Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Кафедра «Информационных технологий и систем»

**Лабораторная работа №4**   
по дисциплине:

«Защита информации»

Разработал:

Студент группы 8091

Васильев И.В.  
 «\_\_» \_\_\_\_\_2021г

Проверил:

Жгун Т. В. \_\_\_\_\_\_

«\_\_» \_\_\_\_\_2021г

**Великий Новгород**

**2021**

**Задача:**

Зашифровать фрагмент текста объемом около 500 символов (допустим диапазон 500-1000 символов). Использовать следующие способы шифрования: ECB, CBC, OFB, CFB.   
Построить гистограммы для полученных зашифрованных текстов. Проанализировать результат (четыре гистограммы).

**Описание алгоритмов:**

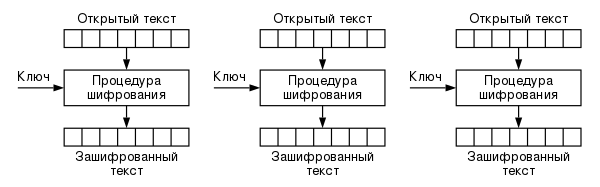
**Electronic Codebook(ECB)**

В ГОСТ 28147—89 этот режим называется **режимом простой замены**.

*Шифрование:*

Пусть дано сообщение **{\displaystyle P}P** (открытый текст, последовательность бит, данные).

Во время шифрования выполняются следующие действия:

[](https://commons.wikimedia.org/wiki/File:ECB_Encryption_ru.svg?uselang=ru)

Шифрование в режиме ECB (режиме электронной кодовой книги)

1. Сообщение делится на блоки одинакового размера. Размер (длина) блока равен ***n*** и измеряется в битах. В результате получается последовательность блоков {\displaystyle P\_{1},P\_{2},...,P\_{q}}**P1, P2, P3, … Pq** Последний блок при необходимости дополняется до длины ***n***../.
2. Каждый блок {\displaystyle P\_{i}}**Pi** шифруется алгоритмом шифрования {\displaystyle E\_{k}}**Ek**с использованием ключа *k*:

 {\displaystyle C\_{i}=E\_{k}\left(P\_{i},k\right),}

где:

* *i* — номер блока;
* *k* — [ключ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F));
* {\displaystyle P\_{i}}*Pi*— блок сообщения (открытый текст);
* {\displaystyle C\_{i}}*Ci* — зашифрованный блок (шифротекст);
* {\displaystyle E\_{k}}*Ek* — функция, выполняющая блочное шифрование.

В результате получаются зашифрованные блоки **{\displaystyle C\_{1},C\_{2},...,C\_{q}}C1, C2, C3, … Cq**.

*Расшифровка:*

выполняется функцией {\displaystyle D\_{k}}Dk с использованием того же ключа *k*:

 {\displaystyle P\_{i}=D\_{k}\left(C\_{i},k\right).}

Особенности:

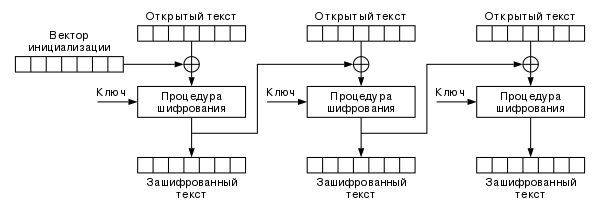
* каждый блок шифруется/расшифровывается независимо от других блоков.

Недостатки ECB:

* сохранение статистических особенностей [открытого текста](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) (поскольку одинаковым блокам [шифротекста](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82) соответствуют одинаковые блоки [открытого текста](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82)).

### Cipher Block Chaining (CBC)

Для шифрования некоторого сообщения {\displaystyle P}**P** выполняются следующие действия.

[](https://commons.wikimedia.org/wiki/File:CBC_Encryption_ru.svg?uselang=ru)

Шифрование в режиме CBC (режиме сцепления блоков шифротекста)

* Сообщение разбивается на блоки одинакового размера. Размер (длина) блока равен ***n*** и измеряется в битах. При необходимости последний блок дополняется до длины {\displaystyle n}***n***../.
* Шифрование очередного (*i*-го) блока сообщения ({\displaystyle P\_{i}}**Pi**) выполняется с использованием предыдущего зашифрованного (*(i-1)*-го) блока ({\displaystyle C\_{i-1}}**Сi-1**). Для первого блока ({\displaystyle P\_{1}}**Pi**) зашифрованного блока ({\displaystyle C\_{0}}**Ci-1**) не существует, поэтому первый блок шифруют с использованием «вектора инициализации»

{\displaystyle C\_{0}=IV}  (вектор инициализации — случайное число)

Размер (длина) IV равна размеру блока (*n*).

* В функцию шифрования {\displaystyle E\_{k}}**Ek** передаётся [сумма по модулю 2 («⊕», «xor»)](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_2) текущего блока сообщения {\displaystyle P\_{i}}**Pi**и предыдущего зашифрованного блока {\displaystyle C\_{i-1}}**Ci-1**:

{\displaystyle C\_{i}=E\_{k}\left(P\_{i}\oplus C\_{i-1},k\right),} 

где:

* *i* — номер блока;
* *k* — [ключ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F));
* *IV* — вектор инициализации ([синхропосылка](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BF%D0%BE%D1%81%D1%8B%D0%BB%D0%BA%D0%B0&action=edit&redlink=1));
* {\displaystyle P\_{i}} *Pi* — блок сообщения ([открытый текст](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82));
* {\displaystyle C\_{i-1}}*Сi-1* — зашифрованный блок ([шифротекст](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82)), полученный на предыдущем шаге шифрования;
* {\displaystyle E\_{k}}*Ek* — функция, выполняющая [блочное шифрование](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D1%87%D0%BD%D0%BE%D0%B5_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

Расшифровка выполняется функцией **{\displaystyle D\_{k}}Dk** с использованием тех же ключа *k* и вектора инициализации *IV*:

{\displaystyle C\_{0}=IV}{\displaystyle P\_{i}=C\_{i-1}\oplus D\_{k}\left(C\_{i},k\right)}

Недостатки CBC:

* возможность определения начала изменения данных по изменению [шифротекста](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82) (если сравнить шифротексты двух сообщений с одним и тем же ключом, то номер первого блока, в котором шифротексты различаются, будет соответствовать номеру первого блока, в котором различаются исходные сообщения);
* возможность изменения [открытого текста](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) при перемещении блоков
* возможность изменения блока [шифротекста](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82) {\displaystyle C\_{i-1}} **Сi-1** путём изменения блока [сообщения](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) **Pi** {\displaystyle P\_{i}};
* невозможность [распараллеливания](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC) шифрования (поскольку для шифрования каждого *i*-го блока требуется блок, зашифрованный на предыдущем шаге (блоки связаны между собой))[[1]](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B6%D0%B8%D0%BC_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F#cite_note-six-1).

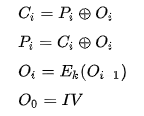
Достоинства CBC:

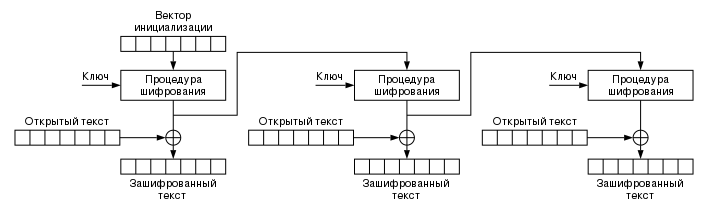
* постоянная скорость обработки блоков (скорость определяется эффективностью реализации шифра; время выполнения операции «[xor](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_2)» пренебрежимо мало);
* отсутствие статистических особенностей, характерных для режима ECB (поскольку каждый блок [открытого текста](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) «смешивается» с блоком [шифротекста](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82), полученным на предыдущем шаге шифрования);
* возможность [распараллеливания](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC) расшифровки[[1]](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B6%D0%B8%D0%BC_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F#cite_note-six-1).

