfo(const std::vector<int>& \_path, int \_cost) : path(\_path), cost(\_cost) {}

};

class GraphWidget : public QGraphicsView

{

Q\_OBJECT

public:

GraphWidget(QWidget\* parent = nullptr) : QGraphicsView(parent) {

setRenderHint(QPainter::Antialiasing);

setScene(new QGraphicsScene(this));

}

void reshGraph(const Graph& graph, const PathInfo& optimalPath);

void visGraph(const Graph& graph);

private:

void drawEdges(const Graph& graph, const vector<QPointF>& vertexPositions);

void drawVertices(const vector<QPointF>& vertexPositions);

void drawPath(const vector<int>& path, const vector<QPointF>& vertexPositions);

const int vertexLabelOffset = 5;

};

#endif // GRAPHWIDGET\_H

**graphwidget.cpp**

#include "graph.h"

#include "graphwidget.h"

#include <QGraphicsView>

#include <QGraphicsScene>

#include <QGraphicsItem>

#include <QGraphicsLineItem>

#include <QPen>

#include <QBrush>

#include <QColor>

#include <QPointF>

#include <QRectF>

void GraphWidget::reshGraph(const Graph& graph, const PathInfo& optimalPath) {

scene()->clear();

int numVertices = graph.getNumVertices();

const int sceneWidth = 1000;

const int sceneHeight = 1000;

const QPointF center(sceneWidth / 2, sceneHeight / 2);

const qreal radius = qMin(sceneWidth, sceneHeight) \* 0.4;

const qreal angleIncrement = 2 \* M\_PI / numVertices;

std::vector<QPointF> vertexPositions;

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

qreal angle = i \* angleIncrement;

qreal x = center.x() + radius \* qCos(angle);

qreal y = center.y() + radius \* qSin(angle);

vertexPositions.emplace\_back(x, y);

}

drawEdges(graph, vertexPositions);

drawVertices(vertexPositions);

drawPath(optimalPath.path, vertexPositions);

}

void GraphWidget::visGraph(const Graph& graph)

{

scene()->clear();

int numVertices = graph.getNumVertices();

const int sceneWidth = 1000;

const int sceneHeight = 1000;

const QPointF center(sceneWidth / 2, sceneHeight / 2);

const qreal radius = qMin(sceneWidth, sceneHeight) \* 0.4;

const qreal angleIncrement = 2 \* M\_PI / numVertices;

std::vector<QPointF> vertexPositions;

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

qreal angle = i \* angleIncrement;

qreal x = center.x() + radius \* qCos(angle);

qreal y = center.y() + radius \* qSin(angle);

vertexPositions.emplace\_back(x, y);

}

drawEdges(graph, vertexPositions);

drawVertices(vertexPositions);

}

void GraphWidget::drawEdges(const Graph& graph, const std::vector<QPointF>& vertexPositions) {

const int numVertices = graph.getNumVertices();

const QPen edgePen(Qt::black);

for (int v1 = 0; v1 < numVertices; v1++) {

for (int v2 = 0; v2 < numVertices; v2++) {

int weight = graph.getEdgeWeight(v1, v2);

if (weight > 0) {

QPointF p1 = vertexPositions[v1];

QPointF p2 = vertexPositions[v2];

scene()->addLine(p1.x(), p1.y(), p2.x(), p2.y(), edgePen);

}

}

}

}

void GraphWidget::drawVertices(const std::vector<QPointF>& vertexPositions) {

const int numVertices = vertexPositions.size();

const int vertexRadius = 10;

const QPen vertexPen(Qt::black);

const QBrush vertexBrush(Qt::yellow);

for (int v = 0; v < numVertices; v++) {

QPointF position = vertexPositions[v];

QRectF rect(position.x() - vertexRadius, position.y() - vertexRadius,

2 \* vertexRadius, 2 \* vertexRadius);

scene()->addEllipse(rect, vertexPen, vertexBrush);

QGraphicsTextItem\* label = scene()->addText(QString::number(v));

label->setPos(position.x() - vertexLabelOffset, position.y() - vertexLabelOffset);

}

}

void GraphWidget::drawPath(const std::vector<int>& path, const std::vector<QPointF>& vertexPositions) {

if (path.empty())

return;

const int pathPenWidth = 2;

const QColor pathColor = Qt::red;

const QPen pathPen(pathColor, pathPenWidth);

const int numVertices = path.size();

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

int v1 = path[i];

int v2 = path[i + 1];

QPointF p1 = vertexPositions[v1];

QPointF p2 = vertexPositions[v2];

scene()->addLine(p1.x(), p1.y(), p2.x(), p2.y(), pathPen);

}

int v1 = path.back();

int v2 = path.front();

QPointF p1 = vertexPositions[v1];

QPointF p2 = vertexPositions[v2];

scene()->addLine(p1.x(), p1.y(), p2.x(), p2.y(), pathPen);

}

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "graphwidget.h"

#include "graph.h"

#include "qpushbutton.h"

#include <QLineEdit>

#include <QMessageBox>

#include <QString>

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 onVertexCountChanged(const QString& text);

void updateGraph(int vertexCount);

void showAdjacencyMatrix();

void addVertex();

void resh();

void addEdge();

void removeEdge();

void removeVertex();

void editWeight();

private:

Ui::MainWindow \*ui;

GraphWidget \*graphWidget;

Graph graph;

int startVertex;

QPushButton\* reshButton;

QLineEdit\* vertexCountLineEdit;

QPushButton\* adjacencyMatrixButton;

QPushButton\* addVertexButton;

QPushButton\* addEdgeButton;

QPushButton\* removeEdgeButton;

QPushButton\* removeVertexButton;

QPushButton\* editEdgeWeightButton;

};

#endif // MAINWINDOW\_H

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <iostream>

#include <queue>

#include <QMessageBox>

#include <QLineEdit>

#include <QIntValidator>

#include <QInputDialog>

#include <QVBoxLayout>

using namespace std;

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow), graph(0)

{

ui->setupUi(this);

graphWidget = new GraphWidget(this);

reshButton = new QPushButton("Решение", this);

connect(reshButton, &QPushButton::clicked, this, &MainWindow::resh);

vertexCountLineEdit = new QLineEdit(this);

vertexCountLineEdit->setGeometry(100,100,300,100);

vertexCountLineEdit->setPlaceholderText("Число вершин");

vertexCountLineEdit->setValidator(new QIntValidator(2, 100, this));

connect(vertexCountLineEdit, &QLineEdit::textChanged, this, &MainWindow::onVertexCountChanged);

adjacencyMatrixButton = new QPushButton("Матрица смежности",this);

connect(adjacencyMatrixButton, &QPushButton::clicked, this, &MainWindow::showAdjacencyMatrix);

addVertexButton = new QPushButton("Добавить вершину", this);

connect(addVertexButton, &QPushButton::clicked, this, &MainWindow::addVertex);

addEdgeButton = new QPushButton("Добавить ребро", this);

connect(addEdgeButton, &QPushButton::clicked, this, &MainWindow::addEdge);

removeEdgeButton = new QPushButton("Удалить ребро", this);

connect(removeEdgeButton, &QPushButton::clicked, this, &MainWindow::removeEdge);

removeVertexButton = new QPushButton("Удалить вершину", this);

connect(removeVertexButton, &QPushButton::clicked, this, &MainWindow::removeVertex);

editEdgeWeightButton = new QPushButton("Изменить вес ребра", this);

connect(editEdgeWeightButton, &QPushButton::clicked, this, &MainWindow::editWeight);

QVBoxLayout \*layout = new QVBoxLayout;

layout->addWidget(vertexCountLineEdit);

layout->addWidget(addVertexButton);

layout->addWidget(removeVertexButton);

layout->addWidget(addEdgeButton);

layout->addWidget(removeEdgeButton);

layout->addWidget(editEdgeWeightButton);

layout->addWidget(graphWidget);

layout->addWidget(reshButton);

layout->addWidget(adjacencyMatrixButton);

QWidget \*widget = new QWidget(this);

widget->setLayout(layout);

setCentralWidget(widget);

}

MainWindow::~MainWindow()

{

delete ui;

}

const int INF = numeric\_limits<int>::max();

int findMin(const vector<int>& arr) {

int minVal = INF;

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

if (arr[i] < minVal) {

minVal = arr[i];

}

}

return minVal;

}

int findMinIndex(const vector<int>& arr) {

int minIndex = -1;

int minVal = INF;

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

if (arr[i] < minVal) {

minVal = arr[i];

minIndex = i;

}

}

return minIndex;

}

PathInfo solveTSP(const Graph& graph, int startVertex) {

int numVertices = graph.getNumVertices();

priority\_queue<PathInfo, vector<PathInfo>, function<bool(const PathInfo&, const PathInfo&)>> pq(

[](const PathInfo& p1, const PathInfo& p2) {

return p1.cost > p2.cost;

}

);

vector<int> path;

path.push\_back(startVertex);

PathInfo initialPath(path, 0);

pq.push(initialPath);

int lowerBound = INF;

while (!pq.empty()) {

PathInfo currentPath = pq.top();

pq.pop();

int currentVertex = currentPath.path.back();

int currentCost = currentPath.cost;

if (currentPath.path.size() == numVertices) {

int finalCost = currentCost + graph.getEdgeWeight(currentVertex, startVertex);

return PathInfo(currentPath.path, finalCost);

}

vector<int> remainingVertices = graph.getVertices();

for (int vertex : currentPath.path) {

remainingVertices.erase(remove(remainingVertices.begin(), remainingVertices.end(), vertex), remainingVertices.end());

}

for (int nextVertex : remainingVertices) {

int edgeWeight = graph.getEdgeWeight(currentVertex, nextVertex);

if (edgeWeight == 0)

continue;

int nextCost = currentCost + edgeWeight;

int minRemainingEdgeWeight = INF;

for (int vertex : remainingVertices) {

int remainingEdgeWeight = graph.getEdgeWeight(nextVertex, vertex);

if (remainingEdgeWeight < minRemainingEdgeWeight)

minRemainingEdgeWeight = remainingEdgeWeight;

}

int lowerBoundCost = nextCost + minRemainingEdgeWeight;

if (lowerBoundCost >= lowerBound)

continue;

vector<int> newPath = currentPath.path;

newPath.push\_back(nextVertex);

