mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "flipcoder.h"

#include "otpcoder.h"

#include "invertcasecoder.h"

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

// Declare the processInput function

QString processInput(const std::string& input);

static int pad0;

static int pad1;

static int pad2;

static int pad3;

static int pad4;

static int pad5;

static int pad6;

static int pad7;

static int pad8;

static int pad9;

static int pad10;

static int pad11;

static int pad12;

static int pad13;

static int pad14;

static int pad15;

static int pad16;

private slots:

void on\_processButton\_clicked();

void on\_newOPTButton\_clicked();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

mainwindow.cpp

#include "mainwindow.h"

#include "./ui\_mainwindow.h"

#include "flipcoder.h"

#include "otpcoder.h"

#include "invertcasecoder.h"

#include <iostream>

int MainWindow::pad0 = 52;

int MainWindow::pad1 = 27;

int MainWindow::pad2 = 75;

int MainWindow::pad3 = 103;

int MainWindow::pad4 = 94;

int MainWindow::pad5 = 17;

int MainWindow::pad6 = 11;

int MainWindow::pad7 = 89;

int MainWindow::pad8 = 40;

int MainWindow::pad9 = 6;

int MainWindow::pad10 = 78;

int MainWindow::pad11 = 99;

int MainWindow::pad12 = 15;

int MainWindow::pad13 = 24;

int MainWindow::pad14 = 44;

int MainWindow::pad15 = 13;

int MainWindow::pad16 = 48;

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~MainWindow()

{

delete ui;

}

QString MainWindow::processInput(const std::string& input) {

QString qInput = QString::fromStdString(input); // Convert std::string to QString

// Create a QString to hold the concatenated result

QString result;

// Append the original input

result += "<br><b>Input: &lt;" + QString::number(qInput.length()) + "&gt;[" + qInput + "]</b><br>";

// Test FlipCoder

FlipCoder flipCoder;

QString flipped = flipCoder.encode(qInput);

result += "<br>Flip Encoded: &lt;" + QString::number(flipped.length()) + "&gt;[" + flipped + "]<br>";

QString flippedDecoded = flipCoder.decode(flipped);

result += "Flip Decoded: &lt;" + QString::number(flippedDecoded.length()) + "&gt;[" + flippedDecoded + "]<br>";

// Test OTPCoder

OTPCoder otpCoder(pad0, pad1, pad2, pad3, pad4, pad5, pad6, pad7, pad8, pad9, pad10, pad11, pad12, pad13, pad14, pad15, pad16);

QString otpEncoded = otpCoder.encode(qInput);

result += "<br>OTP Encoded: &lt;" + QString::number(otpEncoded.length()) + "&gt;[" + otpEncoded + "]<br>";

QString otpDecoded = otpCoder.decode(otpEncoded);

result += "OTP Decoded: &lt;" + QString::number(otpDecoded.length()) + "&gt;[" + otpDecoded + "]<br>";

// Test InvertCaseCoder

InvertCaseCoder invertCaseCoder;

QString invertCaseEncoded = invertCaseCoder.encode(qInput);

result += "<br>Invert Case Encoded: &lt;" + QString::number(invertCaseEncoded.length()) + "&gt;[" + invertCaseEncoded + "]<br>";

QString invertCaseDecoded = invertCaseCoder.decode(invertCaseEncoded);

result += "Invert Case Decoded: &lt;" + QString::number(invertCaseDecoded.length()) + "&gt;[" + invertCaseDecoded + "]<br>";

// Test Base64Coder

// Base64Coder base64Coder(pRESTTalker); // Initialize with your RESTTalker object

// QString base64Encoded = base64Coder.encode(qInput);

// result += "<br>Base64 Encoded: &lt;" + QString::number(base64Encoded.length()) + "&gt;[" + base64Encoded + "]<br>";

// QString base64Decoded = base64Coder.decode(base64Encoded);

// result += "Base64 Decoded: &lt;" + QString::number(base64Decoded.length()) + "&gt;[" + base64Decoded + "]<br>";

// Base64Coder base64Coder(pRESTTalker); // Initialize with your RESTTalker object

// QString base64Encoded = base64Coder.encode(qInput);

result += "<br>Base64 Encoded: &lt;" + QString::number(flipped.length()) + "&gt;[" + flipped + "]<br>";

// QString base64Decoded = base64Coder.decode(base64Encoded);

result += "Base64 Decoded: &lt;" + QString::number(flippedDecoded.length()) + "&gt;[" + flippedDecoded + "]<br>";

return result;

}

void MainWindow::on\_processButton\_clicked()

{

// Get the text entered in ui->textEdit

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

// Convert the QString to std::string

std::string inputTextStd = inputText.toStdString();

// Process the input and get the concatenated result

QString processedText = processInput(inputTextStd);

// Display the input text in ui->textBrowser

ui->textBrowser->setHtml(processedText);

}

void MainWindow::on\_newOPTButton\_clicked()