### Output Feedback (OFB)

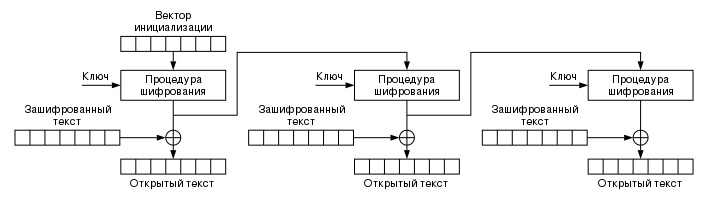
Режим (OFB) обратной связи вывода превращает блочный шифр в синхронный шифр потока: он генерирует ключевые блоки, которые являются результатом сложения с блоками открытого текста, чтобы получить зашифрованный текст. Так же, как с другими шифрами потока, зеркальное отражение в зашифрованном тексте производит зеркально отражённый бит в открытом тексте в том же самом местоположении. Это свойство позволяет многим кодам с исправлением ошибок функционировать как обычно, даже когда исправление ошибок применено перед кодированием.

Из-за симметрии операции сложения, шифрование и расшифрование похожи:

 {\displaystyle C\_{i}=P\_{i}\oplus O\_{i}}{\displaystyle P\_{i}=C\_{i}\oplus O\_{i}}{\displaystyle O\_{i}=E\_{k}(O\_{i-1})}{\displaystyle O\_{0}=IV}

[](https://commons.wikimedia.org/wiki/File:OFB_Encryption_ru.svg?uselang=ru)

Шифрование в режиме **OFB**

[](https://commons.wikimedia.org/wiki/File:OFB_Decryption_ru.svg?uselang=ru)

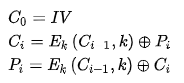
Расшифрование в режиме **OFB**

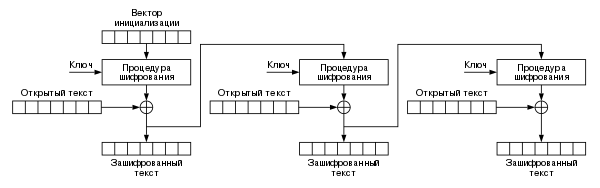
Каждая операция блочного шифра обратной связи вывода зависит от всех предыдущих и поэтому не может быть выполнена параллельно. Однако, из-за того, что открытый текст или зашифрованный текст используются только для конечного сложения, операции блочного шифра могут быть выполнены заранее, позволяя выполнить заключительное шифрование параллельно с открытым текстом.  
  
Обратная связь по выходу на k разрядов не рекомендуется из соображений криптостойкости. Режим OFB имеет следующее преимущество по сравнению с режимом CFB: ошибки, возникающие в результате передачи по каналу с шумом, при дешифровании не «размазываются» по всему шифротексту, а локализуются в пределах одного блока. Однако открытый текст может быть изменён путём определённых манипуляций с блоками шифротекста. Несмотря на то, что OFB-шифрование не поддаётся распараллеливанию, эффективность процедуры может быть повышена за счёт предварительной генерации независимой последовательности блоков.

Данный метод называется также «режим обратной связи по выходу».

### Cipher Feedback (CFB)

Во время шифрования каждый блок открытого текста складывается по модулю 2 с блоком, зашифрованным на предыдущем шаге.



[](https://commons.wikimedia.org/wiki/File:CFB_Encryption_ru.svg?uselang=ru)

Шифрование в режиме обратной связи по шифротексту

Криптостойкость CFB определяется криптостойкостью используемого шифра. Блоки открытого текста «смешиваются» («маскируются») с блоками шифротекста. Если в режиме CFB с полноблочной обратной связью имеется два идентичных блока шифротекста, результат, например, шифрования алгоритмом DES на следующем шаге будет тем же. Скорость шифрования режима CFB с полноблочной обратной связью та же, что и у блочного шифра, причём возможности распараллеливания процедуры шифрования ограничены.

**Реализация**

Поставленная задача решена с помощью языка программирования С++ на фреймворке QT.

**Описание интерфейса**

Программа предоставляет пользовательское окно, которое поделено на две вкладки:

- Работа с текстом

- Отображение гистограмм

Пользователь во вкладке работы с текстом имеет возможность ввода следующих данных:

- Текст, необходимый для анализа

- K0

- K1

- P0

- C0

Во вкладке отображение гистограмм, пользователь может увидеть работу алгоритмов шифрования в графическом виде.

Ось x – буква в 10 виде (а=0, б=1, в=2 и тд)

Ось y – количество букв.

**Код**

Код представлен только в виде основного выполняемого окна, остальные файлы, такие как (Заголовки, визуализация окна), не будут присутствовать в отчёте по функциональной ненадобности.

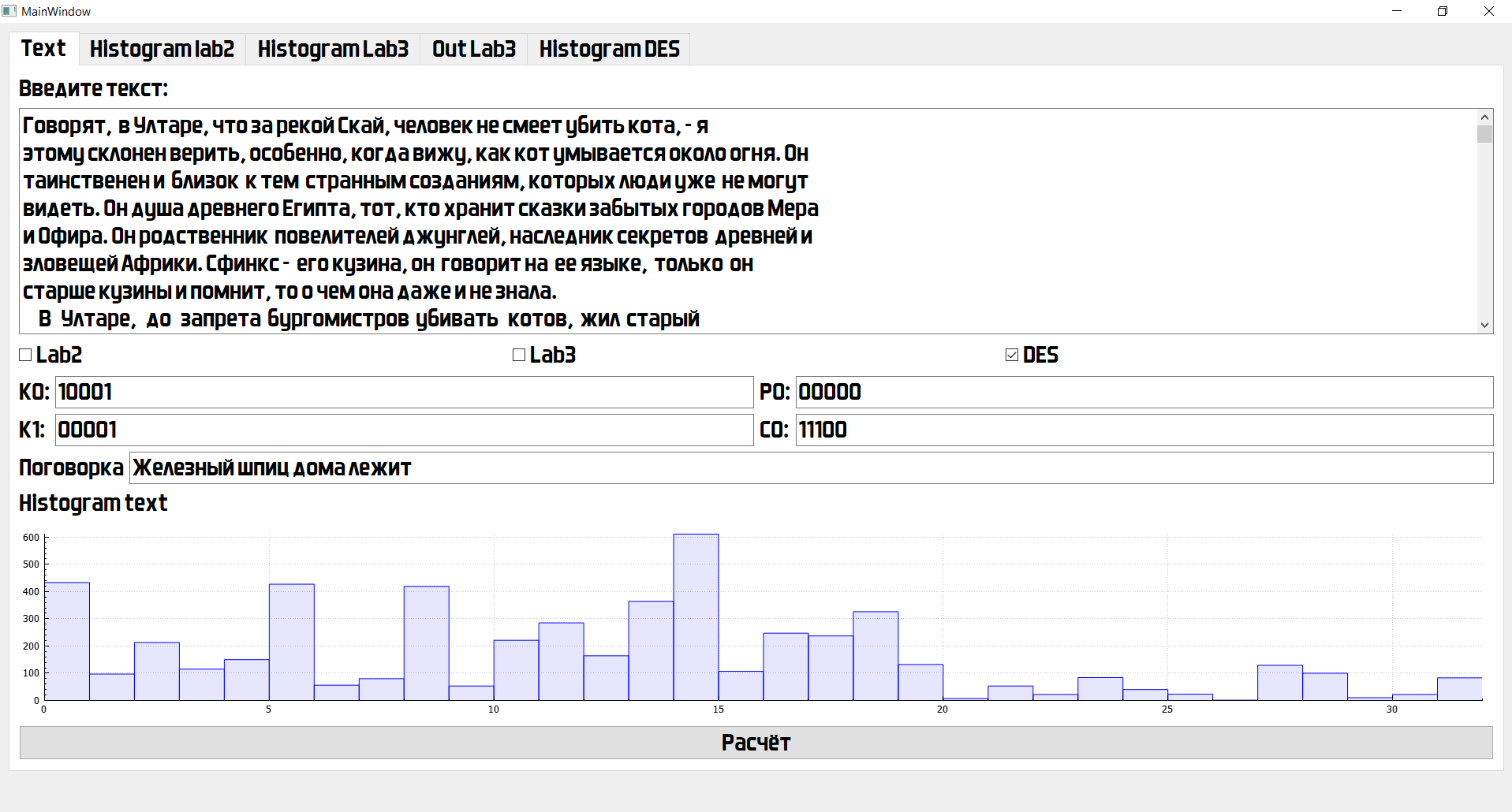
Полный код и исходные файлы (в том числе исполняемый .exe файл) можно найти в репозитории по ссылке:

https://github.com/VasilevIvanVladimirovich/Information-Security-Course/tree/main/Lab2

Запуск алгоритмов работы программы осуществляется по нажатию на кнопку “Расчёт”.

**Пример работы программы**

## Для первого примера, я взял произведение Говарда Ф. Лавкрафта “Коты Ултара” объёмом в 1013 слов.





**Вывод:**

В результате визуального анализа двух примеров, можно утверждать, что CPC и СFB шифрование лучше всего подходят, так как их распределение ближе всего совпадает с равномерным, следовательно враг будет видеть только шум.

**Код программы:**

#include "mainwindow.h"

#include "qcustomplot.h"

#include "ui\_mainwindow.h"

const int const\_num = 4;

QString debug = "";

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*), ui(new Ui::MainWindow) {

ui->setupUi(this);

ui->Round1->setReadOnly(true);

ui->Round2->setReadOnly(true);

ui->Round3->setReadOnly(true);

ui->Round4->setReadOnly(true);

ui->Round5->setReadOnly(true);

ui->Round6->setReadOnly(true);

ui->Round7->setReadOnly(true);

ui->Round8->setReadOnly(true);

}

MainWindow::~***MainWindow***() { delete ui; }

void MainWindow::**bin**(int n,

QVector<int> &bit\_array) //функция перевода числа в

//двоичную систему счисления

{

int arr[5];

for (int i = 4; i >= 0; i--) {

arr[i] = n % 2;

n = n / 2;

}

for (int i = 0; i < 5; i++) {

bit\_array.push\_back(arr[i]);

}

}

double MainWindow::**MaxValue**(QVector<int> &gen, int n) {

double max = gen[0];

for (int i = 1; i < n; i++) {

if (gen[i] > max)

max = gen[i];

}

return max;

}

double MainWindow::**MinValue**(QVector<int> &gen, int n) {

double min = gen[0];

for (int i = 1; i < n; i++) {

if (gen[i] < min)

min = gen[i];

}

return min;

}

void MainWindow::**Word**(

QString str,

QVector<int> &bit\_array) // функция перевода строки в двоичный код

{

int \*arr\_num = new int[str.length()];

QString smal[32] = {"а", "б", "в", "г", "д", "е", "ж", "з", "и", "й", "к",

"л", "м", "н", "о", "п", "р", "с", "т", "у", "ф", "х",

"ц", "ч", "ш", "щ", "ъ", "ы", "ь", "э", "ю", "я"};

QString big[32] = {"А", "Б", "В", "Г", "Д", "Е", "Ж", "З", "И", "Й", "К",

"Л", "М", "Н", "О", "П", "Р", "С", "Т", "У", "Ф", "Х",

"Ц", "Ч", "Ш", "Щ", "Ъ", "Ы", "Ь", "Э", "Ю", "Я"};

for (int i = 0; i < str.length();

i++) //перебор символов и запись их в вектор двоичных чисел

{

for (int j = 0; j < 32; j++) {

if (str[i] == smal[j]) {

arr\_num[i] = j;

bin(arr\_num[i], *bit\_array*);

break;

} else if (str[i] == big[j]) {

arr\_num[i] = j;

bin(arr\_num[i], *bit\_array*);

break;

}

}

}

delete[] arr\_num;

}

void MainWindow::**invertLastBit**(QVector<int> &out\_bit) {

int back;

if (out\_bit.back() == 0)

back = 1;

else if (out\_bit.back() == 1)

back = 0;

out\_bit.pop\_back();

out\_bit.push\_back(back);

}

void MainWindow::**sum**(QVector<int> &out\_bit, int \*buf\_array, int \*K) {

for (int i = 0; i < 5; i++) {

if ((buf\_array[i] == 1 && K[i] == 1) || (buf\_array[i] == 0 && K[i] == 0))

out\_bit.push\_back(0);

if ((buf\_array[i] == 1 && K[i] == 0) || (buf\_array[i] == 0 && K[i] == 1))

out\_bit.push\_back(1);

}

}

void MainWindow::**ECB**(QVector<int> &bit\_array, QVector<int> &ECB\_out\_bit,

int \*K0, int \*K1) {

int buf\_array[5];

for (int i = 0; i < bit\_array.length(); i += 5) {

for (int j = 0; j < 5; j++) {

buf\_array[j] = bit\_array[j + i];

}

if (buf\_array[4] == 0)

sum(*ECB\_out\_bit*, *buf\_array*, *K0*); // P1+K

else

sum(*ECB\_out\_bit*, *buf\_array*, *K1*);

invertLastBit(*ECB\_out\_bit*);

}

}

void MainWindow::**convertBinareDecimal**(QVector<int> &out\_bit,

QVector<int> &ECB\_out) {

int sum;

for (int i = 0; i < out\_bit.length(); i += 5) {

sum = 0;

for (int j = 4, k = 0; j >= 0; j--, k++) {

if (out\_bit[i + k] == 1)

sum += pow(2, j);

}

ECB\_out.push\_back(sum);

}

}

void MainWindow::**CPC**(QVector<int> &bit\_array, QVector<int> &CPC\_out\_bit,

int \*C0) {

int C[5];

for (int i = 0; i < 5; i++)

C[i] = C0[i];

int buf[5];

for (int i = 0; i < bit\_array.length(); i += 5) {

for (int j = 0; j < 5; j++) {

buf[j] = bit\_array[i + j];

}

sum(*CPC\_out\_bit*, *buf*, *C*); // P1+C0

if (CPC\_out\_bit[i] == 0)

CPC\_out\_bit[i] = 1;

else if (CPC\_out\_bit[i] == 1)

CPC\_out\_bit[i] = 0;

for (int k = 0; k < 5; k++)

C[k] = CPC\_out\_bit[k + i];

}

//Этот код написал я, Василье Иван :D !

