1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 2**

1. «**Основы частотного криптоанализа**»
2. по дисциплине «Основы информационной безопасности»
3. Выполнил
4. студент гр. 4831001\20001 Маронова К.Д.

<*подпись*>

1. Преподаватель
2. асс. преподавателя Пахомов М.А.

<*подпись*>

1. Санкт-Петербург
2. 2023
3. **Цель работы**

В работе необходимо приобрести навыки криптоанализа, ознакомиться со способом дешифрования криптограмм на примере применения метода частного криптоанализа.

1. **Постановка задачи**

Необходимо разработать программу, реализующую функции инструмента криптоаналитика. Кроме того, с использованием данной программы нужно расшифровать данную криптограмму.

1. **Теоретические исследования**

Работа алгоритма шифрования при помощи шифра моноалфавитной подстановки заключается в следующем:

- выбирается набор символов, которые будут использоваться для составления текста, требующего шифрования;

- выбирается алфавит шифрования;

- устанавливается взаимно однозначное соответствие между символами нормативного алфавита и символами алфавита шифрования.

Моноалфавитные подстановки обладают важным свойством – они не нарушают частот появления символов, характерных для данного языка. Все естественные языки имеют характерное частотное распределение символов. Например, буква «О» встречается в русском языке чаще других, а буква «Ъ» − самая редкая.

Для того, чтобы получить открытый текст при помощи частотного анализа, необходимо сопоставить частоты появления символов шифра с вероятностями появления букв используемого алфавита. После этого наиболее частые символы криптограммы заменяются наиболее вероятными символами алфавита, остальные замены производятся на основе вероятных слов и знания синтаксических правил используемого языка.

