НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені Ігоря Сікорського»

Кафедра прикладної математики

**КУРСОВА РОБОТА**

з дисципліни “Програмування”

на тему:

ШИФРУВАННЯ ТА ДЕШИФРУВАННЯ ТЕКСТУ

Виконав:

студент І курсу групи КМ-73,

спеціальність 113 – прикладна

математика

Садченко Микита В’ячеславович

Керівник:

Громова В. В.

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_

Оцінка ECTS: \_\_\_\_\_\_\_\_

Київ — 2018

ЗМІСТ

[ВСТУП 3](#_Toc515873672)

[1 ПОСТАНОВКА ЗАДАЧІ 4](#_Toc515873678)

[2 ВИБІР МЕТОДІВ РОЗВ’ЯЗАННЯ 5](#_Toc515873680)

[3 АЛГОРИТМИ ОБРАНИХ МЕТОДІВ 7](#_Toc515873681)

[4 ОПИС ПРОГРАМИ 10](#_Toc515873682)

[5 РЕЗУЛЬТАТ ПРОГРАМИ 12](#_Toc515873683)

[ВИСНОВОК 15](#_Toc515873684)

[ЛІТЕРАТУРА 16](#_Toc515873685)

[ДОДАТОК А (ТЕКСТ ПРОГРАМИ НА МОВІ С++) 17](#_Toc515873686)

# ВСТУП

Шифрування - оборотне перетворення інформації з метою приховування від неавторизованих осіб, з наданням, в цей же час, авторизованим користувачам доступу до неї. Головним чином, шифрування служить завданням дотримання конфіденційності інформації, що передається. Важливою особливістю будь-якого алгоритму шифрування є використання ключа, який стверджує вибір конкретного перетворення з сукупності можливих для даного алгоритму.

Для того, щоб прочитати зашифровану інформацію, приймаючій стороні необхідні ключ і дешифратор (пристрій, що реалізує алгоритм розшифрування). Ідея шифрування полягає в тому, що зловмисник, перехопивши зашифровані дані і не маючи до них ключа, не може ні прочитати, ні змінити передану інформацію.

Шифром називається пара алгоритмів, що реалізують кожне із зазначених перетворень. Ці алгоритми застосовуються до даних з використанням ключа. Ключі для шифрування і для розшифрування можуть відрізнятися, а можуть бути однаковими.

На даний момент існує величезна кількість методів шифрування. Головним чином ці методи діляться, в залежності від структури використовуваних ключів, на симетричні методи і асиметричні методи. Крім того, методи шифрування можуть мати різну криптостійкості і по-різному обробляти вхідні дані - блокові шифри і потокові шифри. Всіма цими методами, їх створенням і аналізом займається наука криптографія.

Зараз будуть розглянуті досить прості алгоритми, які неважко розшифрувати, але які є цікавими.

# 1 ПОСТАНОВКА ЗАДАЧІ

В даній курсовій роботі необхідно виконати наступні вимоги:

1. Написати програму на C-подібній мові (в даному випадку C++ за допомогою графічного редактора QT Creator).
2. В програмі повинно бути присутнє меню, в якому присутні наступні пункти: Exit (вихід), About program (загальна інформація про програму), та пункти п’ятьох алгоритмів (Generator, Caesar and Hill ciphers, Morse code, Translit).
3. Інтерфейс програми має бути простим, інтуїтивно зрозумілим та відповідати поставленим вимогам.
4. На сторінці кожного алгоритму містяться кнопки повернення до головного меню (to the Menu), інформація про даний алгоритм (About ..) та приклад у вигляді рисунку для візуального сприйняття (є не у всіх), та власне поля, що необхідні алгоритму.
5. Задача кожного алгоритму має бути зрозуміла користувачеві. Для цього є додаткова інформація в відповідному пункті.

# 2 ВИБІР МЕТОДІВ РОЗВ’ЯЗАННЯ

Для даної курсової роботи підходять за темою всі алгоритми шифрування даних, але обрано було тільки п’ять – це Генератор паролів, Шифри Цезаря та Хілла, Азбука Морзе та так званий «Трансліт».

Реалізація *генератору паролів* за допомогою комп’ютера виконується лише за допомогою псевдо-випадкових чисел (вони не повністю випадкові, бо для їх генерації існує алгоритм). В даній програмі це зроблено так: спочатку рандомом обирається множина (великий регістр літер англійського алфавіту, нижній регістр та цифри), далі, теж рандомом, із цієї множини обирається символ, який одразу додається до результуючої строки.

*Шифр Цезаря*, мабуть, є найпростішим алгоритмом для шифрування повідомлень. Він переставляє кожну літеру слова на заданий крок в алфавіті. Для його реалізації на мові C++ можна задати алфавіт у масиві та працювати з його індексами, а можна напряму працювати з ASCII-кодом алфавіту (оскільки літери алфавіту у таблиці йдуть один за одним), не витрачаючи пам’ять. Мною було обрано другий варіант. Хоч з ASCII-кодом важче працювати, ніж з масивом, через постійну необхідність конвертації типів даних, але даний спосіб не витрачає пам’ять, тому трохи кращий.

*Шифр Хілла* з усіх описаних алгоритмів найскладніший, оскільки тут виконуються дії з матрицями. Для цього шифру існує єдиний алгоритм, винайдений Лестером Хіллом. Спочатку вводиться ключ (довжина має дорівнювати цілому кореню цілого числа, за винятком одиниці) та повідомлення. Ключ зашифровується у квадратну матрицю, де кожен елемент відповідає положенню символу у заданому алфавіті (зазвичай, звичайний алфавіт збільшують за допомогою додаткових символів для отримання простого числа (що ділиться само на себе та на одиницю), через що ключ складніше розшифрувати). Повідомлення ділиться на рядки з довжиною розміру матриці ключа. Кожен з цих рядків для *шифрування* множиться на матрицю ключа. Дали всі отримані рядки з’єднуємо та одержуємо результат. Для *дешифрування* рядки необхідно множити на обернену матрицю ключа (в знаходженні якої приймає участь розширений алгоритм Евкліда).

*Азбука Морзе* була винайдена для передачі повідомлень за допомогою сигналів радіохвиль. Його суть у коротких(довгих) сигналах та паузах між ними. Короткий сигнал – точка; довгий сигнал – тире. За одиницю часу сприймається одна точка. Довжина тире – 3 точки, пауза між елементами одного знаку – 1, між знаками у слові – 3, між словами – 7 точок. Точка і тире + амперсанд (&, для зручного розділу слів) складають алфавіт Морзе у цій програмі. Алгоритм стандартно працює з масивами та ставить у відповідність символам на Морзе англійським літерам, і навпаки.