}

void MainWindow::**sumArray**(int \*arr1, int \*arr2, int \*result) {

for (int i = 0; i < 5; i++) {

if ((arr1[i] == 1 && arr2[i] == 1) || (arr1[i] == 0 && arr2[i] == 0))

result[i] = 0;

if ((arr1[i] == 1 && arr2[i] == 0) || (arr1[i] == 0 && arr2[i] == 1))

result[i] = 1;

}

}

void MainWindow::**sumArray**(QVector<int> &arr1, QVector<int> &arr2,

QVector<int> &result) {

for (int i = 0; i < arr1.length(); i++) {

if ((arr1[i] == 1 && arr2[i] == 1) || (arr1[i] == 0 && arr2[i] == 0))

result.push\_back(0);

if ((arr1[i] == 1 && arr2[i] == 0) || (arr1[i] == 0 && arr2[i] == 1))

result.push\_back(1);

}

}

void MainWindow::**OFB**(QVector<int> &bit\_array, QVector<int> &OFB\_out\_bit,

int \*C0, int \*P0) {

int buf\_sum[5];

int bin\_word[5];

int C[5];

int P[5];

for (int i = 0; i < 5; i++)

C[i] = C0[i];

for (int i = 0; i < 5; i++)

P[i] = P0[i];

for (int i = 0; i < bit\_array.length(); i += 5) {

for (int j = 0; j < 5; j++)

bin\_word[j] = bit\_array[i + j];

sumArray(*C*, *P*, *buf\_sum*); // C0+P0

sumArray(*bin\_word*, *buf\_sum*, *C*); // P1+(C0+P0)

if (C[4] == 0)

C[4] = 1;

else if (C[4] == 1)

C[4] = 0;

for (int j = 0; j < 5; j++)

P[j] = bin\_word[j];

for (int j = 0; j < 5; j++)

OFB\_out\_bit.push\_back(C[j]);

}

}

void MainWindow::**CFB**(QVector<int> &bit\_array, QVector<int> &OFB\_out\_bit,

int \*C0) {

int C[5];

int buf[5];

for (int i = 0; i < 5; i++)

C[i] = C0[i];

if (C[0] == 0)

C[0] = 1;

else if (C[0] == 1)

C[0] = 0;

for (int i = 0; i < bit\_array.length(); i += 5) {

for (int j = 0; j < 5; j++)

buf[j] = bit\_array[i + j];

sum(*OFB\_out\_bit*, *buf*, *C*); // P1+С0

invertLastBit(*OFB\_out\_bit*);

for (int j = 0; j < 5; j++)

C[j] = OFB\_out\_bit[i + j];

}

}

bool MainWindow::**checkWordInArray**(int idexText, QString key,

QString \*arrayText) {

for (int i = 0; i < 16; i++) {

if (key[idexText] == arrayText[i] || key[idexText] == " ")

return false;

}

return true;

}

bool MainWindow::**checkWordInAlph**(int indexAlph, QString \*alph,

QString \*arrayText) {

for (int i = 0; i < 16; i++) {

if (alph[indexAlph] == arrayText[i])

return false;

}

return true;

}

void MainWindow::**createNewAlph**(QString key, QString \*alph) {

QString arrayText[16] = {0};

QString arrayLast[16] = {0};

QString lastWord;

for (int idexText = 0, indexArrey = 0; indexArrey < 16;

idexText++) // алгоритм заполнения верхнего массива (текстового по ключу)

{

if (checkWordInArray(idexText, key, *arrayText*)) {

arrayText[indexArrey] = key[idexText];

indexArrey++;

}

}

for (int indexAlph = 0, ind = 0; indexAlph < 32 && ind < 17;

indexAlph++) // алгоритм заполнения нижнего массива (оставшиеся пары)

{

if (checkWordInAlph(indexAlph, *alph*, *arrayText*)) {

arrayLast[ind] = alph[indexAlph];

ind++;

} else

continue;

}

bool exit = false;

for (int indexAlph = 0; indexAlph < 32; indexAlph++) {

exit = false;

for (int i = 0; i < 16; i++) {

if (alph[indexAlph] == arrayText[i]) {

alph[indexAlph] = arrayLast[i];

exit = true;

break;

}

}

if (exit)

continue;

for (int i = 0; i < 16; i++) {

if (alph[indexAlph] == arrayLast[i]) {

alph[indexAlph] = arrayText[i];

break;

}

}

}

}

bool MainWindow::**checkWordInArray2**(int idexText, QString key,

QString \*arrayText) {

for (int i = 0; i < 32; i++) {

if (key[idexText] == arrayText[i] || key[idexText] == " ")

return false;

}

return true;

}

bool MainWindow::**checkWordInArray2**(int idexText, QString \*key,

QString \*arrayText) {

for (int i = 0; i < 32; i++) {

if (key[idexText] == arrayText[i] || key[idexText] == " ")

return false;

}

return true;

}

void MainWindow::**createNewAlphPogvorka**(QString key, QString \*alph,

QString \*alphDefault) {

QString newAlph[32] = {0};

int indexArreyAlph = 0;

for (int idexText = 0; idexText < key.length(); idexText++) {

if (checkWordInArray2(idexText, key, *newAlph*)) {

newAlph[indexArreyAlph] = key[idexText];

indexArreyAlph++;

}

}

for (; indexArreyAlph < 32; indexArreyAlph++) {

for (int indexDefAlph = 0; indexDefAlph < 32;

indexDefAlph++) { //по массиву дефолтного алфавита

if (checkWordInArray2(indexDefAlph, *alphDefault*, *newAlph*)) {

newAlph[indexArreyAlph] = alphDefault[indexDefAlph];

break;

}

}

}

for (int i = 0; i < 32; i++) {

alph[i] = newAlph[i];

}

}

QString MainWindow::**translateText**(QString text, QString \*alph,

QString \*alphDefault) {

QString newText;

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

for (int indexAlph = 0; indexAlph < 32; indexAlph++) {

if (text[i] == alphDefault[indexAlph])

newText += alph[indexAlph];

}

}

for (int i = 0; i < 32; i++) {

alph[i] = alphDefault[i];

}

return newText;

}

void MainWindow::**createNewAlphConstant**(QString \*alph) {

QString alphDefault[32] = {"а", "б", "в", "г", "д", "е", "ж", "з",

"и", "й", "к", "л", "м", "н", "о", "п",

"р", "с", "т", "у", "ф", "х", "ц", "ч",

"ш", "щ", "ъ", "ы", "ь", "э", "ю", "я"};

for (int i = 0; i < 28; i++) {

alph[i] = alphDefault[i + 4];

}

for (int i = 28, k = 0; i < 32; i++, k++) {

alph[i] = alphDefault[k];

}

}

void MainWindow::**MatrixMultiplication**(int (&A\_matr)[5][5], int \*x0,

int \*x\_func) {

int sum;

for (int i = 0; i < 5; i++) {

sum = 0;

for (int j = 0; j < 5; j++) {

sum += A\_matr[i][j] \* x0[j];

}

x\_func[i] = sum;

}

}

void MainWindow::**GenerateLFSR**(QVector<int> &arrayLFSR, int (&A\_matr)[5][5],

int N) {

int x0[5] = {1, 0, 0, 0, 0};

int x\_func[5];

int sum\_binare;

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

MatrixMultiplication(*A\_matr*, *x0*, *x\_func*);

for (int count = 0; count < 5; count++) {

x0[count] = x\_func[count];

x0[count] %= 2;

}

sum\_binare = 0;

for (int j = 4, k = 0; k < 5; j--, k++) {

sum\_binare += x0[j] \* pow(2, k);

}

arrayLFSR.push\_back(sum\_binare);

}

}

void MainWindow::**lab2**(QString key, QString text) {

QString newalph[32] = {"а", "б", "в", "г", "д", "е", "ж", "з", "и", "й", "к",

"л", "м", "н", "о", "п", "р", "с", "т", "у", "ф", "х",

"ц", "ч", "ш", "щ", "ъ", "ы", "ь", "э", "ю", "я"};

QString alphDefault[32] = {"а", "б", "в", "г", "д", "е", "ж", "з",

"и", "й", "к", "л", "м", "н", "о", "п",

"р", "с", "т", "у", "ф", "х", "ц", "ч",

"ш", "щ", "ъ", "ы", "ь", "э", "ю", "я"};

QString alphSdvig[32] = {"а", "б", "в", "г", "д", "е", "ж", "з",

"и", "й", "к", "л", "м", "н", "о", "п",

"р", "с", "т", "у", "ф", "х", "ц", "ч",

"ш", "щ", "ъ", "ы", "ь", "э", "ю", "я"};