1. **Описание решения**

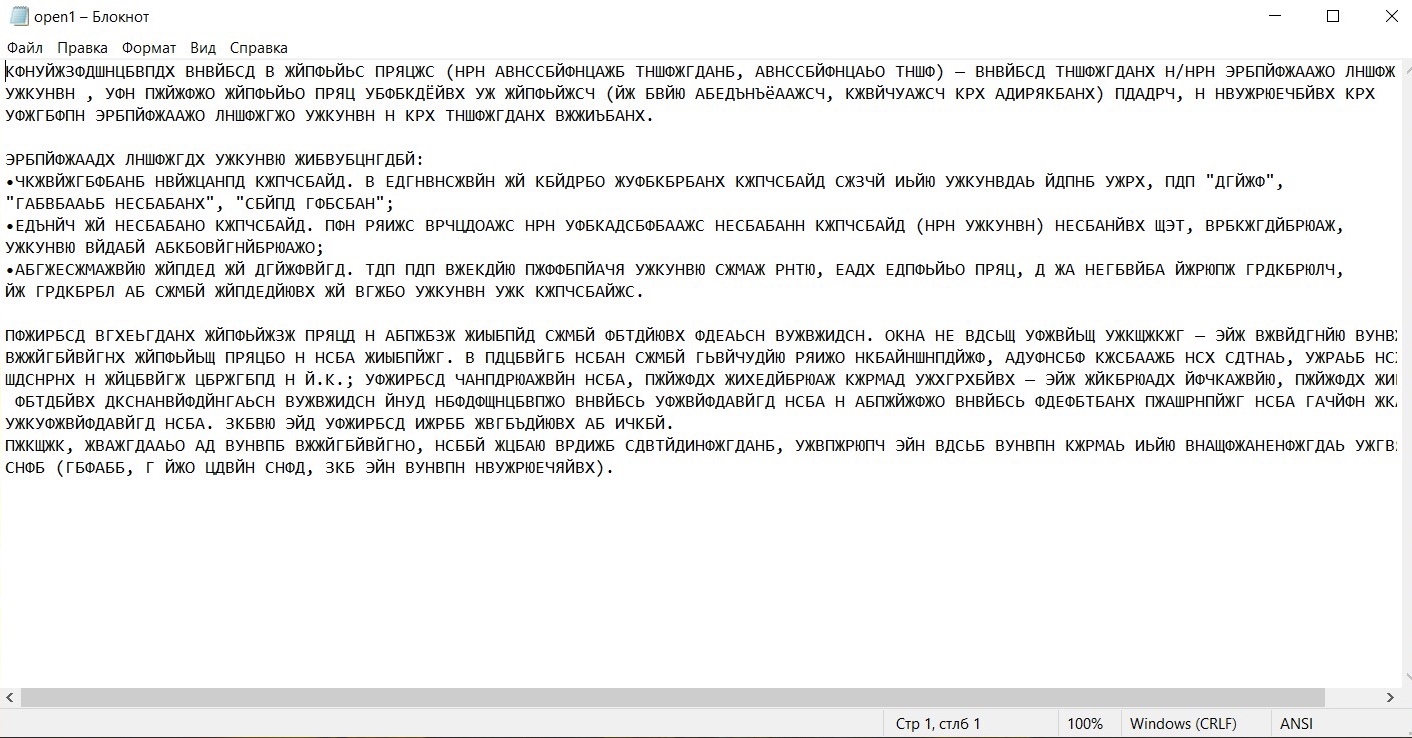
* Исходная криптограмма

Рисунок 1 – Исходная криптограмма

* Реализация:

В функции main файл с криптограммой открывается для чтения и затем текст записывается в глобальный двумерный массив history с текущим шагом 0. Затем вызывается функция (interaction ()) меню взаимодействия с пользователем, состоящим из возможных операций. Данная функция возвращает значение, введенное пользователем (чисто от 1 до 7) и в зависимости от этого далее выполняется одна из операций с криптограммой. Кроме того, выполняется локализация для считывания и работы с символами русского алфавита.

Функция анализа частот букв в тексте: void analiz(). Создается и обнуляется массив count типа float для вычисления частоты каждой буквы. После этого массив состоящий из букв русского алфавита сортируется методом пузырька и на экран выводятся предполагаемые замены.

Функция группировки по кол-ву букв: void groupByLen().

С помощью функции strtok производится разбиение текста на слова по разделителям. Они закидываются в массив, а затем сортируются пузырьком по длине с помощью strlen и выводятся на экран.

Функция сортировки по кол-ву нерасшифрованных на данный момент букв: void groupByCryptography.

С помощью функции strtok производится разбиение текста на слова по разделителям. Они закидываются в массив, а затем сортируются пузырьком по расшифрованным буквам с помощью checkCryptography(ведет счетчик расшифрованных букв в слове) и выводятся на экран. (строчные буквы – расшифрованные буквы, прописные – не расшифрованные).

Функция замены букв: void replace(). Запрашивает у пользователя 2 буквы: ту которую он хочет заменить и ту на которую он хочет заменить. Затем производится замена букв. Попутно увеличивается значение текущего шага (+1). Чтобы в последствии отслеживать шаги и делать откат.

Функция автозамены: void avtozamena(). Выполняет полный перебор всех возможных перестановок, основываясь на криптоанализе из п.1. И поэтапно подставляет в наш текст.

Функция отмены действия: void backup(). Показывает состояние текста на шаге = currentStep-step(введен пользователем).

Функция истории: void showCurrentCryptogramState(). Выводит на экран состояние текста на текущем шаге.