«*Трансліт*» взагалі означає транскрипцію звуків однієї мови літерами іншої, але в даному випадку це відповідність російських літер з англійськими на клавіатурі. Реалізовано також за допомогою масивів.

# 3 АЛГОРИТМИ ОБРАНИХ МЕТОДІВ

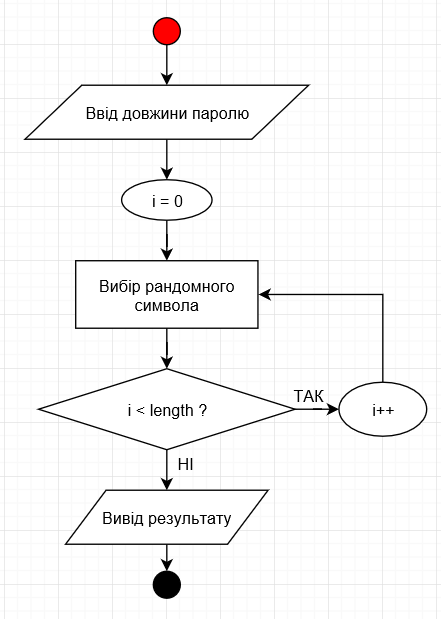


Рисунок 1. Генератор паролів



Рисунок 2. Шифр Цезаря

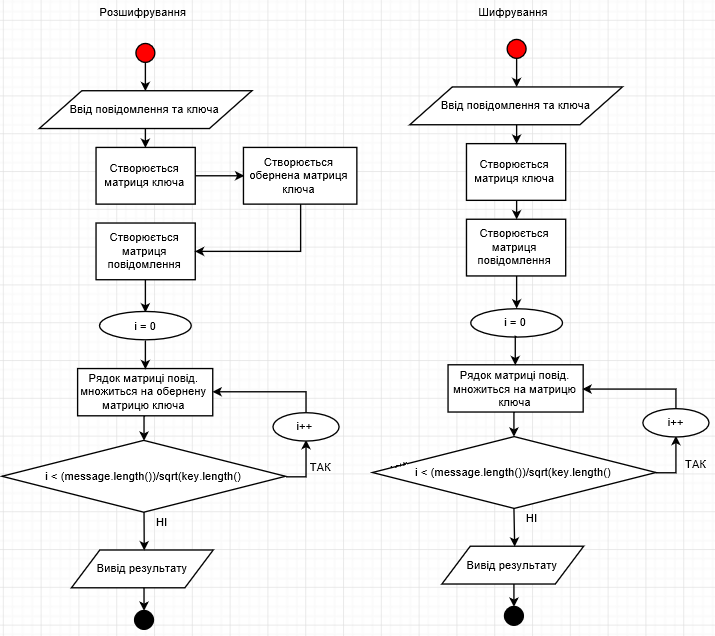


Рисунок 3. Шифр Хілла

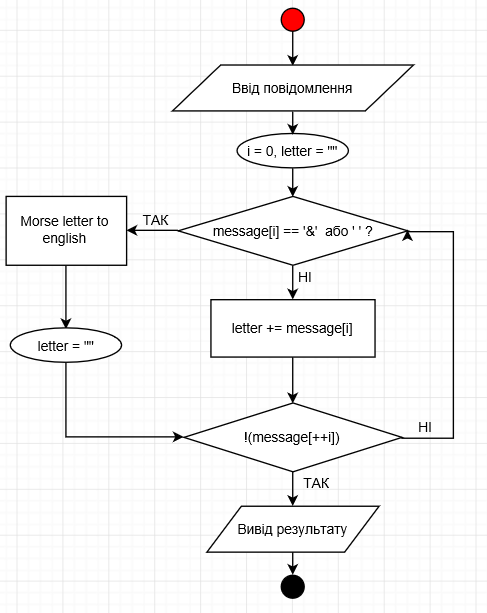


Рисунок 4. Азбука Морзе до англ.

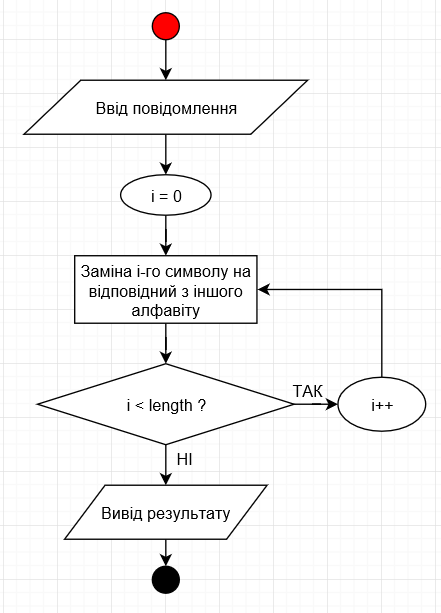


Рисунок 5. Трансліт

# 4 ОПИС ПРОГРАМИ

Програма починається з того, що з’являється головне меню, з якого є доступ до всіх алгоритмів, інформації про програму та виходу з програми. У кожному з алгоритмів у полі вводу накладено маску – задано свою множину значень, що може натиснути користувач (на інші клавіші реагувати не буде). Наприклад, *QRegExp dig( "^[0-9]\*$" ) –* для вводу цифр. У всіх класах є інформація про даний алгоритм та кнопка повернення у головне меню.

Також всі класи пов’язані з файлом *“ciphers.h”,* що вказує на функції файлу *“ciphers.cpp”.* Це склад усіх реалізованих алгоритмів, що необхідні для функціонування програми.

Натиснувши на першу кнопку – програма перейде у вікно алгоритму **генерації паролей**, що реалізовано завдяки псевдо-випадковим числам. (ф-ї rand() та srand(), що задає параметр time\_t). Користувач може ввести довжину пароля, після чого отримати результат. Цей результат можна зберегти у необхідний файл. При натисканні кнопки «Зберегти» користувачеві випаде системне вікно з можливістю перейти до необхідної директорії та обиранням файлу. Реалізовано за допомогою функції та однойменної бібліотеки для роботи з директоріями *QFileDialog*. За бажанням користувач може приписати назву сервісу, до якого буде призначено цей пароль.

Другим алгоритмом є шифр Цезаря. Його реалізовано за допомогою роботи з ASCII-кодом. Він оперує з строкою типу *QString*, що є типовою тільки для QT Creator. У цьому вікні буде можливість ввести повідомлення для шифрування та крок, на який буде переміщатися кожен символ в заданому алфавіті. Ще є кнопки опису алгоритму на прикладу у вигляді картинки – реалізовано за допомогою *QMessageBox* та *QPixmap*. Також необхідно було перед цим додати цей рисунок до файлу ресурсів QT, щоб програма могла сприймати.