PathInfo newPathInfo(newPath, nextCost);

pq.push(newPathInfo);

}

}

return PathInfo({}, INF);

}

void MainWindow::resh()

{

int vertexCount = graph.getNumVertices();

startVertex = QInputDialog::getInt(this, "Стартовая вершина", "Введите номер вершины", 0, 0, vertexCount - 1, 1);

PathInfo optimalPath = solveTSP(graph, startVertex);

if (!optimalPath.path.empty()) {

QString optimalPathString = "Оптимальный путь: ";

for (int vertex : optimalPath.path) {

optimalPathString += QString::number(vertex) + " ";

}

optimalPathString += QString::number(startVertex);

QString costString = "Стоимость: " + QString::number(optimalPath.cost);

QMessageBox::information(this, "Результаты", optimalPathString + "\n" + costString);

graphWidget->reshGraph(graph, optimalPath);

}

else {

QMessageBox::information(this, "Результаты", "Не удалось найти оптимальный путь.");

}

}

void MainWindow::onVertexCountChanged(const QString& text)

{

bool ok;

int vertexCount = text.toInt(&ok);

if (ok) {

updateGraph(vertexCount);

} else {

QMessageBox::warning(this, "Ошибка", "Пожалуйста, введите целое число вершин.");

}

}

void MainWindow::updateGraph(int vertexCount)

{

graph = Graph(vertexCount);

if (vertexCount < 2) {

QMessageBox::warning(this, "Ошибка", "Число вершин должно быть не менее 2.");

return;

}

bool ok = true;

while(ok){

int startikVertex = QInputDialog::getInt(this, "Начальная вершина", "Введите номер начальной вершины для ребра", 0, 0, vertexCount - 1, 1, &ok);

int endVertex = QInputDialog::getInt(this, "Конечная вершина", "Введите номер конечной вершины для ребра", 0, 0, vertexCount - 1, 1, &ok);

int weight = QInputDialog::getInt(this, "Вес ребра", "Введите вес ребра", 0, 0, std::numeric\_limits<int>::max(), 1, &ok);

graph.addEdge(startikVertex, endVertex, weight);

QMessageBox::StandardButton reply = QMessageBox::question(this, "Создание ребра", "Хотите создать еще одно ребро?", QMessageBox::Yes | QMessageBox::No);

if (reply == QMessageBox::No) {

ok = false;

}

}

graphWidget->visGraph(graph);

}

void MainWindow::showAdjacencyMatrix()

{

int numVertices = graph.getNumVertices();

std::vector<std::vector<int>> adjacencyMatrix = graph.getAdjacencyMatrix();

QString matrixString;

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

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

matrixString += QString::number(adjacencyMatrix[i][j]) + "\t";

}

matrixString += "\n";

}

QMessageBox::information(this, "Матрица смежности", matrixString);

}

void MainWindow::addVertex() {

graph.addVertex();

updateGraph(graph.getNumVertices());

graphWidget->visGraph(graph);

}

void MainWindow::addEdge()

{

int numVertices = graph.getNumVertices();

int startikVertex = QInputDialog::getInt(this, "Начальная вершина", "Введите номер начальной вершины для ребра", 0, 0, numVertices - 1, 1);

int endVertex = QInputDialog::getInt(this, "Конечная вершина", "Введите номер конечной вершины для ребра", 0, 0, numVertices - 1, 1);

int weight = QInputDialog::getInt(this, "Вес ребра", "Введите вес ребра", 0, 0, std::numeric\_limits<int>::max(), 1);

graph.addEdge(startikVertex, endVertex, weight);

graphWidget->visGraph(graph);

}

void MainWindow::removeEdge()

{

bool ok = true;

int numVertices = graph.getNumVertices();

while(ok){

int startikVertex = QInputDialog::getInt(this, "Начальная вершина", "Введите номер начальной вершины для ребра", 0, 0, numVertices - 1, 1, &ok);

int endVertex = QInputDialog::getInt(this, "Конечная вершина", "Введите номер конечной вершины для ребра", 0, 0, numVertices - 1, 1, &ok);

graph.removeEdge(startikVertex, endVertex);

QMessageBox::StandardButton reply = QMessageBox::question(this, "Удаление ребра", "Хотите удалить еще одно ребро?", QMessageBox::Yes | QMessageBox::No);

if (reply == QMessageBox::No) {

ok = false;

}

}

graphWidget->visGraph(graph);

}

void MainWindow::removeVertex()

{

int numVertices = graph.getNumVertices();

int Vertex = QInputDialog::getInt(this, "Вершина", "Введите номер удаляемой вершины", 0, 0, numVertices - 1, 1);

graph.removeVertex(Vertex);

graphWidget->visGraph(graph);

}

void MainWindow::editWeight()

{

int numVertices = graph.getNumVertices();

int startikVertex = QInputDialog::getInt(this, "Начальная вершина", "Введите номер начальной вершины для ребра", 0, 0, numVertices - 1, 1);

int endVertex = QInputDialog::getInt(this, "Конечная вершина", "Введите номер конечной вершины для ребра", 0, 0, numVertices - 1, 1);

int weight = QInputDialog::getInt(this, "Новый вес ребра", "Введите новый вес ребра", 0, 0, std::numeric\_limits<int>::max(), 1);

graph.editEdgeWeight(startikVertex, endVertex, weight);

graphWidget->visGraph(graph);