1. **Тестирование и результаты работы программы**

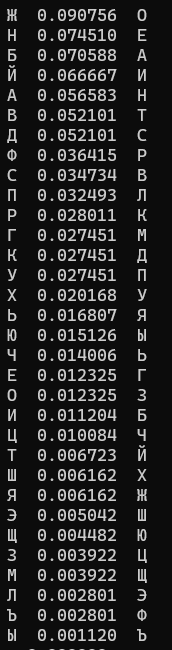
****

Рисунок 2 – Частотный анализ криптограммы

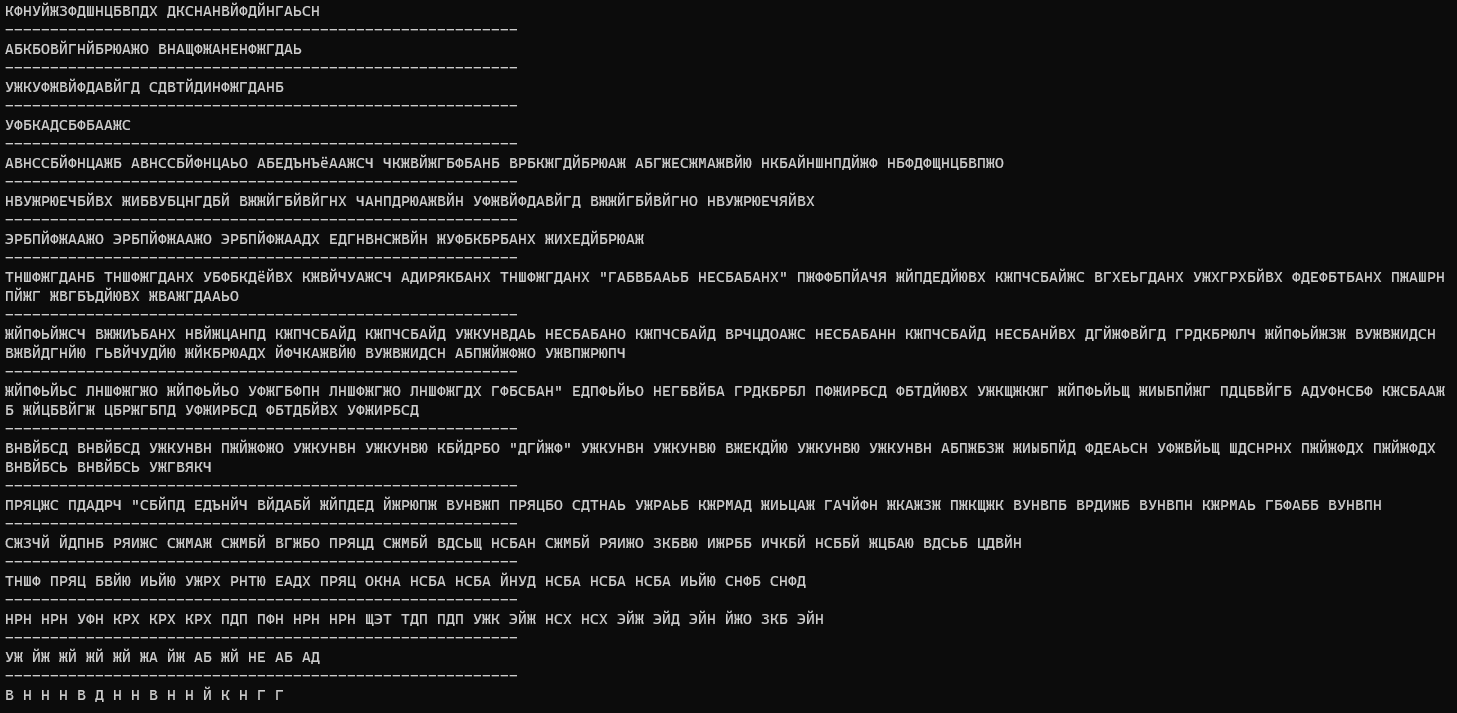


Рисунок 3 – Сортировка по кол-ву букв

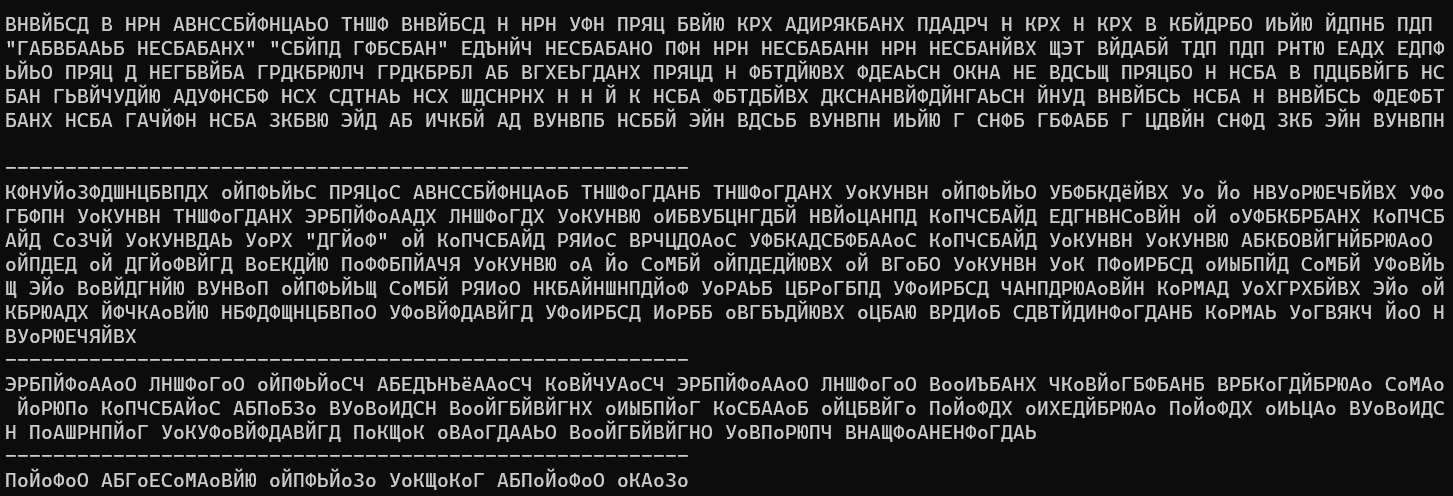


Рисунок 4 – Сортировка по кол-ву нерасшифрованных букв на некоторой интеграции цикла

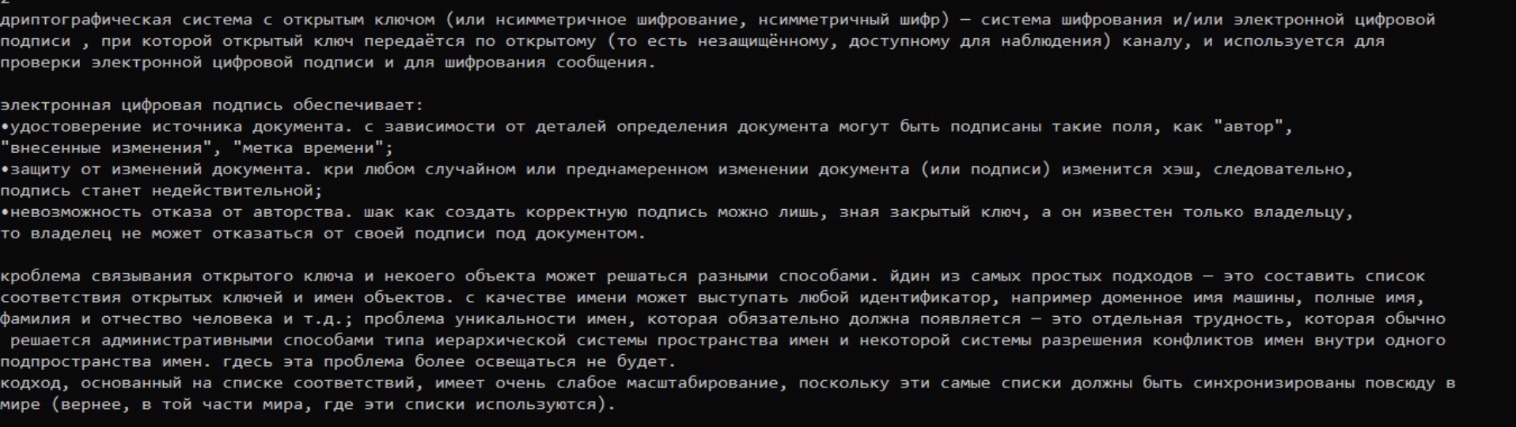


Рисунок 7 – Расшифрованная криптограмма

Таблица 1 – Ключ

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |
| **д** | **и** | **г** | **з** | **к** | **б** | **м** | **е** | **н** | **о** | **п** | **р** | **с** | **а** | **ж** | **у** | **ф** | **в** | **й** | **ч** | **ш** | **щ** | **л** | **ц** | **т** | **ъ** |
| Ъ | Ы | Ь | Э | Ю | Я |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **ю** | **ь** | **ю** | **э** | **я** | **х** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

1. **Выводы**

В ходе работы была разработана программу, реализующую функции инструмента криптоаналитика. И с использованием данной программы была расшифрована криптограмма. Были получены знания в сфере дешифровки и изучен шифр моноалфавитной подстановки.