Алгоритмом під номером 3 є шифр Хілла. Він працює з матрицями. У вікні цього алгоритму буде можливість обрати між шифруванням / дешифруванням повідомлення за допомогою ключа, які потрібно ввести. Тут також є інформація та приклад у вигляді рисунку, якщо користувачеві щось буде незрозуміло.

Четвертим алгоритмом є Азбука Морзе. Тут також можна продивитися алфавіт у вигляді картинки для ознайомлення, та ввести своє повідомлення на Азбуці Морзе(англійському) для перекладу на англ.(Азб. Морзе). За допомогою типу *QVector <QString>* програма порівнює введені символи на Морзе з табличними значеннями. Вектори було обрано через зручність експлуатації, в порівнянні зі звичайним масивом. Для них спеціально було додано підтримку c++11, бо нові версії не підтримують такий запис вектора: {"…"}.

Класом під номером 5 є «трансліт». Він реалізовує відповідний перехід між англійською та російською розкладкою. Тобто, при введенні англійських літер в результаті отримаєте російські і навпаки. Для візуалізації наведено приклад у вигляді клавіатури.

# 5 РЕЗУЛЬТАТ ПРОГРАМИ

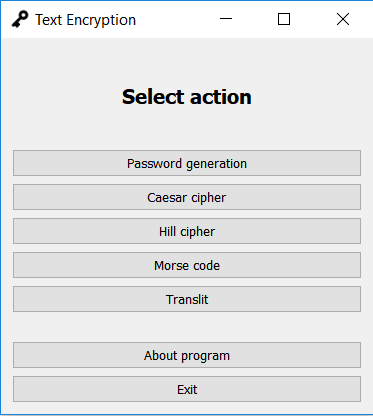


Рисунок 6. Головне меню

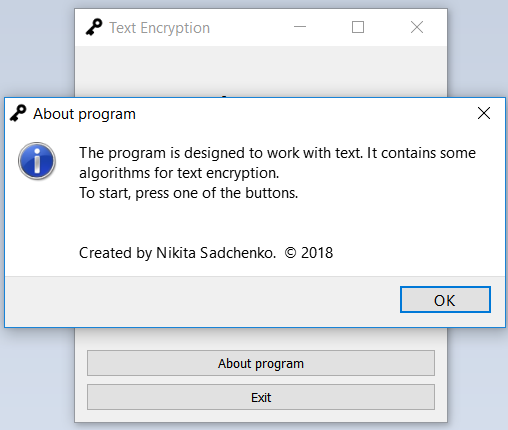


Рисунок 7. Інформація про програму

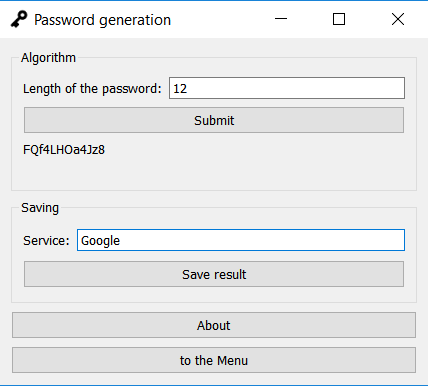


Рисунок 8

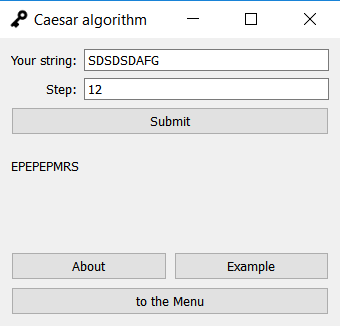


Рисунок 9

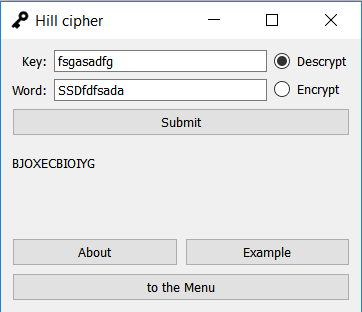


Рисунок 10

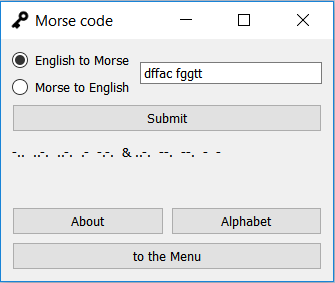


Рисунок 11

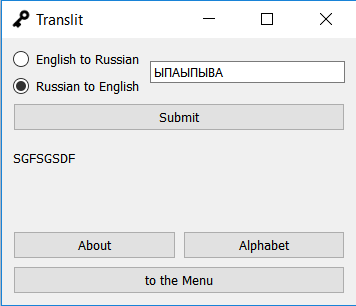


Рисунок 12

# ВИСНОВОК

Отже, шифрування тексту вже дуже давно увійшло в наше життя та без нього ми б не мали ні онлайн платежів, ні мессенджерів, ні взагалі інтернету, бо все це потребує ключі шифрування для функціонування.

Розглянені алгоритми (мається на увазі шифри Цезаря та Хілла) в свій час поклали початок всьому, що ми маємо зараз з технологічного світу.

Ця програма створена для того, щоб ознайомити користувача з деякими алгоритмами шифрування. Також її можна позицініювати як менеджер паролів, незамінну на сьогодні річ.

# ЛІТЕРАТУРА

1. Костерин В.В. Камаев В.А. “Технологии программирования: Учебник для вузов”
2. Герберт Шилдт. “Полный справочник по С++. Четвёртое издание”
3. Жельников В. [Криптография от папируса до компьютера](http://cins.ict.nsc.ru/citonod/My/Crypto/zhelnik.html) / Под ред. А. Б. Васильева. — М.: ABF, 1996. — 335 с.
4. [Мао В.](https://www.wikidata.org/wiki/Q21704656) [Современная криптография](https://www.wikidata.org/wiki/Q21704663): Теория и практика — М.: [Вильямс](https://www.wikidata.org/wiki/Q21694521), 2005. — 768 с.