}

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

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

QApplication a(argc, argv);

MainWindow w;

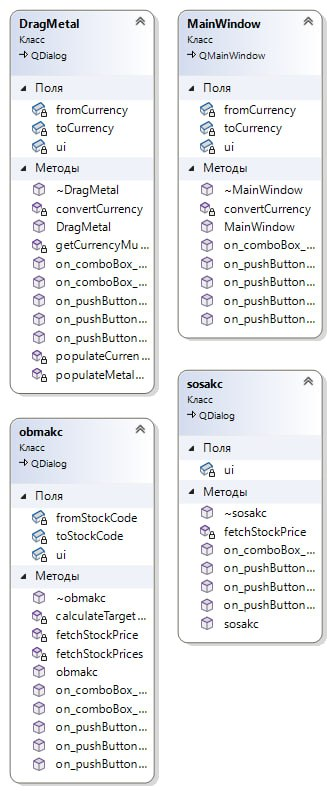
w.show();

return a.exec();

}

**АРМ Трейдера**

UML – диаграмма

****

Используемые инструменты.

Для реализации АРМ Трейдера был использован язык программирования C++, а также фреймворк Qt, с собственной средой разработки Qt Creator. Для записи видео с экрана компьютера была использована программа OBS.

Библиотеки.

Для реализации АРМ Трейдера были задействованы следующие API:  
API Центробанка России - <https://www.cbr-xml-daily.ru/daily_json.js>  
API Биржи Альфавантаж - https://www.alphavantage.co

Для визуализации АРМ Трейдера были использованы следующие классы, входящие в набор разработчика Qt: QMainWindow, QLineEdit, QMessageBox, QString, QObject, QWidget, QGraphicsView, QGraphicsScene, QGraphicsItem, QGraphicsLineItem, QPen, QBrush, QColor, QPointF, QRectF, QGraphicsTextItem

Код программы

**dragmetal.h**

#ifndef DRAGMETAL\_H

#define DRAGMETAL\_H

#include <QDialog>

namespace Ui {

class DragMetal;

}

class DragMetal : public QDialog

{

Q\_OBJECT

public:

explicit DragMetal(QWidget \*parent = nullptr);

~DragMetal();

private slots:

void on\_pushButton\_4\_clicked();

void on\_comboBox\_currentIndexChanged(int index);

void on\_comboBox\_2\_currentIndexChanged(int index);

void on\_pushButton\_11\_clicked();

void on\_pushButton\_9\_clicked();

void on\_pushButton\_10\_clicked();

private:

Ui::DragMetal \*ui;

QString fromCurrency;

QString toCurrency;

double convertCurrency(const QString& fromCurrency, const QString& toCurrency);

void populateMetalComboBox();

void populateCurrencyComboBox();

double getCurrencyMultiplier();

};

#endif // DRAGMETAL\_H

**dragmetal.cpp**

#include "dragmetal.h"

#include "qevent.h"

#include "ui\_dragmetal.h"

#include <mainwindow.h>

#include "QString"

#include "qeventloop.h"

#include <QNetworkAccessManager>

#include <QNetworkReply>

#include <QJsonDocument>

#include <QJsonObject>

#include <QJsonArray>

#include <QUrlQuery>

#include <QTextEdit>

#include <QDomDocument>

#include <obmakc.h>

#include <sosakc.h>

DragMetal::DragMetal(QWidget \*parent) :

QDialog(parent),

ui(new Ui::DragMetal)

{

ui->setupUi(this);

populateMetalComboBox();

fromCurrency = ui->comboBox->currentText();

toCurrency = ui->comboBox\_2->currentData().toString();

}

DragMetal::~DragMetal()

{

delete ui;

}

void DragMetal::on\_pushButton\_4\_clicked()

{

close();

MainWindow\* mainWindow = new MainWindow();

mainWindow->show();

}

void DragMetal::populateCurrencyComboBox()

{

// Assuming you have a list of currency names and codes

QStringList currencyNames = { "RUB", "EUR", "USD" };

// Clear the combo box

ui->comboBox->clear();

// Populate the combo box with currency names

ui->comboBox->addItems(currencyNames);

}

void DragMetal::populateMetalComboBox()

{

// Assuming you have a list of metal names and codes

QStringList metalNames = { "Gold", "Silver", "Platinum", "Palladium" };

QStringList metalCodes = { "1", "2", "3", "4" };

// Clear the combo box

ui->comboBox\_2->clear();

// Populate the combo box with metal names

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

QString metalName = metalNames.at(i);

QString metalCode = metalCodes.at(i);

ui->comboBox\_2->addItem(metalName, metalCode); // Set the metal code as the data for each item

}

}

void DragMetal::on\_comboBox\_currentIndexChanged(int /\*text\*/)

{

fromCurrency = ui->comboBox->currentText();

qDebug() << "Selected From Currency:" << fromCurrency;

}

void DragMetal::on\_comboBox\_2\_currentIndexChanged(int /\*index\*/)