QString alphPogovorka[32] = {"а", "б", "в", "г", "д", "е", "ж", "з",

"и", "й", "к", "л", "м", "н", "о", "п",

"р", "с", "т", "у", "ф", "х", "ц", "ч",

"ш", "щ", "ъ", "ы", "ь", "э", "ю", "я"};

key = key.toLower();

text = text.toLower();

int N = text.length();

double h, hh = 0;

double ni;

int m = 32;

double a, b;

///////////////////////////////////////////// 1

QString newText1;

createNewAlph(key, *newalph*);

newText1 = translateText(text, *newalph*, *alphDefault*);

QVector<int> bit\_array\_one;

QVector<int> decimal\_array\_one;

Word(newText1, *bit\_array\_one*);

convertBinareDecimal(*bit\_array\_one*, *decimal\_array\_one*);

N = decimal\_array\_one.length();

hh = 0;

a = MinValue(*decimal\_array\_one*, N);

b = MaxValue(*decimal\_array\_one*, N);

h = 1;

QVector<double> x1(m), y1(m);

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

ni = 0;

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

if (decimal\_array\_one[j] >= double(a) + hh and

decimal\_array\_one[j] < double(a) + hh + h)

ni++;

else

continue;

}

x1[i] = double(a) + hh + (h / 2);

y1[i] = ni;

hh += h;

}

double first\_maxY = 1;

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

if (y1[i] > first\_maxY)

first\_maxY = y1[i];

}

ui->plot1->xAxis->setRange(0, 32);

ui->plot1->yAxis->setRange(0,

first\_maxY); //сюда надо вставить максимальное h

QCPBars \*bars1 = new QCPBars(*ui->plot1->xAxis*, *ui->plot1->yAxis*);

bars1->setData(x1, y1, true);

bars1->setWidth(h);

ui->plot1->replot();

///////////////////////////////////////////// 2 -константа

QString newText2;

createNewAlphConstant(*alphSdvig*);

newText2 = translateText(text, *alphSdvig*, *alphDefault*);

QVector<int> bit\_array\_two;

QVector<int> decimal\_array\_two;

Word(newText2, *bit\_array\_two*);

convertBinareDecimal(*bit\_array\_two*, *decimal\_array\_two*);

N = decimal\_array\_two.length();

hh = 0;

a = MinValue(*decimal\_array\_two*, N);

b = MaxValue(*decimal\_array\_two*, N);

h = 1;

QVector<double> x2(m), y2(m);

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

ni = 0;

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

if (decimal\_array\_two[j] >= double(a) + hh and

decimal\_array\_two[j] < double(a) + hh + h)

ni++;

else

continue;

}

x2[i] = double(a) + hh + (h / 2);

y2[i] = ni;

hh += h;

}

first\_maxY = 1;

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

if (y2[i] > first\_maxY)

first\_maxY = y2[i];

}

ui->plot2->xAxis->setRange(0, 32);

ui->plot2->yAxis->setRange(0,

first\_maxY); //сюда надо вставить максимальное h

QCPBars \*bars2 = new QCPBars(*ui->plot2->xAxis*, *ui->plot2->yAxis*);

bars2->setData(x2, y2, true);

bars2->setWidth(h);

ui->plot2->replot();

///////////////////////////////////////////// 3

QString newText3;

createNewAlphPogvorka(key, *alphPogovorka*, *alphDefault*);

newText3 = translateText(text, *alphPogovorka*, *alphDefault*);

QVector<int> bit\_array\_tree;

QVector<int> decimal\_array\_tree;

Word(newText3, *bit\_array\_tree*);

convertBinareDecimal(*bit\_array\_tree*, *decimal\_array\_tree*);

N = decimal\_array\_tree.length();

hh = 0;

a = MinValue(*decimal\_array\_tree*, N);

b = MaxValue(*decimal\_array\_tree*, N);

h = 1;

QVector<double> x3(m), y3(m);

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

ni = 0;

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

if (decimal\_array\_tree[j] >= double(a) + hh and

decimal\_array\_tree[j] < double(a) + hh + h)

ni++;

else

continue;

}

x3[i] = double(a) + hh + (h / 2);

y3[i] = ni;

hh += h;

}

first\_maxY = 1;

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

if (y3[i] > first\_maxY)

first\_maxY = y3[i];

}

ui->plot3->xAxis->setRange(0, 32);

ui->plot3->yAxis->setRange(0,

first\_maxY); //сюда надо вставитьмаксимальное h

QCPBars \*bars3 = new QCPBars(*ui->plot3->xAxis*, *ui->plot3->yAxis*);

bars3->setData(x3, y3, true);

bars3->setWidth(h);

ui->plot3->replot();

///////////////////////////////////////////// 4

QVector<int> bin\_array\_text;

Word(text, *bin\_array\_text*);

int X4 = 1;

int X3 = 1;

int X2 = 0;

int X1 = 1;

int X0 = 1;

int A\_matr[5][5] = {

{X4, X3, X2, X1, X0}, {1, 0, 0, 0, 0}, {0, 1, 0, 0, 0},

{0, 0, 1, 0, 0}, {0, 0, 0, 1, 0},

};

QVector<int> arrayLFSR;

QVector<int> binare\_arrayLFSR;

QVector<int> binare\_array\_sum;

QVector<int> decimal\_array\_four;

GenerateLFSR(*arrayLFSR*, *A\_matr*, text.length());

sumArray(*bin\_array\_text*, *binare\_arrayLFSR*, *binare\_array\_sum*);

convertBinareDecimal(*binare\_array\_sum*, *decimal\_array\_four*);

N = decimal\_array\_four.length();

hh = 0;

a = 0;

b = 32;

h = 1;

QVector<double> x4(m), y4(m);

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

ni = 0;

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

if (decimal\_array\_four[j] >= a + hh and

decimal\_array\_four[j] < a + hh + h)

ni++;

else

continue;

}

x4[i] = double(a) + hh + (h / 2);

y4[i] = ni;

hh += h;

}

first\_maxY = 1;

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

if (y4[i] > first\_maxY)

first\_maxY = y4[i];

}

ui->plot4->xAxis->setRange(0, 32);

ui->plot4->yAxis->setRange(0, first\_maxY);

QCPBars \*bars4 = new QCPBars(*ui->plot4->xAxis*, *ui->plot4->yAxis*);

bars4->setData(x4, y4, true);

bars4->setWidth(h);

ui->plot4->replot();

}

void MainWindow::**bin8bit**(int n,

QVector<int> &bit\_array) //функция перевода числа в

//двоичную систему счисления

{

int arr[8];

for (int i = 7; i >= 0; i--) {

arr[i] = n % 2;

n = n / 2;

}

for (int i = 0; i < 8; i++) {

bit\_array.push\_back(arr[i]);

}

}

void MainWindow::**WordEightBit**(

QString str, QVector<int> &bit\_array,

bool iskey) // функция перевода строки в двоичный код

{

int \*arr\_num = new int[str.length()];

QString smal[32] = {"а", "б", "в", "г", "д", "е", "ж", "з", "и", "й", "к",

"л", "м", "н", "о", "п", "р", "с", "т", "у", "ф", "х",

"ц", "ч", "ш", "щ", "ъ", "ы", "ь", "э", "ю", "я"};

QString big[32] = {"А", "Б", "В", "Г", "Д", "Е", "Ж", "З", "И", "Й", "К",

"Л", "М", "Н", "О", "П", "Р", "С", "Т", "У", "Ф", "Х",

"Ц", "Ч", "Ш", "Щ", "Ъ", "Ы", "Ь", "Э", "Ю", "Я"};

if (!iskey) {

if (str.length() % 2 != 0) {

for (int i = 0; i < 8; i++)

bit\_array.push\_back(0);