ПРИЛОЖЕНИЕ А

Листинг программы «Основы частотного криптоанализа»

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <Windows.h>

#define STEPS 35 //по мощности алфавита (33 буквы) (тк сколько различных букв - столько замен мы и можем совершить)

#define delitels " ,-.:;\n•()—/ "

#define MAX\_WORD\_LENGTH 30

#define MAX\_WORD\_COUNT 500

#define MAX\_FILE\_LENGTH 10000

#define len 32

int length;

int\* st;

char\* v;

char history[STEPS][MAX\_FILE\_LENGTH];

int currentStep = 0;

FILE\* file;

char alfavit[33] = "АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ";

char chastota[33] = "ОЕАИНТСРВЛКМДПУЯЫЬГЗБЧЙХЖШЮЦЩЭФЪ";

void analiz() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

FILE\* file;

file = fopen("open3.txt", "r");

float count[33]; //кол-во каждой конкретной буквы в тексте

int countletter = 0;//общее кол-во букв в тексте

char b;

for (int i = 0; i < 33; i++) {

count[i] = 0;

}

while ((b = fgetc(file)) != EOF) {

countletter++;

//b = fgetc(file);

for (int f = 0; f < 34; f++) {

if (b == alfavit[f]) {

count[f]++;

}

}

}

for (int i = 0; i < 33; i++) {

count[i] = count[i] / countletter;

}

float tmp; char btmp;

for (int i = 33 - 1; i >= 0; i--)

{

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

{

if (count[j] < count[j + 1])

{

tmp = count[j];

count[j] = count[j + 1];

count[j + 1] = tmp;

btmp = alfavit[j];

alfavit[j] = alfavit[j + 1];

alfavit[j + 1] = btmp;

}

}

}

for (int i = 0; i < 33; i++) {

printf("%c", alfavit[i]);

printf(" ");

printf("%f", count[i]);

printf(" ");

printf("%c\n", chastota[i]);

}

fclose(file);

printf("\n\n");

}

void groupByLen() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int g;

int wordCount;

char words[MAX\_WORD\_COUNT][MAX\_WORD\_LENGTH];

char currentString[MAX\_FILE\_LENGTH];

strcpy(currentString, history[currentStep]);

char\* word = strtok(currentString, delitels); // Разбиваем текст на слова и копируем слова в массив слов

for (wordCount = 0; word != NULL; word = strtok(NULL, delitels), wordCount++) {

strcpy(words[wordCount], word);

}

char tmp[MAX\_FILE\_LENGTH];

for (int i = wordCount - 1; i >= 0; i--)

{

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

{

if (strlen(words[j]) < strlen(words[j + 1]))

{

strcpy(tmp, words[j]);

strcpy(words[j], words[j + 1]);

strcpy(words[j + 1], tmp);

}

}

}

int maxlen = strlen(words[0]); int teklen;//выводим слова блоками по одинаковой длине, разделяю новой строкой и тире

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

teklen = strlen(words[i]);

if (teklen == maxlen) {

printf(words[i]);

printf(" ");

}

else {

maxlen = teklen;

printf("\n---------------------------------------------------------\n");

printf(words[i]);

printf(" ");

}

}

printf("\n");

}

int checkCryptography(char\* str) {

int count = 0; int i = 0;

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

if (!str[i]) return count;

if (str[i] > 'а' && str[i] < 'я' || str[i] == 'ё') count++;

}

}

void groupByCryptography() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int wordCount; int g;

char words[MAX\_WORD\_COUNT][MAX\_WORD\_LENGTH];

char currentString[MAX\_FILE\_LENGTH];

strcpy(currentString, history[currentStep]);

char\* word = strtok(currentString, delitels); // Разбиваем текст на слова и копируем слова в массив слов

for (wordCount = 0; word != NULL; word = strtok(NULL, delitels), wordCount++) {

strcpy(words[wordCount], word);

}

char tmp[MAX\_FILE\_LENGTH];

for (int i = wordCount - 1; i >= 0; i--)

{

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

{

if (checkCryptography(words[j]) > checkCryptography(words[j + 1]))

{

strcpy(tmp, words[j]);

strcpy(words[j], words[j + 1]);

strcpy(words[j + 1], tmp);

}

}

}

int maxlen = checkCryptography(words[0]); int teklen;//выводим слова блоками по одинаковой длине, разделяю новой строкой и тире

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

teklen = checkCryptography(words[i]);

if (teklen == maxlen) {

printf(words[i]);

printf(" ");

}

else {

maxlen = teklen;

printf("\n---------------------------------------------------------\n");

printf(words[i]);

printf(" ");

}

}

printf("\n");

}

void showCurrentCryptogramState() {

printf("Текущее состояние частично расшифрованной криптограммы (строчные буквы - дешифрованное, прописные - ещё зашифрованное):\n"); printf("\n");

printf(history[currentStep]);

printf("\n\n");