{

// Update the toCurrency value when the metal selection changes

toCurrency = ui->comboBox\_2->currentData().toString();

}

double getCurrencyMultiplier(const QString& currencyCode)

{

QNetworkAccessManager manager;

QUrl url("https://www.cbr-xml-daily.ru/daily\_json.js"); // URL API Центробанка

QNetworkRequest request(url);

QNetworkReply\* reply = manager.get(request);

QEventLoop loop;

DragMetal::connect(reply, &QNetworkReply::finished, &loop, &QEventLoop::quit);

loop.exec();

if (reply->error() != QNetworkReply::NoError) {

// Обработка ошибки запроса

qDebug() << "Error: " << reply->errorString();

return 1.0; // Возвращаем 1.0 в случае ошибки

}

QByteArray responseData = reply->readAll();

QJsonDocument jsonDocument = QJsonDocument::fromJson(responseData);

if (jsonDocument.isNull()) {

// Обработка ошибки парсинга JSON

qDebug() << "Error: Failed to parse JSON response";

return 1.0; // Возвращаем 1.0 в случае ошибки

}

QJsonObject jsonObject = jsonDocument.object();

QJsonObject valuteObject = jsonObject["Valute"].toObject();

double currencyMultiplier = 1.0;

if (valuteObject.contains(currencyCode)) {

QJsonObject currencyObject = valuteObject[currencyCode].toObject();

currencyMultiplier = currencyObject["Value"].toDouble();

}

else {

qDebug() << "Invalid currency code:" << currencyCode;

}

return currencyMultiplier;

}

double DragMetal::convertCurrency(const QString& fromCurrency, const QString& toCurrency)

{

bool isMetal = false;

// Проверка, является ли toCurrency кодом металла

if (toCurrency.toInt(&isMetal) && isMetal) {

QNetworkAccessManager manager;

QUrl url("https://www.cbr.ru/scripts/xml\_metall.asp?date\_req1=27/05/2023&date\_req2=27/05/2023"); // URL API для получения курса металлов

QNetworkRequest request(url);

QNetworkReply\* reply = manager.get(request);

QEventLoop loop;

connect(reply, &QNetworkReply::finished, &loop, &QEventLoop::quit);

loop.exec();

if (reply->error() != QNetworkReply::NoError) {

// Обработка ошибки запроса

qDebug() << "Error: " << reply->errorString();

return 0.0;

}

QByteArray responseData = reply->readAll();

QString responseDataString = QString::fromUtf8(responseData);

// Обработка XML-ответа

QDomDocument doc;

if (!doc.setContent(responseDataString)) {

// Обработка ошибки парсинга XML

qDebug() << "Error: Failed to parse XML response" << responseData;

return 0.0;

}

QDomElement root = doc.documentElement();

QDomNodeList metalNodes = root.elementsByTagName("Record");

double metalPrice = 0.0;

// Поиск цены металла по его коду

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

QDomElement metalElement = metalNodes.at(i).toElement();

QString metalCode = metalElement.attribute("Code");

if (metalCode == toCurrency) {

QDomElement buyElement = metalElement.firstChildElement("Buy");

QString metalPriceString = buyElement.text().replace(",", ".");

metalPrice = metalPriceString.toDouble();

break;

}

}

if (metalPrice == 0.0) {

qDebug() << "Invalid metal code:" << toCurrency;

return 0.0;

}

double inputValue = ui->textEdit->toPlainText().toDouble();// Получение значения из первого текстового поля

qDebug() << inputValue;

qDebug() << fromCurrency;

if (fromCurrency == "RUB"){

double convertedValue = inputValue / metalPrice;

return convertedValue;

}

if (fromCurrency == "EUR"){

double eurMultiplier = ::getCurrencyMultiplier("EUR");

double convertedValue = (inputValue \* eurMultiplier)/ metalPrice;

return convertedValue;

}

if (fromCurrency == "USD"){

double usdMultiplier = ::getCurrencyMultiplier("USD");

double convertedValue = (inputValue \* usdMultiplier) / metalPrice;

return convertedValue;

}

} else {

}

}

void DragMetal::on\_pushButton\_11\_clicked()

{

double convertedValue = convertCurrency(fromCurrency, toCurrency);

QString convertedValueString = QString::number(convertedValue, 'f', 2);

convertedValueString.append("г"); // Добавление символа "г" к строке

ui->textEdit\_2->setText(convertedValueString);

// Check if the metal code matches the selected value in comboBox\_2

QString selectedMetalCode = ui->comboBox\_2->currentData().toString(); // Assuming you set the metal code as the data for each combo box item

QString selectedMetalName = ui->comboBox\_2->currentText();

qDebug() << "Selected Metal Code:" << selectedMetalCode;

qDebug() << "To Currency:" << toCurrency;

if (QString::compare(selectedMetalCode, toCurrency, Qt::CaseInsensitive) == 0) {

qDebug() << "Metal code matches the selected value in comboBox\_2.";

} else {

qDebug() << "Metal code does not match the selected value in comboBox\_2.";

qDebug() << "Selected Metal Name:" << selectedMetalName;

}

}

void DragMetal::on\_pushButton\_9\_clicked()

