Министерство образования и науки Российской Федерации

Федеральное Государственное Бюджетное Образовательное Учреждение

Высшего Профессионального Образования

Тамбовский Государственный Технический Университет

Кафедра

Лабораторная работа №1

дисциплина «Защита информации»

Вариант 17

Выполнил студент гр. –41

Д. Ю.

Проверила: И.Л.

Тамбов

**Задание:**

*Реализовать четыре алгоритма шифрования:*

1. *Атбаш*
2. *Магический квадрат*
3. *Гаммирование по модулю N*
4. *ADFGX*

**Решение:**

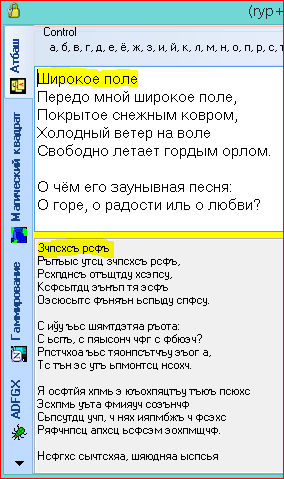
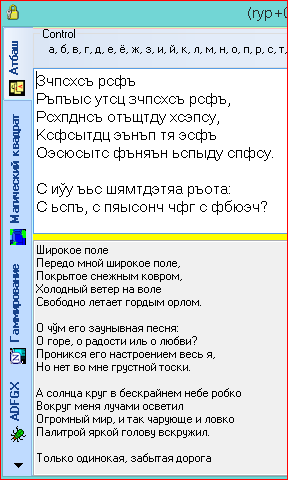
Атбаш

Атбаш можно считать шифром сдвига на всю длину алфавита или того числа символов, которые представлены к замене. Возьмем два алфавита, один из которых написан наоборот:

АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ  
ЯЮЭЬЫЪЩШЧЦХФУТСРПОНМЛКЙИЗЖЕДГВБА

Видно взаимное соответствие букв, которые заменяют друг друга.

Закодировать сообщение этим шифром очень простою  
Возьмем текст: **Широкое поле**

Получаем перевод: **Зчпсхсъ рсфъ**

Из рисунков, приведенных выше, можно сделать вывод, что алгоритм довольно примитивен и вскрывается на раз. Тем не менее, в далекой древности эта методика пользовалась спросом.

Код программы Атбаш без учета визуальных контролов:

**using** System**.**Text**.**RegularExpressions**;**

//http://kriptografea.narod.ru/atbash.html

**namespace** CryptoMona

**{**

/// <summary>

/// Шифрование Атбаш

/// </summary>

class clAtbash

**{**

**public** clAtbash**(**string txtIn**)** **{** **}**

/// <summary>

/// Запрос алфавит-строки

/// </summary>

**internal** string StrAlf**(**string flagLng**)**

**{**

**if** **(**flagLng**.**Equals**(**"ru"**))**

**return** \_ru**;**

**if** **(**flagLng**.**Equals**(**"en"**))**

**return** \_en**;**

**return** "-1"**;**

**}**

/// <summary>

/// принять введеную строку

/// </summary>

**internal** void Analiz**(**string textin**)**

**{**

OutString **=** ""**;**

//взять символ и перевернуть с учетом выбранного алфавита

//1]От начала алфавита до символа столько букв [len,

//2]сколько от конца алфавита до шифр-символа [x.

**foreach** **(**char c **in** textin**)**

**{**

//65..90 OR 97..122 [En]

**if((**c**>=**65 **&&** c**<=**90**)||(**c**>=**97 **&&** c**<=**122**))**

**{** //1]

int len **=** isToUp**(**c**,** **true)** **?** c **-** 65 **:** c **-** 97**;**

//2]

char x **=** **(**char**)(**isToUp**(**c**,** **true)** **?** 90 **-** len **:** 122 **-** len**);**

OutString **+=** x**;**

**}**

//1040..1103 OR 1025 OR 1105 [Ru]

**else** **if** **((**c **>=** 1040 **&&** c **<=** 1103**)** **||** **(**c **==** 1025 **|** c **==** 1105**))**

**{**

//1]65=ё ИЛИ -15=Ё

int len **=** c **-** 1040**;**

//2]

char x **=** isToUp**(**c**,** **false)** **?** char**.**ToUpper**((**char**)** **(**1103 **-** len**))** **:** char**.**ToLower**((**char**)** **(**1103 **-** len**));**