}

}

for (int i = 0; i < str.length();

i++) //перебор символов и запись их в вектор двоичных чисел

{

for (int j = 0; j < 32; j++) {

if (str[i] == smal[j]) {

arr\_num[i] = j;

bin8bit(arr\_num[i], *bit\_array*);

break;

} else if (str[i] == big[j]) {

arr\_num[i] = j;

bin8bit(arr\_num[i], *bit\_array*);

break;

}

}

}

delete[] arr\_num;

}

void MainWindow::**sumArray2**(int \*arr1, int \*arr2, int \*result) {

for (int i = 0; i < 8; i++) {

if ((arr1[i] == 1 && arr2[i] == 1) || (arr1[i] == 0 && arr2[i] == 0))

result[i] = 0;

if ((arr1[i] == 1 && arr2[i] == 0) || (arr1[i] == 0 && arr2[i] == 1))

result[i] = 1;

}

}

void MainWindow::**sdvig**(int \*result) {

int buffer[8];

for (int i = 0; i < 4; i++) {

buffer[i] = result[i];

}

for (int i = 4; i < 8; i++) {

buffer[i - 1] = result[i];

}

buffer[7] = result[3];

for (int i = 0; i < 8; i++) {

result[i] = buffer[i];

}

}

void MainWindow::**round**(QVector<int> &bit\_array\_text, int NumRound,

QVector<int> &bit\_array\_key, QString str) {

bool propuskRight = true;

int left[8];

int right[8];

int key[8];

int result[8];

QVector<int> buffer;

NumRound--;

if (str.length() % 2 != 0)

propuskRight = false;

for (int rezakEpta = 0; rezakEpta < 8; rezakEpta++) {

key[rezakEpta] = bit\_array\_key[rezakEpta + NumRound \* 8];

}

for (int rezak = 0; rezak < bit\_array\_text.length(); rezak += 16) {

for (int i = 0; i < 8; i++)

left[i] = bit\_array\_text[i + rezak];

for (int i = 0; i < 8; i++)

right[i] = bit\_array\_text[i + rezak + 8];

sumArray2(*right*, *key*, *result*);

sdvig(*result*);

if (NumRound != 7) {

for (int i = 0; i < 8; i++) {

right[i] = left[i];

left[i] = result[i];

}

} else {

for (int i = 0; i < 8; i++)

left[i] = result[i];

}

for (int i = 0; i < 8; i++)

buffer.push\_back(left[i]);

if (propuskRight) {

for (int i = 0; i < 8; i++)

buffer.push\_back(right[i]);

propuskRight = true;

} else

propuskRight = true;

}

bit\_array\_text.clear();

bit\_array\_text = buffer;

}

void MainWindow::**convertBinare8Decimal**(QVector<int> &in\_bit,

QVector<int> &out\_decimal) {

out\_decimal.clear();

int sum;

for (int i = 0; i < in\_bit.length(); i += 5) {

sum = 0;

for (int j = 4, k = 0; j >= 0; j--, k++) {

if (in\_bit[i + k] == 1) {

sum += pow(2, j);

}

}

out\_decimal.push\_back(sum);

}

}

void MainWindow::**rubka**(QVector<int> &bit\_array, QVector<int> &decimal\_array) {

QVector<int> buffer;

for (int i = 3; i < bit\_array.length(); i += 8) {

for (int j = 0; j < 5; j++) {

buffer.push\_back(bit\_array[i + j]);

}

convertBinare8Decimal(*buffer*, *decimal\_array*);

}

}

QString MainWindow::**convertDecimalWords**(QVector<int> DecimalArr) {

QString stringOut;

QString alph[32] = {"а", "б", "в", "г", "д", "е", "ж", "з", "и", "й", "к",

"л", "м", "н", "о", "п", "р", "с", "т", "у", "ф", "х",

"ц", "ч", "ш", "щ", "ъ", "ы", "ь", "э", "ю", "я"};

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

stringOut += alph[DecimalArr[i]];

}

return stringOut;

}

void MainWindow::**lab3**(QString text, QString key) {

QVector<int> bit\_array\_text;

QVector<int> bit\_array\_keys;

QVector<int> bit\_one;

QVector<int> decimal\_one;

QVector<int> bit\_two;

QVector<int> decimal\_two;

QVector<int> bit\_tree;

QVector<int> decimal\_tree;

QVector<int> bit\_four;

QVector<int> decimal\_four;

QVector<int> bit\_five;

QVector<int> decimal\_five;

QVector<int> bit\_six;

QVector<int> decimal\_six;

QVector<int> bit\_seven;

QVector<int> decimal\_seven;

QVector<int> bit\_eight;

QVector<int> decimal\_eight;

WordEightBit(text, *bit\_array\_text*, false);

WordEightBit(key, *bit\_array\_keys*, true);

bit\_one = bit\_array\_text;

round(*bit\_one*, 1, *bit\_array\_keys*, text);

rubka(*bit\_one*, *decimal\_one*);

ui->Round1->setText(convertDecimalWords(decimal\_one));

bit\_two = bit\_one;

round(*bit\_two*, 2, *bit\_array\_keys*, text);

rubka(*bit\_two*, *decimal\_two*);

ui->Round2->setText(convertDecimalWords(decimal\_two));

bit\_tree = bit\_two;

round(*bit\_tree*, 3, *bit\_array\_keys*, text);

rubka(*bit\_tree*, *decimal\_tree*);

ui->Round3->setText(convertDecimalWords(decimal\_tree));

bit\_four = bit\_tree;

round(*bit\_four*, 4, *bit\_array\_keys*, text);

rubka(*bit\_four*, *decimal\_four*);

ui->Round4->setText(convertDecimalWords(decimal\_four));

bit\_five = bit\_four;

round(*bit\_five*, 5, *bit\_array\_keys*, text);

rubka(*bit\_five*, *decimal\_five*);

ui->Round5->setText(convertDecimalWords(decimal\_five));

bit\_six = bit\_five;

round(*bit\_six*, 6, *bit\_array\_keys*, text);

rubka(*bit\_six*, *decimal\_six*);

ui->Round6->setText(convertDecimalWords(decimal\_six));

bit\_seven = bit\_six;

round(*bit\_seven*, 7, *bit\_array\_keys*, text);

rubka(*bit\_seven*, *decimal\_seven*);

ui->Round7->setText(convertDecimalWords(decimal\_seven));

bit\_eight = bit\_seven;

round(*bit\_eight*, 8, *bit\_array\_keys*, text);

rubka(*bit\_eight*, *decimal\_eight*);

ui->Round8->setText(convertDecimalWords(decimal\_eight));

int N = text.length();

double h, hh = 0;

double ni;

int m = 32;

double a, b;

////////////////////////////////////////////1

N = decimal\_one.length();

a = MinValue(*decimal\_one*, N);

b = MaxValue(*decimal\_one*, N);

h = 1;

QVector<double> x1(m), y1(m);

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

ni = 0;

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

if (decimal\_one[j] >= double(a) + hh and

decimal\_one[j] < double(a) + hh + h)

ni++;

else

continue;

}

x1[i] = double(a) + hh + (h / 2);

y1[i] = ni;

hh += h;

}

double first\_maxY = 1;

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

if (y1[i] > first\_maxY)

first\_maxY = y1[i];

}

ui->one->xAxis->setRange(0, 32);

ui->one->yAxis->setRange(0, first\_maxY);

QCPBars \*bars1 = new QCPBars(*ui->one->xAxis*, *ui->one->yAxis*);

bars1->setData(x1, y1, true);

bars1->setWidth(h);

ui->one->replot();

////////////////////////////////////////////1

N = decimal\_two.length();

a = MinValue(*decimal\_two*, N);

b = MaxValue(*decimal\_two*, N);

h = 1;

hh = 0;