}

void replace() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char a[2];

char b[2];

printf("Введите букву, которую хотите заменить:");

scanf("%s", &a[0]);

printf("Введите букву, НА которую хотите заменить(строчную):");

scanf("%s", &b[0]);

strcpy(history[currentStep + 1], history[currentStep]); //копируем текущий шаг на последующий

++currentStep;//увеличиваем значение текущего шага

char\* newCryptogram = history[currentStep]; //записываем то, что лежит в нем в текущую переменную

for (int i = 0; newCryptogram[i] != NULL; i++) {

if (newCryptogram[i] == a[0]) {

newCryptogram[i] = b[0];

}

}

showCurrentCryptogramState();

}

void backup() {

int step;

printf("Сколько шагов вы хотите удалить?");

scanf("%d", &step);

if (step >= currentStep) currentStep = 0;

else currentStep -= step;

printf("Текущий шаг: %u\n", currentStep);

showCurrentCryptogramState();

}

int interaction() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int a;

printf("Выберите операцию:\n\

1.Анализ частоты\n\

2.Группировка по кол-ву букв\n\

3.Группировка по кол-ву нерасшифрованных букв\n\

4.Показать криптограмму\n\

5.Замена букв\n\

6.Отмена действия\n\

7.Автозамена\n");

scanf\_s("%d", &a);

return a;

}

//ПЕРЕСТАНОВКИ ДЛЯ АВТОЗАМЕНЫ

void Sort(char\* mass, int \_len)

{

int i, j;

char tmp;

for (i = 0; i < \_len; i++)

{

for (j = 0; j < \_len - i - 1; j++)

{

if (mass[j] > mass[j + 1])

{

tmp = mass[j];

mass[j] = mass[j + 1];

mass[j + 1] = tmp;

}

}

}

}

void Initialize(char\* \_v, int \_length)

{

int i;

length = \_length;

v = (char\*)malloc((length \* sizeof(char)) + 1);

st = (int\*)malloc(length \* sizeof(int));

for (i = 0; i < length; i++)

v[i] = \_v[i];

v[length] = 0;

for (i = 0; i < length; i++)

st[i] = 0;

}

void Swap(char\* a, char\* b)

{

char t = \*a;

\*a = \*b;

\*b = t;

}

void Reverse(char\* v, int start)

{

int i;

for (i = 0; i < (length - start) / 2; i++)

Swap(&(v[start + i]), &(v[length - 1 - i]));

}

BOOL NextPermutation()

{

int pos, i;

if (length < 2) return FALSE;

pos = length - 2;

while (pos >= 0)

{

if (st[pos] < length - 1 - pos)

{

Reverse(v, pos + 1);

for (i = pos + 1; i < length; )

st[i++] = 0;

st[pos]++;

Swap(&(v[pos]), &(v[pos + st[pos]]));

return TRUE;

}

pos--;

}

return FALSE;

}

void avtozamena() {

FILE\* file;

file = fopen("result.txt", "a+");

char am[] = "бвгдежзиймнкларыьопстуфхцчшщъэюяклары";

char am2[] = "абвгдежзийпрстуфхцчшщъыьэюяклмно";

Sort(am, len);

Initialize(am, len);

char newtext[MAX\_WORD\_COUNT];

int k = 0;

do {

strcpy(newtext, history[currentStep]);

for (int i = 0; newtext[i]!='\0'; i++) {

for (int j = 0; j < 33; j++) {

if ((newtext[i]+32) == am2[j]) {

newtext[i] = v[j];

}

}

}

fprintf(file, newtext);

fprintf(file, "\n");

} while (NextPermutation());

fclose(file);

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

FILE\* file;

file = fopen("open3.txt", "r");

char c; int f = 0;

while ((c = fgetc(file)) != EOF) {

history[currentStep][f] = c;

f++;

}

history[currentStep][f] = '\0';

int p = 0;

while (p != 8) {

int p = interaction();

if (p == 1) {

printf("\n");

analiz();

}

if (p == 2) {

printf("\n");

groupByLen();

}

if (p == 3) {

printf("\n");

groupByCryptography();

}

if (p == 4) {

printf("\n");

showCurrentCryptogramState();

}

if (p == 5) {

printf("\n");

replace();

}

if (p == 6) {

printf("\n");

backup();

}

if (p == 7) {

printf("\n");

avtozamena();

}

}

}