OutString **+=** x**;**

**}**

**else** OutString **+=** c**;**

**}**

**}**

/// <summary>

/// Выходная шифрованая строка

/// </summary>

**internal** string OutString **{** get**;** **private** set**;** **}**

/// <summary>

/// Вернет True при заглавных символах на входе

/// </summary>

/// <param name="c">Символ на входе</param>

/// <param name="flagLeng">true=En</param>

/// <returns>False при наборе [a-z]</returns>

**private** bool isToUp**(**char c**,** bool flagLeng**)**

**{**

var \_regexEn **=** **new** Regex**(**

"[A-Z-[aeiou]]"**,**

RegexOptions**.**CultureInvariant

**|** RegexOptions**.**IgnorePatternWhitespace

**|** RegexOptions**.**Compiled

**);**

var \_regexRu **=** **new** Regex**(**

"[ЁА-Я-[aeiou]]"**,**

RegexOptions**.**CultureInvariant

**|** RegexOptions**.**IgnorePatternWhitespace

**|** RegexOptions**.**Compiled

**);**

**return** flagLeng **?** \_regexEn**.**Match**(**c**.**ToString**()).**Success **:** \_regexRu**.**Match**(**c**.**ToString**()).**Success**;**

**}**

**private** string \_ru **=** "а, б, в, г, д, е, ё, 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"**;**//33

**private** string \_en **=** "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"**;**//26

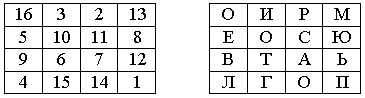
**}**

**}**

Магический квадрат

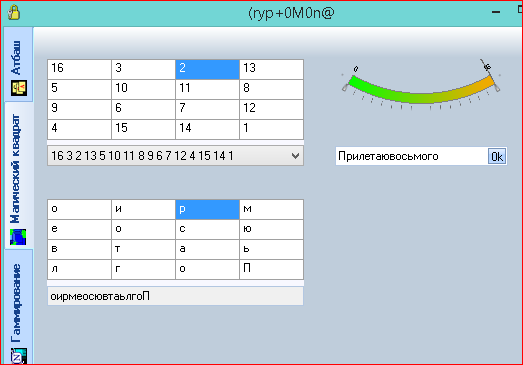
*Магическими квадратами* называют квадратные таблицы с вписанными в их клетки последовательными натуральными числами, начиная от 1, которые дают в сумме по каждому столбцу, каждой строке и каждой диагонали одно и то же число.

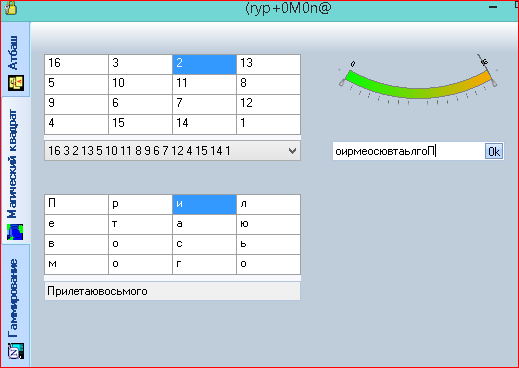
Шифруемый текст вписывали в магические квадраты в соответствии с нумерацией их клеток. Если затем выписать содержимое такой таблицы по строкам, то получится шифртекст, сформированный благодаря перестановке букв исходного сообщения. Считалось, что созданные с помощью магических квадратов шифртексты охраняет не только ключ, но и магическая сила.

Пример магического квадрата и его заполнения сообщением «ПРИЛЕТАЮ ВОСЬМОГО»:

Шифртекст, получаемый при считывании содержимого правой таблицы по строкам, имеет вполне загадочный вид:

**ОИРМ ЕОСЮ ВТАЬ ЛГОП**

Так как таблица имеет размер 4\*4, возникает вопрос – как быть если сообщение имеет большую длину? Как вариант реализации – использовать прогон текста по блокам максимально возможной длинны.



Алгоритм более интересен, чем предыдущий, так как квадратов 4\*4 с разной числовой последовательностью может быть 880, что влияет на время, необходимое для вскрытия. Увы, современные машины с такими задачами справляются на раз.