{

close();

obmakc dialog;

dialog.setModal(true);

dialog.exec();

}

void DragMetal::on\_pushButton\_10\_clicked()

{

close();

sosakc dialog;

dialog.setModal(true);

dialog.exec();

}

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

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\_comboBox\_currentIndexChanged(int index);

void on\_comboBox\_2\_currentIndexChanged(int index);

void on\_pushButton\_4\_clicked();

void on\_pushButton\_6\_clicked();

void on\_pushButton\_3\_clicked();

void on\_pushButton\_5\_clicked();

private:

Ui::MainWindow \*ui;

double convertCurrency(const QString& fromCurrency, const QString& toCurrency);

QString fromCurrency;

QString toCurrency;

};

#endif // MAINWINDOW\_H

**mainwindow.cpp**

#include "mainwindow.h"

#include "./ui\_mainwindow.h"

#include "QString"

#include <QNetworkAccessManager>

#include <QNetworkReply>

#include <QJsonDocument>

#include <QJsonObject>

#include <QJsonArray>

#include <QUrlQuery>

#include <QTextEdit>

#include <dragmetal.h>

#include <obmakc.h>

#include <sosakc.h>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_comboBox\_currentIndexChanged(int /\*index\*/)

{

QString selectedFromCurrency = ui->comboBox->currentText();

QString selectedToCurrency = ui->comboBox\_2->currentText();

if (!selectedFromCurrency.isEmpty()) {

fromCurrency = selectedFromCurrency;

} else {

qDebug() << "Invalid fromCurrency";

}

if (!selectedToCurrency.isEmpty()) {

toCurrency = selectedToCurrency;

} else {

qDebug() << "Invalid toCurrency";

}

}

void MainWindow::on\_comboBox\_2\_currentIndexChanged(int /\*index\*/)

{

QString selectedFromCurrency = ui->comboBox->currentText();

QString selectedToCurrency = ui->comboBox\_2->currentText();

if (!selectedFromCurrency.isEmpty()) {

fromCurrency = selectedFromCurrency;

} else {

qDebug() << "Invalid fromCurrency";

}

if (!selectedToCurrency.isEmpty()) {

toCurrency = selectedToCurrency;

} else {

qDebug() << "Invalid toCurrency";

}

}

double MainWindow::convertCurrency(const QString& fromCurrency, const QString& toCurrency)

{

QNetworkAccessManager manager;

QUrl url("https://www.cbr-xml-daily.ru/daily\_json.js"); // URL API Центробанка

QNetworkRequest request(url);

QNetworkReply\* reply = manager.get(request);

QEventLoop loop;

connect(reply, &QNetworkReply::finished, &loop, &QEventLoop::quit);

loop.exec();

if (reply->error() != QNetworkReply::NoError) {

// Обработка ошибки запроса

qDebug() << "Error: " << reply->errorString();

return 0.0;

}

QByteArray responseData = reply->readAll();

QJsonDocument jsonDocument = QJsonDocument::fromJson(responseData);

if (jsonDocument.isNull()) {

// Обработка ошибки парсинга JSON

qDebug() << "Error: Failed to parse JSON response";

return 0.0;

}

QJsonObject jsonObject = jsonDocument.object();

QJsonObject valuteObject = jsonObject["Valute"].toObject();

double fromRate = 0.0;

double toRate = 0.0;

if (fromCurrency == "RUB") {

// Конвертация из рубля

fromRate = 1.0;

} else {

if (valuteObject.contains(fromCurrency)) {

QJsonObject fromCurrencyObject = valuteObject[fromCurrency].toObject();

fromRate = fromCurrencyObject["Value"].toDouble();

} else {

qDebug() << "Invalid fromCurrency:" << fromCurrency;

}

}

if (toCurrency == "RUB") {

toRate = 1.0;

}

if (valuteObject.contains(toCurrency)) {

QJsonObject toCurrencyObject = valuteObject[toCurrency].toObject();

toRate = toCurrencyObject["Value"].toDouble();

} else {

qDebug() << "Invalid toCurrency:" << toCurrency;

}

if (fromRate == 0.0 || toRate == 0.0) {

// Обработка некорректных значений курсов валют

qDebug() << "Error: Invalid currency code";

return 0.0;

}

double inputValue = ui->textEdit->toPlainText().toDouble(); // Получение значения из первого текстового поля

double convertedValue = inputValue \* (fromRate / toRate);

return convertedValue;

}

void MainWindow::on\_pushButton\_4\_clicked()

{

if (fromCurrency == toCurrency) {

QString inputValue = ui->textEdit->toPlainText();

ui->textEdit\_2->setText(inputValue);

return;

}

// Выполнение операции конвертации и обновление значения валюты

double convertedValue = convertCurrency(fromCurrency, toCurrency);

QString roundedValue = QString::number(convertedValue, 'f', 2); // Округление до сотых

ui->textEdit\_2->setText(roundedValue);

}

void MainWindow::on\_pushButton\_6\_clicked()

{

close();

DragMetal dialog; // Create an instance of the Drag\_Metal dialog

dialog.setModal(true); // Set the dialog as modal (blocks input to other windows)

dialog.exec(); // Execute the dialog, which will open it

}

void MainWindow::on\_pushButton\_3\_clicked()