# ДОДАТОК А (ТЕКСТ ПРОГРАМИ НА МОВІ С++)

Файл kursach3.pro

#-------------------------------------------------

#

# Project created by QtCreator 2018-05-26T11:47:14

#

#-------------------------------------------------

QT += core gui

CONFIG += c++11

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

TARGET = kursach3

TEMPLATE = app

SOURCES += main.cpp\

mainwindow.cpp\

ciphers.cpp \

caesar.cpp \

generator.cpp \

morse.cpp \

translit.cpp \

leshill.cpp

HEADERS += mainwindow.h\

ciphers.h \

caesar.h \

generator.h \

morse.h \

translit.h \

leshill.h

FORMS += mainwindow.ui \

caesar.ui \

generator.ui \

morse.ui \

translit.ui \

leshill.ui

RESOURCES += \

resources.qrc

Файл caesar.h

#ifndef CAESAR\_H

#define CAESAR\_H

#include <QMainWindow>

namespace Ui {

class Caesar;

}

class Caesar : public QMainWindow

{

Q\_OBJECT

public:

explicit Caesar(QWidget \*parent = 0);

~*Caesar*();

private slots:

void on\_algorithm\_clicked();

void on\_to\_menu\_clicked();

void on\_about\_caesar\_clicked();

void on\_example\_clicked();

private:

Ui::Caesar \*ui;

};

#endif // CAESAR\_H

Файл ciphers.h

#ifndef CIPHERS\_H

#define CIPHERS\_H

#include <QString>

QString caesar(QString str, QString step);

QString pass\_generator(QString pass\_size\_str);

QString eng\_to\_morse(QString str);

QString morse\_to\_eng(QString str);

QString transl\_ruseng(QString str);

QString transl\_engrus(QString str);

QString hill\_encrypt(QString key, QString word);

QString hill\_descrypt(QString key, QString word);

#endif

Файл generator.h

#ifndef GENERATOR\_H

#define GENERATOR\_H

#include <QMainWindow>

namespace Ui {

class Generator;

}

class Generator : public QMainWindow

{

Q\_OBJECT

public:

explicit Generator(QWidget \*parent = 0);

~*Generator*();

//signals:

// void sendData(QString str);

private slots:

void on\_algorithm\_clicked();

void on\_to\_menu\_clicked();

void on\_saving\_clicked();

void on\_about\_generator\_clicked();

private:

Ui::Generator \*ui;

};

#endif // GENERATOR\_H

Файл leshill.h

#ifndef LESHILL\_H

#define LESHILL\_H

#include <QMainWindow>

namespace Ui {

class LesHill;

}

class LesHill : public QMainWindow

{

Q\_OBJECT

public:

explicit LesHill(QWidget \*parent = 0);

~*LesHill*();

private slots:

void on\_algorithm\_clicked();

void on\_to\_menu\_clicked();

void on\_about\_hill\_clicked();

void on\_example\_clicked();

private:

Ui::LesHill \*ui;

};

#endif // LESHILL\_H

Файл mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

private slots:

void on\_generator\_clicked();

void on\_caesar\_clicked();

void on\_morse\_clicked();

void on\_exit\_clicked();

void on\_translit\_clicked();

void on\_leshill\_clicked();

void on\_about\_program\_clicked();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

Файл morse.h

#ifndef MORSE\_H

#define MORSE\_H

#include <QMainWindow>

namespace Ui {

class Morse;

}

class Morse : public QMainWindow

{

Q\_OBJECT

public:

explicit Morse(QWidget \*parent = 0);

~*Morse*();

private slots:

void on\_eng\_to\_mor\_clicked();

void on\_mor\_to\_eng\_clicked();

void on\_algorithm\_clicked();

void on\_to\_menu\_clicked();

void on\_about\_morse\_clicked();

void on\_example\_clicked();

private:

Ui::Morse \*ui;

};

#endif // MORSE\_H

Файл translit.h

#ifndef TRANSLIT\_H

#define TRANSLIT\_H

#include <QMainWindow>

namespace Ui {

class Translit;

}

class Translit : public QMainWindow

{

Q\_OBJECT

public:

explicit Translit(QWidget \*parent = 0);

~*Translit*();

private slots:

void on\_eng\_to\_rus\_clicked();

void on\_rus\_to\_eng\_clicked();

void on\_algorithm\_clicked();

void on\_to\_menu\_clicked();

void on\_about\_translit\_clicked();

void on\_example\_clicked();

private:

Ui::Translit \*ui;

};

#endif // TRANSLIT\_H

Файл caesar.cpp

#include "caesar.h"

#include "ui\_caesar.h"

#include "ciphers.h"

#include "mainwindow.h"

#include <QMessageBox>

Caesar::Caesar(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::Caesar)

{

ui->setupUi(this);

this->setWindowTitle("Caesar algorithm");

this->setWindowIcon(QIcon(":/res/img/key.png"));

QRegExp let( "^[A-Za-z ]\*$" );

QValidator \*valid\_str = new QRegExpValidator(let, this);

ui->string->setValidator( valid\_str );

QRegExp dig( "^[0-9-]{1}[0-9]\*$" );

QValidator \*valid\_int = new QRegExpValidator(dig, this);

ui->step->setValidator( valid\_int );

}

Caesar::~*Caesar*()

{

delete ui;

}

void Caesar::on\_algorithm\_clicked()

{

QString str = ui->string->text();

QString step = ui->step->text();

QString result = caesar(str, step);

ui->label\_result->setText(result);

ui->label\_result->setTextInteractionFlags(Qt::TextSelectableByMouse);

}

void Caesar::on\_to\_menu\_clicked()

{

hide();

MainWindow \*mainWindow = new MainWindow();

mainWindow->show();

}

void Caesar::on\_about\_caesar\_clicked()

{

QMessageBox info(QMessageBox::NoIcon, "About Caesar cipher",

"Caesar cipher is one of the simplest and most widely known encryption techniques. "\

"It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. "\

"For example, with a left shift of 3, D would be replaced by A, E would become B, and so on.\n"\

"The method is named after Julius Caesar, who used it in his private correspondence.\n\n"\

"For using, enter your message, step and confirm.",

QMessageBox::Ok, this);

info.setTextInteractionFlags(Qt::TextSelectableByMouse);

info.*exec*();

}

void Caesar::on\_example\_clicked()

{

QMessageBox exam;

exam.setIconPixmap(QPixmap(":/res/img/caesar.png"));

exam.setWindowTitle("Example of Caesar cipher");

exam.show();

exam.*exec*();

}

Файл ciphers.cpp

#include <iostream>

#include <windows.h> // cyrillic

#include <cstdlib> // rand()

#include <time.h> // time\_t

#include <cmath> // pow()

#include <string> // memset()

#include <QString>

#include <QVector>

#include "ciphers.h"