Код алгоритма Магический квадрат:

//http://neudoff.net/info/informatika/shifrovanie-po-metodu-magicheskix-kvadratov/

//http://crypto-r.narod.ru/glava2/glava2\_2.html

**namespace** CryptoMona

**{**

/// <summary>

/// Шифрование методом Магический квадрат

/// </summary>

class clMagicQuad

**{**

**public** clMagicQuad**(**HeaparGrid**.**HeaparGrid squad**,** **object** textNum**)**

**{**

//выпилить массив чисел-строк

string**[]** words **=** textNum**.**ToString**().**Split**(**' '**);** //Делим строку на ПОДСТРОКИ

//заполнить обьект цифровой последовательностью

quadOut **=** SetNumeric**(**squad**,** words**);**

**}**

/// <summary>

/// Обьект на форме, содержит цифровой ряд

/// </summary>

**internal** HeaparGrid**.**HeaparGrid quadOut **{** get**;** **private** set**;** **}**

/// <summary>

/// Залить числа в таблицу

/// </summary>

/// <param name="squad">Таблица которую заполняем</param>

/// <param name="num">Последовательность чисел</param>

/// <returns>Готовый обьект</returns>

**private** HeaparGrid**.**HeaparGrid SetNumeric**(**HeaparGrid**.**HeaparGrid squad**,** string**[]** num**)**

**{**

int d **=** 0**;**

**for** **(**int i **=** 0**;** i **<** squad**.**ColCount**;** i**++)**

**{**

**for** **(**int j **=** 0**;** j **<** squad**.**RowCount**;** j**++)**

**{**

squad**[**j**,** i**].**Data **=** num**[**d**];**

d**++;**

**}**

**}**

**return** squad**;**

**}**

/// <summary>

/// Заполнить таблицу входящим сообщением

/// </summary>

/// <param name="txtIn">Входящее сообщение</param>

/// <param name="gridOut">Таблица для заполнения</param>

**internal** void SetCrypto**(**string txtIn**,** HeaparGrid**.**HeaparGrid gridOut**)**

**{**

TextOut **=** ""**;**

//Ячейка==индекс символа,

//Индекс символа==quadOut[j,i]

**for** **(**int i **=** 0**;** i **<** gridOut**.**ColCount**;** i**++)**

**{**

**for** **(**int j **=** 0**;** j **<** gridOut**.**RowCount**;** j**++)**

**{**

int num **=** int**.**Parse**(**quadOut**[**j**,** i**].**Data**.**ToString**());**

**if** **(**num **>** txtIn**.**Length**)** gridOut**[**j**,** i**].**Data **=** "\_"**;**

**else** gridOut**[**j**,** i**].**Data **=** txtIn**[**num **-** 1**];**

//Записать результат в выходСтроку

TextOut **+=** gridOut**[**j**,** i**].**Data **+** ""**;**

**}**

**}**

**}**

/// <summary>

/// Для передачи шифрстроки на форму

/// </summary>

**internal** string TextOut **{** get**;** **private** set**;** **}**

**}**

**}**

Гаммирование mod N

В аддитивных шифрах символы исходного сообщения заменяются числами, которые складываются по модулю с числами гаммы. Ключом шифра является гамма, символы которой последовательно повторяются.

Перед шифрованием символы сообщения и гаммы заменяются их номерами в алфавите и само кодирование выполняется по формуле:

, где N равен количеству символов применяемого алфавита, C-T-G соответственно, шифрограммы, шифруемого текста и гаммы. Если С будет равно нулю, то его следует приравнять N. Создание шифрограммы завершается заменой полученных чисел C на соответствующие буквы алфавита.