{

close();

obmakc dialog;

dialog.setModal(true);

dialog.exec();

}

void MainWindow::on\_pushButton\_5\_clicked()

{

close();

sosakc dialog;

dialog.setModal(true);

dialog.exec();

}

**obmakc.h**

#ifndef OBMAKC\_H

#define OBMAKC\_H

#include "ui\_obmakc.h"

#include <QDialog>

namespace Ui {

class obmakc;

}

class obmakc : public QDialog

{

Q\_OBJECT

public:

explicit obmakc(QWidget \*parent = nullptr);

~obmakc();

private slots:

void on\_pushButton\_4\_clicked();

void on\_pushButton\_7\_clicked();

void on\_pushButton\_10\_clicked();

void on\_comboBox\_currentIndexChanged(int index);

void on\_comboBox\_2\_currentIndexChanged(int index);

void on\_pushButton\_11\_clicked();

private:

Ui::obmakc \*ui;

QString fromStockCode;

QString toStockCode;

double fetchStockPrice(const QString& fromStockCode, const QString& toStockCode);

void calculateTargetStockQuantity(double fromStockPrice, double toStockPrice);

void fetchStockPrices(const QString& fromStockCode, const QString& toStockCode);

};

#endif // OBMAKC\_H

**obmakc.cpp**

#include "obmakc.h"

#include "ui\_obmakc.h"

#include <dragmetal.h>

#include <mainwindow.h>

#include <sosakc.h>

#include "QString"

#include "qeventloop.h"

#include <QNetworkAccessManager>

#include <QNetworkReply>

#include <QJsonDocument>

#include <QJsonObject>

#include <QJsonArray>

#include <QUrlQuery>

#include <QTextEdit>

#include <QDomDocument>

obmakc::obmakc(QWidget \*parent) :

QDialog(parent),

ui(new Ui::obmakc)

{

ui->setupUi(this);

QString fromStockCode = ui->comboBox->currentText();

QString toStockCode = ui->comboBox\_2->currentText();

}

obmakc::~obmakc()

{

delete ui;

}

void obmakc::on\_pushButton\_4\_clicked()

{

close();

MainWindow\* mainWindow = new MainWindow();

mainWindow->show();

}

void obmakc::on\_pushButton\_7\_clicked()

{

close();

DragMetal dialog;

dialog.setModal(true);

dialog.exec();

}

void obmakc::on\_pushButton\_10\_clicked()

{

close();

sosakc dialog;

dialog.setModal(true);

dialog.exec();

}

void obmakc::on\_comboBox\_currentIndexChanged(int /\*index\*/)

{

QString fromStockCode = ui->comboBox->currentText();

QString toStockCode = ui->comboBox\_2->currentText();

}

void obmakc::on\_comboBox\_2\_currentIndexChanged(int /\*index\*/)

{

QString fromStockCode = ui->comboBox->currentText();

QString toStockCode = ui->comboBox\_2->currentText();

}

double obmakc::fetchStockPrice(const QString& stockCode, const QString& apiKey)

{

QNetworkAccessManager manager;

QUrl url(QString("https://www.alphavantage.co/query?function=GLOBAL\_QUOTE&symbol=%1&apikey=%2").arg(stockCode).arg(apiKey));

QNetworkRequest request(url);

QNetworkReply\* reply = manager.get(request);

QEventLoop loop;

connect(reply, &QNetworkReply::finished, &loop, &QEventLoop::quit);

loop.exec();

if (reply->error() != QNetworkReply::NoError) {

// Handle request error

qDebug() << "Error fetching stock price: " << reply->errorString();

return 0.0;

}

QByteArray responseData = reply->readAll();

qDebug() << "Stock Response Data:" << responseData;

QJsonDocument jsonDocument = QJsonDocument::fromJson(responseData);

if (jsonDocument.isNull()) {

// Handle JSON parsing error

qDebug() << "Error: Failed to parse JSON response";

return 0.0;

}

QJsonObject jsonObject = jsonDocument.object();

QJsonObject globalQuoteObject = jsonObject["Global Quote"].toObject();

double stockPrice = globalQuoteObject["05. price"].toString().toDouble();

qDebug() << "Stock Price:" << stockPrice;

return stockPrice;

}

void obmakc::fetchStockPrices(const QString& fromStockCode, const QString& toStockCode)

{

QString fromApiKey = "9EAQNXHZ4IJ46YGI";

QString toApiKey = "28N1I3P2Z1DBTM4P";

double fromStockPrice = fetchStockPrice(fromStockCode, fromApiKey);

double toStockPrice = fetchStockPrice(toStockCode, toApiKey);

calculateTargetStockQuantity(fromStockPrice, toStockPrice);

}

void obmakc::calculateTargetStockQuantity(double fromStockPrice, double toStockPrice)

{

double inputValue = ui->textEdit->toPlainText().toDouble();

qDebug() << "Input Value:" << inputValue;

// Check for division by zero

if (toStockPrice == 0.0) {

qDebug() << "Error: To stock price is zero";

return;

}

// Calculate the number of target stocks that can be bought

double targetStockQuantity = (inputValue \* fromStockPrice) / toStockPrice;

qDebug() << "Target Stock Quantity:" << targetStockQuantity;