#define ENG\_ALPH 26

#define ASCII\_A (int)'A' // 65

#define ASCII\_Z (int)'Z' // 90

#define ASCII\_a (int)'a' // 97

#define ASCII\_z (int)'z' // 122

//---------------------------------------------------------------------------------

/\* Caesar Cipher \*/

QString caesar(QString str, QString step) {

int step\_int = step.toInt();

int ascii\_str;

int differ1, differ2;

QString result;

int counter = 0;

int str\_len = str.length();

for (int i = 0; i < str\_len; ++i) {

if (str[i] == ' ') {

result[counter++] = ' ';

} else {

ascii\_str = str[i].toLatin1();

if ((ascii\_str >= ASCII\_A) && (ascii\_str <= ASCII\_Z)) { // UPPER case

differ1 = ASCII\_Z - ascii\_str;

differ2 = ascii\_str - ASCII\_A;

if (step\_int > differ1) {

result[counter++] = (char)(ASCII\_A + ((step\_int - (differ1 + 1)) % ENG\_ALPH));

} else if (-step\_int > differ2) {

result[counter++] = (char)(ASCII\_Z + ((step\_int + (differ2 + 1)) % ENG\_ALPH));

} else {

result[counter++] = (char)(ascii\_str + step\_int);

}

}

else if ((ascii\_str >= ASCII\_a) && (ascii\_str <= ASCII\_z)) { // LOWER case

differ1 = ASCII\_z - ascii\_str;

differ2 = ascii\_str - ASCII\_a;

if (step\_int > differ1) {

result[counter++] = (char)(ASCII\_a + ((step\_int - (differ1 + 1)) % ENG\_ALPH));

} else if (-step\_int > differ2) {

result[counter++] = (char)(ASCII\_z + ((step\_int + (differ2 + 1)) % ENG\_ALPH));

} else {

result[counter++] = (char)(ascii\_str + step\_int);

}

}

}

}

return result;

}

//---------------------------------------------------------------------------------

/\* English to Morse \*/

QString eng\_to\_morse(QString str) {

char alpha[ENG\_ALPH] = {'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};

QString morse[ENG\_ALPH] = {".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..", "--", "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-", "...-", ".--", "-..-", "-.--", "--.."};

QString result = "";

int str\_len = str.length();

for (int i = 0; i < str\_len; ++i) {

if ((str[i] == ' ') && (str[i+1] != ' ') && (str[i+1] != '\0'))

result = result + "& ";

else if (str[i] != ' ') {

for (int j = 0; j < ENG\_ALPH; ++j) {

if (str[i].toUpper() == alpha[j]) {

result = result + morse[j] + " ";

break;

}

}

}

}

return result;

}

//---------------------------------------------------------------------------------

/\* Morse to English \*/

QString morse\_to\_eng(QString str) { // added c++11 for syntax of QVector initilization

QVector <QString> morse = {".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..", "--", "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-", "...-", ".--", "-..-", "-.--", "--.."};

char alpha[ENG\_ALPH] = {'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};

QVector <QString> morse2 = {""};

QString result;

int counter = 0;

int str\_len = str.length();

for (int i = 0; i < str\_len; ++i) {

if ((str[i] == ' ') || (str[i] == '&') || (str[i+1] == '\0')) {

if ((str[i+1] == '\0') && (str[i] != ' ') && (str[i] != '&'))

morse2[0] = morse2[0] + str[i];

for (int j = 0; j < ENG\_ALPH; j++) {

if (morse2[0] == morse[j])

result[counter++] = alpha[j];

}

morse2[0] = "";

if (str[i] == '&')

result[counter++] = ' ';

}

else morse2[0] = morse2[0] + str[i];

}

return result;

}

//---------------------------------------------------------------------------------

/\* russian to english \*/

QString transl\_ruseng(QString str) {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

QString eng[ENG\_ALPH] = {"q","w","e","r","t","y","u","i","o","p","a","s","d","f","g","h","j","k","l","z","x","c","v","b","n","m"};

int rus[ENG\_ALPH] = {233,246,243,234,229,237,227,248,249,231,244,251,226,224,239,240,238,235,228,255,247,241,236,232,242,252};

// char rus[ENG\_ALPH] = {'й','ц','у','к','е','н','г','ш','щ','з','ф','ы','в','а','п','р','о','л','д','я','ч','с','м','и','т','ь'};

QString result = "";

str = str + ".";

int str\_len = str.length();

for (int i = 0; i < str\_len; i++) {

if ((str[i] == ' ') && (str[i-1] != ' '))

result = result + ' ';

else {

// cyrillic to ASCII

QString str\_low(str[i].toLower());

QByteArray byte\_str = str\_low.toLocal8Bit();

int ascii\_str = static\_cast<unsigned char>(byte\_str[0]);

for (int j = 0; j < ENG\_ALPH; j++) {

// if ((char)(tolower(str[i])) == (char)rus[j]) {

// if ((char)(tolower(str[i])) == str[i]) std::cout << eng[j];

if (ascii\_str == rus[j]) {

if (str[i].toLower() == str[i]) {

result = result + eng[j];

}

else result = result + eng[j].toUpper();

break;

}

}

}

}

return result;

}

//---------------------------------------------------------------------------------

/\* english to russian \*/

QString transl\_engrus(QString str) {

QString eng[ENG\_ALPH] = {"q","w","e","r","t","y","u","i","o","p","a","s","d","f","g","h","j","k","l","z","x","c","v","b","n","m"};

QString rus[ENG\_ALPH] = {"й","ц","у","к","е","н","г","ш","щ","з","ф","ы","в","а","п","р","о","л","д","я","ч","с","м","и","т","ь"};

QString result = "";

int str\_len = str.length();

for (int i = 0; i < str\_len; i++) {

if ((str[i] == ' ') && (str[i-1] != ' '))

result = result + ' ';

else {

for (int j = 0; j < ENG\_ALPH; j++) {

if (static\_cast<QString>(str[i]) == eng[j]) {

result = result + rus[j];

break;

}

}

}

}

return result;

}

//---------------------------------------------------------------------------------

/\* Password Generator \*/

QString pass\_generator(QString pass\_size\_str) {

int pass\_size = pass\_size\_str.toInt();

QString password;

srand(time(NULL));

int t;

for (int i = 0; i < pass\_size; ++i) {

t = rand()%3;

if (t == 0) {

password[i] = QChar(char(ASCII\_A + rand()%26));

}

else if (t == 1) {

password[i] = QChar(char(ASCII\_a + rand()%26));

}

else {

password[i] = QChar(char(48 + rand()%10)); // ASCII 'zero'

}

}

return password;