В примере ниже исходное сообщение - «КАФЕДРА СИСТЕМ ИНФОРМАТИКИ», используемая гамма - «СИМВОЛ»:

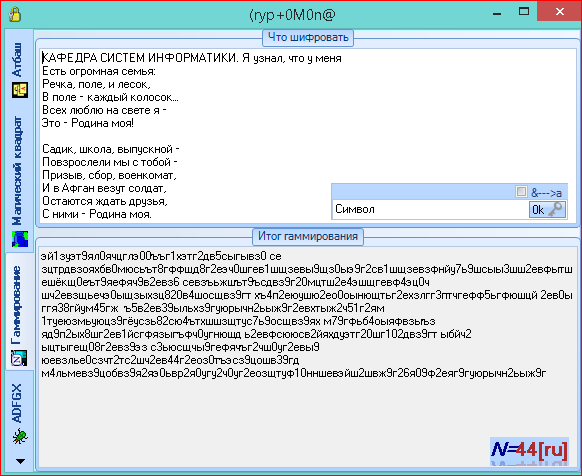
В данной теме используется алфавит, состоящий из 44 символов:

Дешифрирование выполняется по формуле:

, где T это символы исходного сообщения, C символы зашифрованного сообщения, G символы гаммы. Если Т=0, то его приравнивают к N.

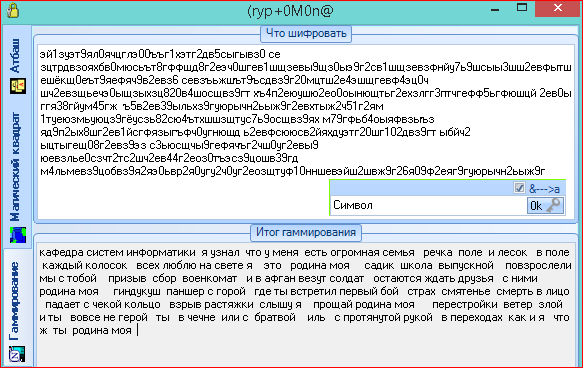
В примере ниже зашифрованное выше сообщение вновь приводится к исходному виду:

Шифруем:



Алгоритм достаточно понятен, на практике возможна реализация числа N как произвольно заданного, это затруднит разгадку ключа. Для обмена сообщениями, которые не представляют особой важности для третьих лиц это самое то.

Расшифровка:



Коды запуска алгоритма:

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Linq**;**

**using** System**.**Text**;**

//http://www.altaev-aa.narod.ru/security/XOR.html

//http://edu.dvgups.ru/METDOC/ENF/PRMATEM/INFORMAT/METOD/KRIPTOGR\_MET/Kom\_4.htm

//http://edu.dvgups.ru/METDOC/ENF/PRMATEM/INFORMAT/METOD/KRIPTOGR\_MET/Kom\_5.htm

**namespace** CryptoMona

**{**

/// <summary>

/// Шифрование метод гаммирования по модулю N

/// N - количество символов алфавита

/// </summary>

class clGammaN

**{**

**public** clGammaN**(**int n**)** **{** \_N **=** n**;** **}**

/// <summary>

/// Колличество букв алфавита+цифр+пробел

/// </summary>

**private** int \_N**;**

/// <summary>

/// Выполнить зашифровку входного текста

/// </summary>

/// <param name="textIn">Строка для шифровки</param>

/// <param name="gamma">Ключ для шифровки</param>

/// <param name="flag">true - режим шифровки вкл</param>

**internal** string CriptoTextIn**(**string textIn**,** string gamma**,** bool flag**)**

**{**

//index+1 равен позиций символа в алфавите

string alf **=** global**::**CryptoMona**.**Properties**.**Settings**.**Default**.**Ru**;**

//выравниваем строки

**if** **(**textIn**.**Length **>** gamma**.**Length**)** **while** **(**gamma**.**Length **<** textIn**.**Length**)** gamma **+=** gamma**;**

int i **=** textIn**.**Length**;**

//перегнать символы в числа a=1, b=2, ...// 0 такого символа нет меняем на 34 пробел

string numT **=**

**(**textIn**.**Aggregate**(**""**,**

**(**current**,** c**)** **=>**

current **+** **(**alf**.**IndexOf**(**c**.**ToString**(),** StringComparison**.**OrdinalIgnoreCase**)** **+** 1 **+** " "**))).**

Replace**(**" 0"**,** " 34"**);**

string numG **=**

**(**gamma**.**Aggregate**(**""**,**

**(**current**,** c**)** **=>**

current **+** **(**alf**.**IndexOf**(**c**.**ToString**(),** StringComparison**.**OrdinalIgnoreCase**)** **+** 1 **+** " "**))).**