QVector<double> x2(m), y2(m);

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

ni = 0;

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

if (decimal\_two[j] >= double(a) + hh and

decimal\_two[j] < double(a) + hh + h)

ni++;

else

continue;

}

x2[i] = double(a) + hh + (h / 2);

y2[i] = ni;

hh += h;

}

first\_maxY = 1;

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

if (y2[i] > first\_maxY)

first\_maxY = y2[i];

}

ui->two->xAxis->setRange(0, 32);

ui->two->yAxis->setRange(0, first\_maxY);

QCPBars \*barstwo = new QCPBars(*ui->two->xAxis*, *ui->two->yAxis*);

barstwo->setData(x2, y2, true);

barstwo->setWidth(h);

ui->two->replot();

////////////////////////////////////////////3

N = decimal\_tree.length();

a = MinValue(*decimal\_tree*, N);

b = MaxValue(*decimal\_tree*, N);

h = 1;

hh = 0;

QVector<double> x3(m), y3(m);

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

ni = 0;

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

if (decimal\_tree[j] >= double(a) + hh and

decimal\_tree[j] < double(a) + hh + h)

ni++;

else

continue;

}

x3[i] = double(a) + hh + (h / 2);

y3[i] = ni;

hh += h;

}

first\_maxY = 1;

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

if (y3[i] > first\_maxY)

first\_maxY = y3[i];

}

ui->tree->xAxis->setRange(0, 32);

ui->tree->yAxis->setRange(0, first\_maxY);

QCPBars \*barstree = new QCPBars(*ui->tree->xAxis*, *ui->tree->yAxis*);

barstree->setData(x3, y3, true);

barstree->setWidth(h);

ui->tree->replot();

////////////////////////////////////////////4

N = decimal\_four.length();

a = MinValue(*decimal\_four*, N);

b = MaxValue(*decimal\_four*, N);

h = 1;

hh = 0;

QVector<double> x4(m), y4(m);

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

ni = 0;

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

if (decimal\_four[j] >= double(a) + hh and

decimal\_four[j] < double(a) + hh + h)

ni++;

else

continue;

}

x4[i] = double(a) + hh + (h / 2);

y4[i] = ni;

hh += h;

}

first\_maxY = 1;

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

if (y4[i] > first\_maxY)

first\_maxY = y4[i];

}

ui->four->xAxis->setRange(0, 32);

ui->four->yAxis->setRange(0, first\_maxY);

QCPBars \*barsfour = new QCPBars(*ui->four->xAxis*, *ui->four->yAxis*);

barsfour->setData(x4, y4, true);

barsfour->setWidth(h);

ui->four->replot();

////////////////////////////////////////////5

N = decimal\_five.length();

a = MinValue(*decimal\_five*, N);

b = MaxValue(*decimal\_five*, N);

h = 1;

hh = 0;

QVector<double> x5(m), y5(m);

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

ni = 0;

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

if (decimal\_five[j] >= double(a) + hh and

decimal\_five[j] < double(a) + hh + h)

ni++;

else

continue;

}

x5[i] = double(a) + hh + (h / 2);

y5[i] = ni;

hh += h;

}

first\_maxY = 1;

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

if (y5[i] > first\_maxY)

first\_maxY = y5[i];

}

ui->five->xAxis->setRange(0, 32);

ui->five->yAxis->setRange(0, first\_maxY);

QCPBars \*barsfive = new QCPBars(*ui->five->xAxis*, *ui->five->yAxis*);

barsfive->setData(x5, y5, true);

barsfive->setWidth(h);

ui->five->replot();

////////////////////////////////////////////6

N = decimal\_six.length();

a = MinValue(*decimal\_six*, N);

b = MaxValue(*decimal\_six*, N);

h = 1;

hh = 0;

QVector<double> x6(m), y6(m);

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

ni = 0;

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

if (decimal\_six[j] >= double(a) + hh and

decimal\_six[j] < double(a) + hh + h)

ni++;

else

continue;

}

x6[i] = double(a) + hh + (h / 2);

y6[i] = ni;

hh += h;

}

first\_maxY = 1;

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

if (y6[i] > first\_maxY)

first\_maxY = y6[i];

}

ui->six->xAxis->setRange(0, 32);

ui->six->yAxis->setRange(0, first\_maxY);

QCPBars \*barssix = new QCPBars(*ui->six->xAxis*, *ui->six->yAxis*);

barssix->setData(x6, y6, true);

barssix->setWidth(h);

ui->six->replot();

////////////////////////////////////////////7

N = decimal\_seven.length();

a = MinValue(*decimal\_seven*, N);

b = MaxValue(*decimal\_seven*, N);

h = 1;

hh = 0;

QVector<double> x7(m), y7(m);

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

ni = 0;

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

if (decimal\_seven[j] >= double(a) + hh and

decimal\_seven[j] < double(a) + hh + h)

ni++;

else

continue;

}

x7[i] = double(a) + hh + (h / 2);

y7[i] = ni;

hh += h;

}

first\_maxY = 1;

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

if (y7[i] > first\_maxY)

first\_maxY = y7[i];

}

ui->seven->xAxis->setRange(0, 32);

ui->seven->yAxis->setRange(0, first\_maxY);

QCPBars \*barsseven = new QCPBars(*ui->seven->xAxis*, *ui->seven->yAxis*);

barsseven->setData(x7, y7, true);

barsseven->setWidth(h);

ui->seven->replot();

////////////////////////////////////////////8

N = decimal\_eight.length();

a = MinValue(*decimal\_eight*, N);

b = MaxValue(*decimal\_eight*, N);

h = 1;

hh = 0;

QVector<double> x8(m), y8(m);

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

ni = 0;

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

if (decimal\_eight[j] >= double(a) + hh and

decimal\_eight[j] < double(a) + hh + h)

ni++;

else

continue;

}

x8[i] = double(a) + hh + (h / 2);

y8[i] = ni;

hh += h;

}

first\_maxY = 1;

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

if (y8[i] > first\_maxY)

first\_maxY = y8[i];

}

ui->eight->xAxis->setRange(0, 32);

ui->eight->yAxis->setRange(0, first\_maxY);

QCPBars \*barseight = new QCPBars(*ui->eight->xAxis*, *ui->eight->yAxis*);

barseight->setData(x8, y8, true);

barseight->setWidth(h);

ui->eight->replot();

}

void MainWindow::**on\_pushButton\_clicked**() {

ui->OFB->clearPlottables(); //очистка гистограмм

ui->CPC->clearPlottables();

ui->CFB->clearPlottables();

ui->ECB->clearPlottables();

ui->plot1->clearPlottables();

ui->plot2->clearPlottables();

ui->plot3->clearPlottables();

ui->plot4->clearPlottables();

ui->one->clearPlottables();

ui->two->clearPlottables();

ui->tree->clearPlottables();

ui->four->clearPlottables();

ui->five->clearPlottables();

ui->six->clearPlottables();

ui->seven->clearPlottables();

ui->eight->clearPlottables();

ui->textPlot->clearPlottables();

ui->Round1->clear();

ui->Round2->clear();

ui->Round3->clear();

ui->Round4->clear();

ui->Round5->clear();

ui->Round6->clear();

ui->Round7->clear();

ui->Round8->clear();

QString text = ui->textEdit->toPlainText(); //ввод с окна

QString k0 = ui->lineEdit\_4->text();

QString k1 = ui->lineEdit\_3->text();

QString p0 = ui->lineEdit->text();

QString c0 = ui->lineEdit\_2->text();

QString key = ui->key->text(); //ввод с окна

if (key == "") {

// ui->key->setText("Сурьма косых глаз не исправит");

// key="Сурьма косых глаз не исправит";

ui->key->setText("Железный шпиц дома лежит");

key = "Железный шпиц дома лежит";