// Display the result in textEdit\_2

ui->textEdit\_2->setText(QString::number(targetStockQuantity, 'f', 2));

}

void obmakc::on\_pushButton\_11\_clicked()

{

QString fromStockCode = ui->comboBox->currentText();

QString toStockCode = ui->comboBox\_2->currentText();

qDebug() << "From Stock Code:" << fromStockCode;

qDebug() << "To Stock Code:" << toStockCode;

fetchStockPrices(fromStockCode, toStockCode);

}

**sosakc.h**

#ifndef SOSAKC\_H

#define SOSAKC\_H

#include <QDialog>

namespace Ui {

class sosakc;

}

class sosakc : public QDialog

{

Q\_OBJECT

public:

explicit sosakc(QWidget \*parent = nullptr);

~sosakc();

private slots:

void on\_pushButton\_4\_clicked();

void on\_pushButton\_7\_clicked();

void on\_pushButton\_9\_clicked();

void on\_comboBox\_currentIndexChanged(int index);

void on\_pushButton\_11\_clicked();

private:

Ui::sosakc \*ui;

double fetchStockPrice(const QString& fromStockCode);

};

#endif // SOSAKC\_H

**sosakc.cpp**

#include "sosakc.h"

#include "ui\_sosakc.h"

#include <mainwindow.h>

#include <obmakc.h>

#include <dragmetal.h>

#include <QNetworkAccessManager>

#include <QNetworkReply>

#include <QJsonDocument>

#include <QJsonObject>

#include <QJsonArray>

#include <QUrlQuery>

#include <QTextEdit>

#include <QDomDocument>

#include <QWebEngineView>

sosakc::sosakc(QWidget \*parent) :

QDialog(parent),

ui(new Ui::sosakc)

{

ui->setupUi(this);

QString fromStockCode = ui->comboBox->currentText();

ui->textEdit->setText(QString::number(fetchStockPrice(fromStockCode), 'f', 2).append("$"));

}

sosakc::~sosakc()

{

delete ui;

}

void sosakc::on\_pushButton\_4\_clicked()

{

close();

MainWindow\* mainWindow = new MainWindow();

mainWindow->show();

}

void sosakc::on\_pushButton\_7\_clicked()

{

close();

DragMetal dialog;

dialog.setModal(true);

dialog.exec();

}

void sosakc::on\_pushButton\_9\_clicked()

{

close();

obmakc dialog;

dialog.setModal(true);

dialog.exec();

}

void sosakc::on\_comboBox\_currentIndexChanged(int /\*index\*/)

{

QString fromStockCode = ui->comboBox->currentText();

ui->textEdit->setText(QString::number(fetchStockPrice(fromStockCode), 'f', 2).append("$"));

}

double sosakc::fetchStockPrice(const QString& fromStockCode)

{

QNetworkAccessManager manager;

QUrl url(QString("https://www.alphavantage.co/query?function=GLOBAL\_QUOTE&symbol=%1&apikey=9EAQNXHZ4IJ46YGI").arg(fromStockCode));

QNetworkRequest request(url);

QNetworkReply\* reply = manager.get(request);

QEventLoop loop;

connect(reply, &QNetworkReply::finished, &loop, &QEventLoop::quit);

loop.exec();

if (reply->error() != QNetworkReply::NoError) {

// Handle request error

qDebug() << "Error fetching stock price: " << reply->errorString();

return 0.0;

}

QByteArray responseData = reply->readAll();

qDebug() << "Stock Response Data:" << responseData;

QJsonDocument jsonDocument = QJsonDocument::fromJson(responseData);

if (jsonDocument.isNull()) {

// Handle JSON parsing error

qDebug() << "Error: Failed to parse JSON response";

return 0.0;

}

QJsonObject jsonObject = jsonDocument.object();

QJsonObject globalQuoteObject = jsonObject["Global Quote"].toObject();

double stockPrice = globalQuoteObject["05. price"].toString().toDouble();

qDebug() << "Stock Price:" << stockPrice;

return stockPrice;

}

void sosakc::on\_pushButton\_11\_clicked()

{

QWebEngineView\* webView = new QWebEngineView(this);

QString stockCode = ui->comboBox->currentText();

webView->load(QUrl("https://finance.yahoo.com/chart/" + stockCode));

Ё

QMainWindow\* mainWindow = new QMainWindow(this);

mainWindow->setCentralWidget(webView);

mainWindow->resize(900, 900); // Установка размеров окна

mainWindow->setWindowTitle("LSD ARM");

mainWindow->show();

}

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

#include <QLocale>

#include <QTranslator>

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

{

QApplication a(argc, argv);

a.setWindowIcon(QIcon(":/new/prefix1/images/icon.ico"));

QTranslator translator;

const QStringList uiLanguages = QLocale::system().uiLanguages();

for (const QString &locale : uiLanguages) {

const QString baseName = "LSD1\_" + QLocale(locale).name();

if (translator.load(":/i18n/" + baseName)) {

a.installTranslator(&translator);

break;

}

}

MainWindow w;

w.show();

return a.exec();

}

Видео-презентация с тестами программ - https://youtu.be/Cu8JgR9nTfU