Replace**(**" 0"**,** " 34"**);**

**return** **!**flag **?** Cripto**(**numT**,** numG**,** i**)** **:** Decripto**(**numT**,** numG**,** i**);**

**}**

/// <summary>

/// Формула (Символ[i]+Гамма[i])mod\_N

/// </summary>

/// <param name="numT"></param>

/// <param name="numG"></param>

/// <returns></returns>

**private** string Cripto**(**string numT**,** string numG**,** int index**)**

**{**

string alf **=** global**::**CryptoMona**.**Properties**.**Settings**.**Default**.**Ru**;**

var t **=** numT**.**Split**(**' '**);**

var g **=** numG**.**Split**(**' '**);**

string outText **=** ""**;**

//складываем

**for** **(**int i **=** 0**;** i **<** index**;** i**++)**

**{**

var sum **=** int**.**Parse**(**t**[**i**])** **+** int**.**Parse**(**g**[**i**]);**

var mod **=** sum**%**\_N**;**

t**[**i**]** **=** mod **==** 0 **?** \_N**.**ToString**()** **:** mod**.**ToString**();**

//перегнать числа в буквы

outText **+=** alf**[**int**.**Parse**(**t**[**i**])-**1**];**

**}**

**return** outText**;**

**}**

/// <summary>

/// Формула (Символ[i]-Гамма[i]+N)mod\_N

/// </summary>

/// <param name="textIn"></param>

/// <param name="gamma"></param>

/// <returns></returns>

**private** string Decripto**(**string numT**,** string numG**,** int index**)**

**{**

string alf **=** global**::**CryptoMona**.**Properties**.**Settings**.**Default**.**Ru**;**

var t **=** numT**.**Split**(**' '**);**

var g **=** numG**.**Split**(**' '**);**

string outText **=** ""**;**

//складываем

**for** **(**int i **=** 0**;** i **<** index**;** i**++)**

**{**

var ras **=** int**.**Parse**(**t**[**i**])** **-** int**.**Parse**(**g**[**i**]);**

var mod **=** **(**ras **+** \_N**)%**\_N**;**

t**[**i**]** **=** mod **==** 0 **?** \_N**.**ToString**()** **:** mod**.**ToString**();**

//перегнать числа в буквы

outText **+=** alf**[**int**.**Parse**(**t**[**i**])** **-** 1**];**

**}**

**return** outText**;**

**}**

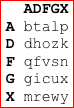
**}**

**}**

ADFGX

Bпервые был использован во время решающих этапов Первой мировой войны, когда в марте 1918 г. кайзеровские генералы начали крупное наступление. Шифр ADFGX был разработан полковником Фрицем Небелем, офицером связи, служившим в штабе германской армии. Шифра предполагает вначале применение к буквам исходного сообщения замену, после чего для получения окончательной шифрограммы выполняется перестановка.

В матрицу 5х5 случайным образом записывают (определенным образом) буквы латинского алфавита. Используем слово-ключ (например, *formula*).

**Сообщение**: one two three 

1) каждой букве сообщения ставим в соответствие 2 буквы (по таблице):

2) полученный текст записываем под ключом:



3) переставляем столбцы так, чтобы буквы ключа упорядочились по алфавиту:

4) записываем буквы из полученной матрицы (по столбцам) в шифр:

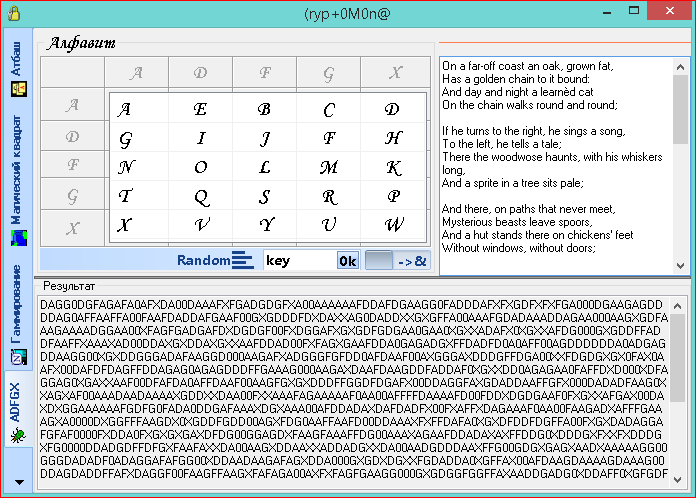
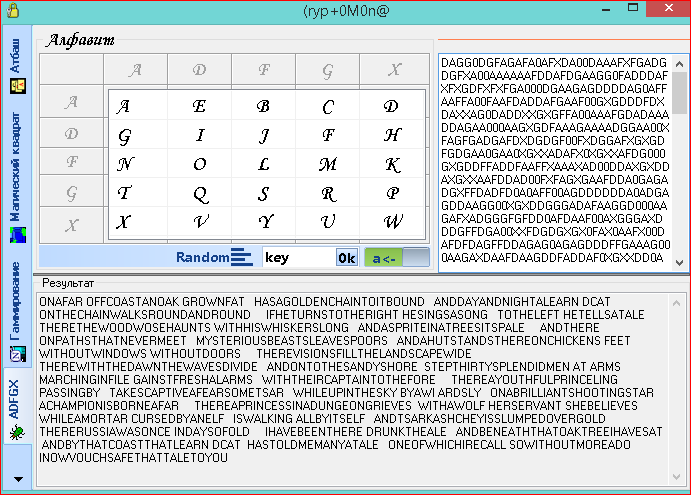
**Сообщение**: one two three