}

if (ui->checkBox\_3->isChecked())

lab2(key, text);

if (ui->checkBox\_2->isChecked())

lab3(text, key);

int K0[5];

int K1[5];

int P0[5];

int C0[5];

////////////////////////////////////////Перевод входных значений в int

if (k0 == "" && k1 == "" && p0 == "" && c0 == "") {

k0 = "10001";

k1 = "00001";

p0 = "00000";

c0 = "11100";

ui->lineEdit\_4->setText(k0);

ui->lineEdit\_3->setText(k1);

ui->lineEdit->setText(p0);

ui->lineEdit\_2->setText(c0);

}

if (k0.length() == 5) {

for (int i = 0; i < 5; i++) {

if (k0[i] == '0')

K0[i] = 0;

else if (k0[i] == '1')

K0[i] = 1;

else

ui->lineEdit\_4->setText("Ошибка ввода");

}

} else

ui->lineEdit\_4->setText("Ошибка ввода");

if (k1.length() == 5) {

for (int i = 0; i < 5; i++) {

if (k1[i] == '0')

K1[i] = 0;

else if (k1[i] == '1')

K1[i] = 1;

else

ui->lineEdit\_3->setText("Ошибка ввода");

}

} else

ui->lineEdit\_3->setText("Ошибка ввода");

if (p0.length() == 5) {

for (int i = 0; i < 5; i++) {

if (p0[i] == '0')

P0[i] = 0;

else if (p0[i] == '1')

P0[i] = 1;

else

ui->lineEdit->setText("Ошибка ввода");

}

} else

ui->lineEdit->setText("Ошибка ввода");

if (c0.length() == 5) {

for (int i = 0; i < 5; i++) {

if (c0[i] == '0')

C0[i] = 0;

else if (c0[i] == '1')

C0[i] = 1;

else

ui->lineEdit\_2->setText("Ошибка ввода");

}

} else

ui->lineEdit\_2->setText("Ошибка ввода");

////////////////////////////////////////////////////////////Чтение и перевод

///в двоичный код текста

QVector<int> bit\_array;

QVector<int> text\_decimal;

Word(text, *bit\_array*);

convertBinareDecimal(*bit\_array*, *text\_decimal*);

QString out = "";

int N = text\_decimal.length();

double h, hh = 0;

double ni;

int m = 32;

double a, b;

//////////////////////////////////////////////////////////// График Текста

N = text\_decimal.length();

hh = 0;

a = MinValue(*text\_decimal*, N);

b = MaxValue(*text\_decimal*, N);

h = 1;

QVector<double> x0(m), y0(m);

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

ni = 0;

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

if (text\_decimal[j] >= double(a) + hh and

text\_decimal[j] < double(a) + hh + h)

ni++;

else

continue;

}

x0[i] = double(a) + hh + (h / 2);

y0[i] = ni;

hh += h;

}

double first\_maxY = 1;

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

if (y0[i] > first\_maxY)

first\_maxY = y0[i];

}

ui->textPlot->xAxis->setRange(0, 32);

ui->textPlot->yAxis->setRange(0, first\_maxY);

QCPBars \*bars0 = new QCPBars(*ui->textPlot->xAxis*, *ui->textPlot->yAxis*);

bars0->setData(x0, y0, true);

bars0->setWidth(h);

ui->textPlot->replot();

if (ui->checkBox->isChecked()) {

////////////////////////////////////////////////////////////График ECB

QVector<int> ECB\_out\_bit;

QVector<int> ECB\_out;

ECB(*bit\_array*, *ECB\_out\_bit*, *K0*, *K1*);

convertBinareDecimal(*ECB\_out\_bit*, *ECB\_out*);

N = ECB\_out.length();

hh = 0;

a = MinValue(*ECB\_out*, N);

b = MaxValue(*ECB\_out*, N);

h = 1;

QVector<double> x1(m), y1(m);

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

ni = 0;

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

if (ECB\_out[j] >= double(a) + hh and ECB\_out[j] < double(a) + hh + h)

ni++;

else

continue;

}

x1[i] = double(a) + hh + (h / 2);

y1[i] = ni;

hh += h;

}

ui->ECB->xAxis->setRange(0, 32);

ui->ECB->yAxis->setRange(0, first\_maxY);

QCPBars \*bars1 = new QCPBars(*ui->ECB->xAxis*, *ui->ECB->yAxis*);

bars1->setData(x1, y1, true);

bars1->setWidth(h);

ui->ECB->replot();

///////////////////////////////////////////// График CPC

QVector<int> CPC\_out\_bit;

QVector<int> CPC\_out;

CPC(*bit\_array*, *CPC\_out\_bit*, *C0*);

convertBinareDecimal(*CPC\_out\_bit*, *CPC\_out*);

N = CPC\_out.length();

hh = 0;

a = MinValue(*CPC\_out*, N);

b = MaxValue(*CPC\_out*, N);

h = 1;

QVector<double> x2(m), y2(m);

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

ni = 0;

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

if (CPC\_out[j] >= double(a) + hh and CPC\_out[j] < double(a) + hh + h)

ni++;

else

continue;

}

x2[i] = double(a) + hh + (h / 2);

y2[i] = ni;

hh += h;

}

ui->CPC->xAxis->setRange(0, 32);

ui->CPC->yAxis->setRange(0,

first\_maxY); //сюда надо вставить максимальное h

QCPBars \*bars2 = new QCPBars(*ui->CPC->xAxis*, *ui->CPC->yAxis*);

bars2->setData(x2, y2, true);

bars2->setWidth(h);

ui->CPC->replot();

////////////////////////////////////////////////////////////График OFB

QVector<int> OFB\_out\_bit;

QVector<int> OFB\_out;

OFB(*bit\_array*, *OFB\_out\_bit*, *C0*, *P0*);

convertBinareDecimal(*OFB\_out\_bit*, *OFB\_out*);

N = OFB\_out.length();

hh = 0;

a = MinValue(*OFB\_out*, N);

b = MaxValue(*OFB\_out*, N);

h = 1;

QVector<double> x3(m), y3(m);

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

ni = 0;

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

if (OFB\_out[j] >= double(a) + hh and OFB\_out[j] < double(a) + hh + h)

ni++;

else

continue;

}

x3[i] = double(a) + hh + (h / 2);

y3[i] = ni;

hh += h;

}

ui->OFB->xAxis->setRange(0, 32);

ui->OFB->yAxis->setRange(0, first\_maxY);

QCPBars \*bars3 = new QCPBars(*ui->OFB->xAxis*, *ui->OFB->yAxis*);

bars3->setData(x3, y3, true);

bars3->setWidth(h);

ui->OFB->replot();

////////////////////////////////////////////////////////////График CFB

QVector<int> CFB\_out\_bit;

QVector<int> CFB\_out;

CFB(*bit\_array*, *CFB\_out\_bit*, *C0*);

convertBinareDecimal(*CFB\_out\_bit*, *CFB\_out*);

N = CFB\_out.length();

hh = 0;

a = MinValue(*CFB\_out*, N);

b = MaxValue(*CFB\_out*, N);

h = 1;

QVector<double> x4(m), y4(m);

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

ni = 0;

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

if (CFB\_out[j] >= double(a) + hh and CFB\_out[j] < double(a) + hh + h)

ni++;

else

continue;

}

x4[i] = double(a) + hh + (h / 2);

y4[i] = ni;

hh += h;

}

ui->CFB->xAxis->setRange(0, 32);

ui->CFB->yAxis->setRange(0, first\_maxY);

QCPBars \*bars4 = new QCPBars(*ui->CFB->xAxis*, *ui->CFB->yAxis*);

bars4->setData(x4, y4, true);

bars4->setWidth(h);

ui->CFB->replot();

}

}