}

//---------------------------------------------------------------------------------

// Extended Euclidean Algorithm

int ext\_euc\_alg(int det, const int len) {

int a = det, b = len;

int ua = 1, va = 0, ub = 0, vb = 1;

int q, r, buf;

// a = ua \* det + va \* len;

// b = ub \* det + vb \* len;

while (b != 0) {

q = a/b;

r = a%b;

a = b; b = r;

buf = ub;

ub = ua - q \* ub;

ua = buf; buf = vb;

vb = va - q \* vb;

va = buf;

}

// int d = a, y = va;

int x = 0;

if ((a == -1) && (ua > 0))

x = len - ua;

else if ((ua < 0) && (det < 0))

x = -ua;

else if ((ua < 0) && (det > 0))

x = len + ua;

else if (ua > 0)

x = ua;

return x;

}

//---------------------------------------------------------------------------------

/\* Matrix Declaring \*/

int\*\* mat\_declare(int rows, int cols) {

int\*\* matrix = new int\*[rows];

for(int i = 0; i < rows; ++i)

matrix[i] = new int[cols];

return matrix;

}

//---------------------------------------------------------------------------------

/\* Matrix Filling with declaring \*/

int\*\* filling(QString message, int rows, int cols) {

QString alpha[ENG\_ALPH+3] = {"A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z"," ",",","?"};

int mess\_len = message.length();

int iter = 0;

int\*\* matrix = mat\_declare(rows, cols);

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

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

for (int k = 0; k < ENG\_ALPH+3; ++k) {

if (iter >= mess\_len)

matrix[i][j] = 26; // position of ' ' in alpha

else if (static\_cast<QString>(message[iter]).toUpper() == alpha[k])

matrix[i][j] = k;

}

++iter;

}

}

return matrix;

}

//---------------------------------------------------------------------------------

/\* Matrix Freeing \*/

void freeing(int\*\* matrix, int rows) {

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

delete [] matrix[i];

}

delete [] matrix;

}

//---------------------------------------------------------------------------------

/\* Determinant of Matrix \*/

int determinant(int\*\* matrix, int size) {

int det = 0;

if (size == 1) det = matrix[0][0];

else if (size == 2)

det = matrix[0][0]\* matrix[1][1] - matrix[1][0]\* matrix[0][1];

else if (size == 3)

det = matrix[0][0]\* matrix[1][1]\* matrix[2][2] + matrix[0][1]\* matrix[1][2]\* matrix[2][0] +

matrix[1][0]\* matrix[2][1]\* matrix[0][2] - matrix[0][2]\* matrix[1][1]\* matrix[2][0] -

matrix[0][0]\* matrix[1][2]\* matrix[2][1] - matrix[0][1]\* matrix[1][0]\* matrix[2][2];

return det;

}

//---------------------------------------------------------------------------------

/\* Finding of Inverse Matrix \*/

int\*\* inverse\_matrix(int\*\* matrix, int size, const int alpha\_length) {

int\*\* inv\_matrix = mat\_declare(size, size);

int det = determinant(matrix, size);

if (det != 0) {

int cof[size][size];

int adj[size][size];

int x = ext\_euc\_alg(det, alpha\_length);

if (size == 2) {

for (int i = 0; i < 2; ++i) {

for (int j = 0; j < 2; ++j) {

cof[i][j] = pow(-1, i+j)\* (matrix[1-i][1-j] %(ENG\_ALPH+3));

}

}

}

else if (size == 3) {

int ia,ib,ja,jb;

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

if (i+1 > 2) ib = 0;

else ib = i+1;

if (i-1 < 0) ia = 2;

else ia = i-1;

if (j+1 > 2) jb = 0;

else jb = j+1;

if (j-1 < 0) ja = 2;

else ja = j-1;

cof[i][j] = (matrix[ib][jb]\* matrix[ia][ja] - matrix[ia][jb]\* matrix[ib][ja]) %(ENG\_ALPH+3);

std::cout << "cof " << cof[i][j] << std::endl;

}

}

}

// TRANSPOSE cofactor matrix (result - adjugate matrix)

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

for (int j = i+1; j < size; ++j) {

adj[i][j] = cof[j][i];

adj[j][i] = cof[i][j];

}

adj[i][i] = cof[i][i];

}

// RESULT MATRIX

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

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

if (adj[i][j] < 0)

inv\_matrix[i][j] = (ENG\_ALPH+3) + ((adj[i][j] \* x) %(ENG\_ALPH+3));

else

inv\_matrix[i][j] = (adj[i][j] \*x) %(ENG\_ALPH+3);

}

}

}

return inv\_matrix;

}

//---------------------------------------------------------------------------------

/\* Hill Cipher (Encryption) \*/

QString hill\_encrypt(QString key, QString word) {

QString alpha[ENG\_ALPH+3] = {"A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z"," ",",","?"};

int size = sqrt(key.length());

int blocks = (word.length())/size;

if ((word.length())%size != 0)

++blocks;

int\*\* mat\_key = filling(key, size, size);

int\*\* mat\_word = filling(word, blocks, size);

QString result = "";

int det = determinant(mat\_key, size);

if (det != 0) {

int mat\_mult[blocks][size];

memset(mat\_mult, 0, sizeof(mat\_mult));

// MATRIX MULTIPLICATION + DECODING

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

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

for (int k = 0; k < size; ++k) {

mat\_mult[i][j] = (mat\_mult[i][j] + (mat\_word[i][k] \* mat\_key[k][j])) %(ENG\_ALPH+3);

}

result = result + alpha[ mat\_mult[i][j] ];

}

}

} else {

result = "error: det=0";

}

freeing(mat\_key, size);

freeing(mat\_word, blocks);

return result;

}

//---------------------------------------------------------------------------------

/\* Hill Cipher (Descryption) \*/

QString hill\_descrypt(QString key, QString word) {

QString alpha[ENG\_ALPH+3] = {"A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z"," ",",","?"};

int size = sqrt(key.length());

int blocks = (word.length())/size;

if ((word.length())%size != 0)

++blocks;

int\*\* mat\_key = filling(key, size, size);

int\*\* mat\_word = filling(word, blocks, size);

int\*\* inv\_mat\_key = inverse\_matrix(mat\_key, size, ENG\_ALPH+3);

QString result = "";

int det = determinant(mat\_key, size);

if (det != 0) {

int mat\_mult[blocks][size];

memset(mat\_mult, 0, sizeof(mat\_mult));