**Шифр**: ADXF DDDF FAFX XDDX FXDX FGXF XFXF

Для расшифровки необходимо знать три вещи: ключ, шифр, таблицу с алфавитом.

Процесс выполняется в обратном порядке: ключ выставляется буквами по алфавиту а столбцы заполняют из шифра, далее ключ приводят к исходному виду и вместе с этим меняются индексы столбцов таблицы. Выписываем построчно полученные координаты и ищем через них символ в таблице.

Плюсом алгоритма является то, что исчезновение одного символа ведет к нарушению общего смысла всей шифро посылки, так же прибавляет рандомная таблица с алфавитом.

А минус в том, что на один символ уходит аж две буквы шифра, расточительно.

Код алгоритма:

//http://habrahabr.ru/post/116716/

**namespace** CryptoMona

**{**

/// <summary>

/// Шифр ADFGX

/// </summary>

class clAdfgx

**{**

**public** clAdfgx**()**

**{**

\_list **=** **new** KeyValuePair**<**char**,** KeyValuePair**<**char**,** char**>>[**25**];**

**}**

**private** KeyValuePair**<**char**,** KeyValuePair**<**char**,** char**>>[]** \_list**;**

/// <summary>

/// Получить таблицу с рандомным алфавитом

/// </summary>

/// <param name="listView\_Aalf"></param>

/// <returns></returns>

**internal** ListView ListAlf**(**ListView listView\_Aalf**)**

**{**

string alf **=** global**::**CryptoMona**.**Properties**.**Settings**.**Default**.**En**;**

var rnd **=** **new** Random**();**

var navigate **=** **new** **[]** **{**'A'**,** 'D'**,** 'F'**,** 'G'**,** 'X'**};**

**for** **(**int j **=** 0**,** k **=** 0**,** indx **=** 0**;** j **<** alf**.**Length**-**1**;** j**+=**5**,** indx**++)**

**{**

var mass **=** **new** char**[**5**];**

//пять букв

string tmp **=** alf**.**Substring**(**j**,** 5**);**

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

**{**

int pos**;**

//берем первую

char a **=** tmp**[**i**];**

**do**//проверить полученное

**{**//зарандомить

pos **=** rnd**.**Next**(**0**,** 5**);**

**}** **while** **(!**mass**[**pos**].**Equals**(**'\0'**));**

//записать в ячейку

mass**[**pos**]** **=** a**;**

**}**

var item **=**

**new** ListViewItem**(new[]**

**{**

mass**[**0**].**ToString**(),** mass**[**1**].**ToString**(),** mass**[**2**].**ToString**(),** mass**[**3**].**ToString**(),**

mass**[**4**].**ToString**()**

**});**

listView\_Aalf**.**Items**.**Add**(**item**);**

**for** **(**var i **=** 0**;** i **<** 5**;** i**++,** k**++)**

**{**

\_list**[**k**]** **=** **new** KeyValuePair**<**char**,** KeyValuePair**<**char**,** char**>>(**mass**[**i**],** **new** KeyValuePair**<**char**,** char**>(**navigate**[**indx**],** navigate**[**i**]));**