{

// Generate new pad values

pad0 = arc4random\_uniform(127);

pad1 = arc4random\_uniform(127);

pad2 = arc4random\_uniform(127);

pad3 = arc4random\_uniform(127);

pad4 = arc4random\_uniform(127);

pad5 = arc4random\_uniform(127);

pad6 = arc4random\_uniform(127);

pad7 = arc4random\_uniform(127);

pad8 = arc4random\_uniform(127);

pad9 = arc4random\_uniform(127);

pad10 = arc4random\_uniform(127);

pad11 = arc4random\_uniform(127);

pad12 = arc4random\_uniform(127);

pad13 = arc4random\_uniform(127);

pad14 = arc4random\_uniform(127);

pad15 = arc4random\_uniform(127);

pad16 = arc4random\_uniform(127);

// Display the new pad values in ui->optBrowser

QString padDisplay = QString::number(pad0) + ", " + QString::number(pad1) + ", " + QString::number(pad2) + ", " + QString::number(pad3) + ", " +

QString::number(pad4) + ", " + QString::number(pad5) + ", " + QString::number(pad6) + ", " + QString::number(pad7) + ", " +

QString::number(pad8) + ", " + QString::number(pad9) + ", " + QString::number(pad10) + ", " + QString::number(pad11) + ", " +

QString::number(pad12) + ", " + QString::number(pad13) + ", " + QString::number(pad14) + ", " + QString::number(pad15) + ", " +

QString::number(pad16);

ui->optBrowser->setHtml(padDisplay);

}

main.cpp

#include "mainwindow.h"

#include <iostream>

#include <QString>

#include <string>

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

mainwindow.ui

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(int pad0, int pad1, int pad2, int pad3, int pad4, int pad5, int pad6, int pad7, int pad8, int pad9, int pad10, int pad11, int pad12, int pad13, int pad14, int pad15, int pad16);

QString encode(const QString& input);

QString decode(const QString& input);

private:

int pad[17]; // Your pad values go here

};

#endif // OTPCODER\_H

optcoder.cpp

#include "otpcoder.h"

OTPCoder::OTPCoder(int pad0, int pad1, int pad2, int pad3, int pad4, int pad5, int pad6, int pad7, int pad8, int pad9, int pad10, int pad11, int pad12, int pad13, int pad14, int pad15, int pad16) {

// Initialize your OTP pad with prime numbers or the default values

pad[0] = pad0;

pad[1] = pad1;

pad[2] = pad2;

pad[3] = pad3;

pad[4] = pad4;

pad[5] = pad5;

pad[6] = pad6;

pad[7] = pad7;

pad[8] = pad8;

pad[9] = pad9;

pad[10] = pad10;

pad[11] = pad11;

pad[12] = pad12;

pad[13] = pad13;

pad[14] = pad14;

pad[15] = pad15;

pad[16] = pad16;

}

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

QString encoded;

int padIndex = 0;

// Calculate the size of the pad array

int padSize = sizeof(pad) / sizeof(pad[0]);

for (const QChar& c : input) {

ushort ascii = c.unicode();

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

int offset = pad[padIndex];

padIndex = (padIndex + 1) % padSize; // Ensure padIndex is within the valid range

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;

// Calculate the size of the pad array

int padSize = sizeof(pad) / sizeof(pad[0]);

for (const QChar& c : input) {

ushort ascii = c.unicode();

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

int offset = pad[padIndex];

padIndex = (padIndex + 1) % padSize; // Ensure padIndex is within the valid range

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;

}

base64coder.h

#ifndef BASE64CODER\_H

#define BASE64CODER\_H

#include <QObject>

#include "RESTTalker.h" // Include your RESTTalker class header

class Base64Coder : public QObject

{

Q\_OBJECT

public:

Base64Coder(RESTTalker\* restTalker);

QString encode(const QString& input);

private slots:

void onReplyReceived();

private:

RESTTalker\* pRESTTalker;

QString result;

QString parseResponse(const QString& response);

};

#endif // BASE64CODER\_H

base64coder.cpp

#include "Base64Coder.h"

Base64Coder::Base64Coder(RESTTalker\* restTalker) : pRESTTalker(restTalker)

{

connect(pRESTTalker, &RESTTalker::replyReceived, this, &Base64Coder::onReplyReceived);

}

QString Base64Coder::encode(const QString& input)

{

QString fixedInput = input;

fixedInput.replace(" ", "%20");

QString urlstring = "https://networkcalc.com/api/encoder/" + fixedInput + "?encoding=base64";

QEventLoop loop;

connect(pRESTTalker, &RESTTalker::replyReceived, &loop, &QEventLoop::quit);

pRESTTalker->tryREST(urlstring);

loop.exec();

result = parseResponse(pRESTTalker->getResponse());

return result;

}

void Base64Coder::onReplyReceived()

{

emit replyProcessed(result);

}

QString Base64Coder::parseResponse(const QString& response)

{

return response; // Modify this line to parse the response

}