// MATRIX MULTIPLICATION + DECODING

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

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

for (int k = 0; k < size; ++k) {

mat\_mult[i][j] = (mat\_mult[i][j] + (mat\_word[i][k] \* inv\_mat\_key[k][j])) %(ENG\_ALPH+3);

}

result = result + alpha[ mat\_mult[i][j] ];

}

}

} else {

result = "error: det=0";

}

freeing(mat\_key, size);

freeing(mat\_word, blocks);

freeing(inv\_mat\_key, size);

return result;

}

Файл generator.cpp

#include "generator.h"

#include "ui\_generator.h"

#include "mainwindow.h"

#include "ciphers.h"

#include <iostream>

#include <QMessageBox>

#include <QFile>

#include <QFileDialog> // choosing directory

Generator::Generator(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::Generator)

{

ui->setupUi(this);

this->setWindowTitle("Password generation");

this->setWindowIcon(QIcon(":/res/img/key.png"));

QRegExp dig( "^[0-9]\*$" );

QValidator \*valid\_int = new QRegExpValidator(dig, this);

ui->length->setValidator( valid\_int );

}

Generator::~*Generator*()

{

delete ui;

}

void Generator::on\_algorithm\_clicked()

{

QString pass\_length = ui->length->text();

QString result = pass\_generator(pass\_length);

ui->label\_result->setText(result);

ui->label\_result->setTextInteractionFlags(Qt::TextSelectableByMouse);

}

void Generator::on\_to\_menu\_clicked()

{

hide();

MainWindow \*mainWindow = new MainWindow();

mainWindow->show();

}

void Generator::on\_saving\_clicked()

{

QString dir = QFileDialog::getOpenFileName(this,

tr("Open Directory"),

"D:/",

"All files (\*.\*);;Text files (\*.txt);;Doc files (\*.doc);;Docx files (\*.docx);;Fb2 files(\*.fb2)");

QFile file(dir);

QByteArray buff;

int num = 1;//, sensor = 0, buff\_num;

if (file.*open*(QIODevice::ReadOnly | QIODevice::Text)) {

while(!file.*atEnd*()) {

// buff\_num = 0;

// sensor = 1;

buff = file.readLine();

// if ((buff != "\r\n") && (buff != "\n") && (buff != ""))

num++;

// else buff\_num = 1;

}

file.*close*();

}

// else QMessageBox::warning(this, "Some Error", "File didn't opened...");

if (file.*open*(QIODevice::Append | QIODevice::Text)) {

QString pass = ui->label\_result->text();

QString service = ui->service->text();

if (pass != "<center>result</center>") {

if (service != "") {

// if ((buff\_num == 1) || (sensor == 0)) {

file.write((QString::number(num) + ". " + service + " — " + pass + "\r\n").toUtf8());

// } else {

// file.write(("\r\n" + QString::number(num) + ". " + service + " — " + pass + "\r\n").toUtf8());

// }

} else {

// if ((buff\_num == 1) || (sensor == 0)) {

file.write((QString::number(num) + ". " + pass + "\r\n").toUtf8()); // QString to const char\* (toUtf8)

// } else {

// file.write(("\r\n" + QString::number(num) + ". " + pass + "\r\n").toUtf8());

// }

}

}

file.*close*();

}

else QMessageBox::warning(this, "Some Error", "File didn't opened...");

}

void Generator::on\_about\_generator\_clicked()

{

QMessageBox::information(this, "About password generator", "Algorithm generates a password due to pseudo-random numbers.\n"\

"For using, enter length of a password and confirm.");

}

Файл leshill.cpp

#include "leshill.h"

#include "ui\_leshill.h"

#include "ciphers.h"

#include "mainwindow.h"

#include <QMessageBox>

LesHill::LesHill(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::LesHill)

{

ui->setupUi(this);

this->setWindowTitle("Hill cipher");

this->setWindowIcon(QIcon(":/res/img/key.png"));

QRegExp let( "^[?A-Za-z, ]\*$" );

QValidator \*valid\_int = new QRegExpValidator(let, this);

ui->key->setValidator( valid\_int );

ui->word->setValidator( valid\_int );

}

LesHill::~*LesHill*()

{

delete ui;

}

void LesHill::on\_algorithm\_clicked()

{

QString key = ui->key->text();

int key\_len = key.length();

QString word = ui->word->text();

QString result = "result";

if (((ui->encrypt->isChecked()) || (ui->descrypt->isChecked())) && (word != "") && (key != "")) {

if ((key\_len == 4) || (key\_len == 9)) {

if ((ui->encrypt->isChecked()) && (word != "") && (key != ""))

result = hill\_encrypt(key, word);

else if ((ui->descrypt->isChecked()) && (word != "") && (key != ""))

result = hill\_descrypt(key, word);

if (result == "error: det=0")

QMessageBox::warning(this, "Some Error", "Determinant of key matrix = 0.");

else

ui->label\_result->setText(result);

ui->label\_result->setTextInteractionFlags(Qt::TextSelectableByMouse);

}

else QMessageBox::warning(this, "Some Error", "Length of key must be equal to 4 or 9.");

}

else QMessageBox::warning(this, "Some Error", "All fields must be filled.");

}

void LesHill::on\_to\_menu\_clicked()

{

hide();

MainWindow \*mainWindow = new MainWindow();

mainWindow->show();

}

void LesHill::on\_about\_hill\_clicked()

{

QMessageBox info(QMessageBox::NoIcon, "About Hill cipher",

"Hill cipher is a polygraphic substitution cipher based on linear algebra. "\

"Invented by Lester S. Hill in 1929, it was the first polygraphic cipher in which it was practical (though barely) to operate on more than three symbols at once. "\

"The following discussion assumes an elementary knowledge of matrices.\n\n"\

"You can write key and message with letters of English alphabet + ' ' + ',' + '?'. This is done in order to get prime number (29) — length of new alphabet. "\

"Prime number is more difficult to decipher. Key and word is transformed to matrix of letter positions in alphabet.\n"\

"For encryption blocks(rows) of matrix\_word multiplicates to key\_matrix.\n"\

"For descryption blocks(rows) of matrix\_word multiplicates to inverse\_key\_matrix.\n\n"\

"For using enter key and your message to encrypt/descrypt.",

QMessageBox::Ok, this);

info.setTextInteractionFlags(Qt::TextSelectableByMouse);

info.*exec*();

}

void LesHill::on\_example\_clicked()

{

QMessageBox exam;

exam.setIconPixmap(QPixmap(":/res/img/hill.jpg"));

exam.setWindowTitle("Hill cipher — image");

exam.show();

exam.*exec*();

}

Файл main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

Файл mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "ciphers.h"

#include "generator.h"