**}**

**}**

**return** listView\_Aalf**;**

**}**

/// <summary>

/// Зашифровать сообщение

/// </summary>

/// <param name="text"></param>

/// <param name="key"></param>

**internal** string Cripton**(**string text**,** string key**,** bool flagCriptoOn**)**

**{**

**return** flagCriptoOn **?** Расшифровать**(**text**,** key**)** **:** Зашифровать**(**text**,** key**);**

**}**

**private** string Зашифровать**(**string textIn**,** string key**)**

**{**

//выдернуть все пробелы

var txt **=** textIn**.**Replace**(**" "**,** ""**);**

//количество строк в массиве

var len **=** Math**.**Ceiling**(((**double**)(**2 **\*** txt**.**Length**)** **/** key**.**Length**));**

var arr **=** **new** char**[(**int**)** len**,**key**.**Length**];**

var flagGroupX **=** **false;** var flagGroupY **=** **false;**

//Записать под ключем

**for** **(**int i **=** 0**,** k **=** 0**,** groupXY **=** 0**;** i **<** **(**int**)**len**;** i**++)**//строки

**{**

**for** **(**int j **=** 0**;** j **<** key**.**Length**;** j**++,** groupXY**++)**//столбцы

**{**

**if** **(**k **==** txt**.**Length**)** **break;**

**if** **(**groupXY **%** 2 **==** 0**)**

**{**

arr**[**i**,** j**]** **=** XY**(**txt**[**k**]).**Key**;**

flagGroupX **=** **true;**

**}**

**else**

**{**

arr**[**i**,** j**]** **=** XY**(**txt**[**k**]).**Value**;**

flagGroupY **=** **true;**

**}**

**if** **(**flagGroupX **==** flagGroupY**)**

**{**

k**++;**

flagGroupX **=** flagGroupY **=** **false;**

**}**

**}**

**}**

//сортировать по алфавиту ключ и столбцы

var akey **=** KeyAlfSort**(**key**);**

//переставить столбцы

**return** ReversColl**(**key**,** akey**,** arr**,** **(**int**)**len**,** key**.**Length**);**

**}**

/// <summary>

/// Сортировка без учета регистра

/// </summary>

/// <param name="key">Строка для сортировки</param>

/// <returns>Строка по алфавиту</returns>

**private** string KeyAlfSort**(**IEnumerable**<**char**>** key**)**

**{**

var sort **=** key**.**OrderBy**(**n **=>** n**.**ToString**(),** StringComparer**.**OrdinalIgnoreCase**);**

**return** sort**.**Aggregate**(**""**,** **(**current**,** srt**)** **=>** current **+** srt**);**

**}**

/// <summary>

/// Переставить столбцы ориентир ключ по алфавиту

/// </summary>

/// <param name="key">Ключ</param>

/// <param name="akey">Отсортированный ключ</param>

/// <param name="arr">Таблица координат</param>

/// <param name="arrRow">Количество строк в таблице</param>

/// <param name="arrCol">Количество столбцов в таблице</param>

/// <returns>Вернет шифрованую последовательность</returns>

**private** string ReversColl**(**string key**,** string akey**,** char**[,]** arr**,** int arrRow**,** int arrCol**)**

**{**

var arrNew **=** **new** char**[**arrRow**,** arrCol**];**

//первый символ нового ключа//какой индекс у этого символа в исходном ключе

**for** **(**var i **=** 0**;** i **<** key**.**Length**;** i**++)**

**{**

var chr **=** key**[**i**];**

var indx **=** akey**.**IndexOf**(**chr**,** 0**);**

akey **=** String**.**Concat**(**akey**.**Substring**(**0**,** indx**+**1**).**Replace**(**chr**,** ' '**),** akey**.**Substring**(**indx**+**1**));**//занулить найденое

//по индексу найти столбец и записать первым

**for** **(**var j **=** 0**;** j **<** arrRow**;** j**++)**//по строке

arrNew**[**j**,** i**]** **=** arr**[**j**,** indx**];**//[str, stolb]

**}**

//выписать по столбцам

var res **=** ""**;**

**for** **(**var i0 **=** 0**;** i0 **<** arrCol**;** i0**++)**

**for** **(**var i1 **=** 0**;** i1 **<** arrRow**;** i1**++)**

res **+=** arrNew**[**i1**,** i0**];**

//глушим пустые ячейки при key=3\_5\_7..

