mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

//#include <QNetworkAccessManager>

//#include <QNetworkReply>

#include <QUrl>

#include <QUrlQuery>

#include "../HW1/flipcoder.h"

#include "../HW1/otpcoder.h"

#include "../HW1/invertcasecoder.h"

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void runCoders();

void regenerateOTP();

void callBase64WebService(const QString& text);

private:

Ui::MainWindow \*ui;

OTPCoder\* otpCoder;

};

#endif // MAINWINDOW\_H

mainwindow.cpp

#include "mainwindow.h"

#include "./ui\_mainwindow.h"

#include <QNetworkAccessManager>

#include <QNetworkReply>

#include <QJsonDocument>

#include <QJsonObject>

#include <QMessageBox>

#include <QRandomGenerator>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

// Connect the buttons to slots

connect(ui->runCodersButton, SIGNAL(clicked()), this, SLOT(runCoders()));

connect(ui->regenerateOTPButton, SIGNAL(clicked()), this, SLOT(regenerateOTP()));

// Set default text in QTextEdit

ui->inputTextEdit->setPlainText("THESE are the times that try men's souls.");

// Initialize OTPCoder and display the initial OTP pad

otpCoder = new OTPCoder();

regenerateOTP();

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::runCoders() {

// Get the input text from QTextEdit

QString inputText = ui->inputTextEdit->toPlainText();

// Encode and decode using FlipCoder

FlipCoder flipCoder;

QString flipped = flipCoder.encode(inputText);

QString flippedDecoded = flipCoder.decode(flipped);

// Encode and decode using OTPCoder

QString otpEncoded = otpCoder->encode(flippedDecoded);

QString otpDecoded = otpCoder->decode(otpEncoded);

// Encode and decode using InvertCaseCoder

InvertCaseCoder invertCoder;

QString inverted = invertCoder.encode(otpDecoded);

QString invertedDecoded = invertCoder.decode(inverted);

// Call the Base64 web service for encoding

callBase64WebService(invertedDecoded);

}

void MainWindow::regenerateOTP() {

// Generate a new OTP pad with random values

otpCoder->regenerateOTP();

// Display the current OTP pad

ui->otpPadLabel->setText("Current OTP Pad: " + otpCoder->getCurrentOTP());

}

void MainWindow::callBase64WebService(const QString& text) {

// Create a QNetworkAccessManager

QNetworkAccessManager \*manager = new QNetworkAccessManager(this);

// Create the request URL

QUrl url("https://networkcalc.com/api/encoder/" + text);

QUrlQuery query;

query.addQueryItem("encoding", "base64");

url.setQuery(query);

// Send the GET request

QNetworkReply \*reply = manager->get(QNetworkRequest(url));

// Connect slots for handling the response

connect(reply, &QNetworkReply::finished, this, [=]() {

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

// Parse and display the response

QByteArray responseData = reply->readAll();

QJsonDocument jsonResponse = QJsonDocument::fromJson(responseData);

QString base64Encoded = jsonResponse.object()["encoded"].toString();

ui->resultsTextBrowser->setPlainText("Input (in bold):\n<b>" + text + "</b>\n\nBase64 Encoded:\n" + base64Encoded);

} else {

// Handle the error

ui->resultsTextBrowser->setPlainText("Error: " + reply->errorString());

}

// Clean up

reply->deleteLater();

manager->deleteLater();

});

}

flipcode.h

#ifndef FLIPCODER\_H

#define FLIPCODER\_H

#include <QString>

class FlipCoder {

public:

QString encode(const QString& input);

QString decode(const QString& input);

};

#endif // FLIPCODER\_H

flipcode.cpp

#include "flipcoder.h"

QString FlipCoder::encode(const QString& input) {

QString encoded;

for (const QChar& c : input) {

ushort ascii = c.unicode();

if (ascii >= 0x20 && ascii <= 0x7E && c != ' ') {

encoded += QChar(0x7F - (ascii - 0x20));

} else {

encoded += c;

}

}

return encoded;

}

QString FlipCoder::decode(const QString& input) {

QString decoded;

for (const QChar& c : input) {

ushort ascii = c.unicode();

if (ascii >= 0x20 && ascii <= 0x7E && c != ' ') {

decoded += QChar(0x7F - (ascii - 0x20));

} else {

decoded += c;

}

}

return decoded;

}

invertcasecoder.h

#ifndef INVERTCASECODER\_H

#define INVERTCASECODER\_H

#include <QString>

class InvertCaseCoder {

public:

QString encode(const QString& input);

QString decode(const QString& input);

};

#endif // INVERTCASECODER\_H

invertcasecoder.cpp

#include "invertcasecoder.h"

QString InvertCaseCoder::encode(const QString& input) {

QString encoded;

for (const QChar& c : input) {

if (c.isUpper()) {

encoded += c.toLower();

} else if (c.isLower()) {

encoded += c.toUpper();

} else {

encoded += c;

}

}

return encoded;

}

QString InvertCaseCoder::decode(const QString& input) {

return encode(input); // Decoding is the same as encoding for invert case

}

optcoder.h

#ifndef OTPCODER\_H

#define OTPCODER\_H

#include <QString>

class OTPCoder {

public:

OTPCoder();

QString encode(const QString& input);

QString decode(const QString& input);

private:

int pad[11]; // Your pad values go here

};

#endif // OTPCODER\_H

optcoder.cpp

#include "otpcoder.h"

OTPCoder::OTPCoder() {

// Initialize your OTP pad with prime numbers

pad[0] = 17;

pad[1] = 29;

pad[2] = 5;

pad[3] = 12;

pad[4] = 6;

pad[5] = 22;

pad[6] = 11;

pad[7] = 30;

pad[8] = 8;

pad[9] = 18;

pad[10] = 15;

}

QString OTPCoder::encode(const QString& input) {

QString encoded;

int padIndex = 0;

for (const QChar& c : input) {

ushort ascii = c.unicode();

if (ascii >= 0x20 && ascii <= 0x7E) {

int offset = pad[padIndex];

padIndex = (padIndex + 1) % 11;

int encodedAscii = ascii + offset;

if (encodedAscii > 0x7E) {

encodedAscii -= 0x5F; // Bring it back into the range 0x20 to 0x7E

}

encoded += QChar(encodedAscii);

} else {

encoded += c;

}

}

return encoded;

}

QString OTPCoder::decode(const QString& input) {

QString decoded;

int padIndex = 0;

for (const QChar& c : input) {

ushort ascii = c.unicode();

if (ascii >= 0x20 && ascii <= 0x7E) {

int offset = pad[padIndex];

padIndex = (padIndex + 1) % 11;

int decodedAscii = ascii - offset;

if (decodedAscii < 0x20) {

decodedAscii += 0x5F; // Bring it back into the range 0x20 to 0x7E

}

decoded += QChar(decodedAscii);

} else {

decoded += c;

}

}

return decoded;

}

main.cpp

#include "mainwindow.h"

#include <QCoreApplication>

#include <iostream>

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow mainWindow;

mainWindow.show();

// Test with the first string

QString input1 = "In a hole in the ground there lived a Hobbit 1234567890 ~1@#$%^＆\*0+=\"";

mainWindow.ui->inputTextEdit->setPlainText(input1);

mainWindow.runCoders();

// Output a separator

std::cout << "\n";

// Test with the second string

QString input2 = "England expects every man will do his duty";

mainWindow.ui->inputTextEdit->setPlainText(input2);

mainWindow.regenerateOTP(); // Regenerate OTP for the second test

mainWindow.runCoders();

return a.exec();

}