#include "caesar.h"

#include "morse.h"

#include "translit.h"

#include "leshill.h"

#include <QMessageBox>

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

this->setWindowTitle("Text Encryption");

this->setWindowIcon(QIcon(":/res/img/key.png"));

}

MainWindow::~*MainWindow*()

{

delete ui;

}

void MainWindow::on\_generator\_clicked()

{

hide();

Generator \*mainWindow = new Generator();

mainWindow->show();

}

void MainWindow::on\_caesar\_clicked()

{

hide();

Caesar \*mainWindow = new Caesar();

mainWindow->show();

}

void MainWindow::on\_morse\_clicked()

{

hide();

Morse \*mainWindow = new Morse();

mainWindow->show();

}

void MainWindow::on\_exit\_clicked()

{

close();

}

void MainWindow::on\_translit\_clicked()

{

hide();

Translit \*mainWindow = new Translit();

mainWindow->show();

}

void MainWindow::on\_leshill\_clicked()

{

hide();

LesHill \*mainWindow = new LesHill();

mainWindow->show();

}

void MainWindow::on\_about\_program\_clicked()

{

QMessageBox::information(this, "About program", "The program is designed to work with text. It contains some algorithms for text encryption.\n"\

"To start, press one of the buttons.\n\n\n"\

"Created by Nikita Sadchenko. © 2018");

}

Файл morse.cpp

#include "morse.h"

#include "ui\_morse.h"

#include "ciphers.h"

#include "mainwindow.h"

#include <QMessageBox>

Morse::Morse(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::Morse)

{

ui->setupUi(this);

this->setWindowTitle("Morse code");

this->setWindowIcon(QIcon(":/res/img/key.png"));

}

Morse::~*Morse*()

{

delete ui;

}

void Morse::on\_eng\_to\_mor\_clicked()

{

ui->string->setEnabled(true);

ui->string->clear();

QRegExp let( "^[A-Za-z ]\*$" );

QValidator \*valid\_str = new QRegExpValidator(let, this);

ui->string->setValidator( valid\_str );

}

void Morse::on\_mor\_to\_eng\_clicked()

{

ui->string->setEnabled(true);

ui->string->clear();

QRegExp let( "^[-.& ]\*$" );

QValidator \*valid\_str = new QRegExpValidator(let, this);

ui->string->setValidator( valid\_str );

}

void Morse::on\_algorithm\_clicked()

{

QString str = ui->string->text();

QString result = "result";

if (ui->eng\_to\_mor->isChecked())

result = eng\_to\_morse(str);

else if (ui->mor\_to\_eng->isChecked())

result = morse\_to\_eng(str);

ui->label\_result->setText(result);

ui->label\_result->setTextInteractionFlags(Qt::TextSelectableByMouse);

}

void Morse::on\_to\_menu\_clicked()

{

hide();

MainWindow \*mainWindow = new MainWindow();

mainWindow->show();

}

void Morse::on\_about\_morse\_clicked()

{

QMessageBox info(QMessageBox::NoIcon, "About Morse code",

"Morse code is a method of transmitting text information as a series of on-off tones, lights, or clicks that can be directly understood by a skilled listener or observer without special equipment. "\

"It is named for Samuel F. B. Morse, an inventor of the telegraph.\n"\

"Each Morse code symbol represents either a text character (letter or numeral) or a prosign and is represented by a unique sequence of dots and dashes. "\

"The dot duration is the basic unit of time measurement in code transmission. The duration of a dash is three times the duration of a dot. Each dot or dash is followed by a short silence, equal to the dot duration. "\

"The letters of a word are separated by a space equal to three dots (one dash), and the words are separated by a space equal to seven dots.\n\n"\

"For using, enter message in English(Morse code) for encryption to Morse code(English).\n"\

"Morse code: ' ' — letter separator; '&' — word separator.\n"\

"English: ' ' — word separator.\n\n"\

"Example: [.. & -. . . -.. & .... . .-.. .--.] — [I NEED HELP]",

QMessageBox::Ok, this);

info.setTextInteractionFlags(Qt::TextSelectableByMouse);

info.*exec*();

}

void Morse::on\_example\_clicked()

{

QMessageBox exam;

exam.setIconPixmap(QPixmap(":/res/img/morse.jpg"));

exam.setWindowTitle("Morse code — image");

exam.show();

exam.*exec*();

}

Файл translit.cpp

#include "translit.h"

#include "ui\_translit.h"

#include "ciphers.h"

#include "mainwindow.h"

#include <QMessageBox>

Translit::Translit(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::Translit)

{

ui->setupUi(this);

this->setWindowTitle("Translit");

this->setWindowIcon(QIcon(":/res/img/key.png"));

}

Translit::~*Translit*()

{

delete ui;

}

void Translit::on\_eng\_to\_rus\_clicked()

{

ui->string->setEnabled(true);

ui->string->clear();

QRegExp let( "^[A-Za-z ]\*$" );

QValidator \*valid\_str = new QRegExpValidator(let, this);

ui->string->setValidator( valid\_str );

}

void Translit::on\_rus\_to\_eng\_clicked()

{

ui->string->setEnabled(true);

ui->string->clear();

QRegExp let( "^[А-Яа-я ]\*$" );

QValidator \*valid\_str = new QRegExpValidator(let, this);

ui->string->setValidator( valid\_str );

}

void Translit::on\_algorithm\_clicked()

{

QString str = ui->string->text();

QString result = "result";

if (ui->eng\_to\_rus->isChecked())

result = transl\_engrus(str);

else if (ui->rus\_to\_eng->isChecked())

result = transl\_ruseng(str);

ui->label\_result->setText(result);

ui->label\_result->setTextInteractionFlags(Qt::TextSelectableByMouse); // чтобы можно бб выделить текст

}

void Translit::on\_to\_menu\_clicked()

{

hide();

MainWindow \*mainWindow = new MainWindow();

mainWindow->show();

}

void Translit::on\_about\_translit\_clicked()

{

QMessageBox info(QMessageBox::NoIcon, "About translit",

"This algorithm change russian(english) letters to english(russian) letters with positions of it on keyboard.\n\n"\

"For using, enter your message and choose action (rus\_to\_eng or eng\_to\_rus).",

QMessageBox::Ok, this);

info.setTextInteractionFlags(Qt::TextSelectableByMouse);

info.*exec*();

}

void Translit::on\_example\_clicked()

{

QMessageBox exam;

exam.setIconPixmap(QPixmap(":/res/img/translit.png"));

exam.setWindowTitle("Keyboard — image");

exam.show();

exam.*exec*();

}