**return** res**.**Replace**(**'\0'**,** '0'**);**

**}**

**private** string Расшифровать**(**string textIn**,** string key**)**

**{**

**if** **(**textIn**.**Length **%** 2 **!=** 0**)** **return** ""**;**

//ключ выстраиваем по алфавиту

var akey **=** KeyAlfSort**(**key**);**

//выдернуть все пробелы\неликвид<=>0

//var txt = textIn.Replace("0", "");

//количество строк в массиве

var len **=** Math**.**Ceiling**(((**double**)**textIn**.**Length **/** key**.**Length**));**

var arr **=** **new** char**[(**int**)**len**,** key**.**Length**];**

//textIn записать по столбцам

**for** **(**int i **=** 0**,** indx **=** 0**;** i **<** key**.**Length**;** i**++)**//фиксируем столбец

**for** **(**var j **=** 0**;** indx **<** textIn**.**Length **&** j **<** **(**int**)**len**;** j**++,** indx**++)**//идем по строке

arr**[**j**,** i**]** **=** textIn**[**indx**];**

//выполнить перестановку с восстановлением индексов по ключу

//переставить столбцы

var str **=** ReversColl**(**key**,** akey**,** arr**,** **(**int**)**len**,** key**.**Length**);**

/\*str = str.Replace("0", "");\*/

var arrOut **=** **new** char**[(**int**)**len**,** key**.**Length**];**

//записать в столбцы

**for** **(**int i **=** 0**,** indx **=** 0**;** i **<** key**.**Length**;** i**++)**//фикс столб

**{**

**for** **(**int j **=** 0**;** /\*indx < str.Length & \*/j **<** **(**int**)**len**;** j**++,** indx**++)**//переб строк

**{**

arrOut**[**j**,** i**]** **=** str**[**indx**];**

**}**

**}**

var strOut **=** ""**;**

var pairA **=** ' '**;** var pairB **=** ' '**;**

var flagGroupX **=** **false;** var flagGroupY **=** **false;**

**for** **(**int i **=** 0**,** groupXY **=** 0**;** i **<** **(**int**)**len**;** i**++)**//строки

**{**

**for** **(**int j **=** 0**;** j **<** key**.**Length**;** j **++,** groupXY**++)**//cтолбцы

**{**

**if** **(**groupXY **%** 2 **==** 0**)**

**{**

pairA **=** arrOut**[**i**,** j**];**

flagGroupX **=** **true;**

**}**

**else**

**{**

pairB **=** arrOut**[**i**,** j**];**

flagGroupY **=** **true;**

**}**

**if** **(**flagGroupX **==** flagGroupY**)**

**{**

strOut **+=** Simbol**(new** KeyValuePair**<**char**,** char**>(**pairA**,** pairB**));**

flagGroupX **=** flagGroupY **=** **false;**

pairA **=** ' '**;** pairB **=** ' '**;**

**}**

**}**

**}**

**return** strOut**.**Replace**(**'0'**,** ' '**);**

**}**

/// <summary>

/// Запросить координаты символа

/// </summary>

/// <param name="simbol"></param>

/// <returns>строка\_столбец</returns>

**private** KeyValuePair**<**char**,**char **>** XY**(**char simbol**)**

**{**

**foreach** **(**var keyValuePair **in**

\_list**.**Where**(**keyValuePair **=>** String**.**Equals**(**keyValuePair**.**Key**.**ToString**(),** simbol**.**ToString**(),** StringComparison**.**OrdinalIgnoreCase**)))**

**{**

**return** keyValuePair**.**Value**;**

**}**

**return** **new** KeyValuePair**<**char**,** char**>(**'0'**,** '0'**);**

**}**

/// <summary>

/// Запросить символ по координате

/// </summary>

/// <param name="coordinate">Координаты</param>

/// <returns>символ</returns>

**private** char Simbol**(**KeyValuePair**<**char**,** char**>** coordinate**)**

**{**

**foreach** **(**var keyValuePair **in** \_list**.**Where**(**keyValuePair **=>** keyValuePair**.**Value**.**Equals**(**coordinate**)))**

**return** keyValuePair**.**Key**;**

**return** '0'**;**

**}**

**}**

**}**