Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

Лабораторная работа № 2

**«Разработка программы шифрования/дешифрирования текста с**

**использованием перестановочного шифра.»**

## Отчёт по дисциплине

**«Методы защиты КС»**

Пенза 2020

**Выполнили с-ты гр. 17ВВ2:**

Живаев В.  
Безжонов Р.

**Приняли:**

к.т.н. Дубравин А.В.  
Кармышева Н.С.

**Цель работы.**

Научиться разрабатывать программы шифрования/дешифрования текста с использованием перестановочного шифра.

**Задание.**

Разработать программу, выполняющую шифрование ирасшифровывание произвольного текстового файла с использованиемперестановочного шифра используя в качестве ключа последовательность,соответствующую номеру варианта (табл. 1). Выполнить проверку путем двоичногосравнения исходного файла и фала, полученного после расшифровывания.

Таблица 1 - Вариант задания.

|  |  |
| --- | --- |
| № Варианта | Последовательность перестановки |
| 6 | 2 5 9 3 4 10 1 7 6 8 |

**Листинг.**

// l2.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <fstream>

#include <iostream>

#include <Windows.h>

#include <conio.h>

#include <vector>

#include <math.h>

#include "l2.h"

#define BLOCK\_SIZE 5

typedef unsigned int BIT;

int key[] = { 12, 13, 30, 5, 27, 6, 11, 25, 3, 21, 22, 2, 23, 0, 8, 4, 18, 19, 10, 1, 14, 29, 9, 28, 20, 17, 26, 31, 7, 16, 15, 24 };

using namespace std;

bool debug = false;

wchar\_t\* original;

typedef unsigned int Block;

vector<char> vec1;

void argsParser(int argc, wchar\_t\* argv[]) {

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

if (lstrcmpW(argv[i], L"-h") == 0 || lstrcmpW(argv[i], L"--help") == 0) {

help();

}

if (lstrcmpW(argv[i], L"-d") == 0 || lstrcmpW(argv[i], L"--debug") == 0) {

if (isKeyValid(&argv[i], 2)) {

if (lstrcmpW(argv[i + 1], L"false") == 0) debug = false;

if (lstrcmpW(argv[i + 1], L"true") == 0) debug = true;

original = argv[i + 2];

}

}

if (lstrcmpW(argv[i], L"-e") == 0 || lstrcmpW(argv[i], L"--encrypt") == 0) {

if (isKeyValid(&argv[i], 1)) {

encryptFile(argv[i+1]);

}

}

if (lstrcmpW(argv[i], L"-de") == 0 || lstrcmpW(argv[i], L"--decrypt") == 0) {

if (isKeyValid(&argv[i], 1)) {

if (!debug) {

decryptFile(argv[i + 1]);

}

else

{

decryptFile(argv[i + 1]);

}

}

}

}

}

void help() {

printf("Для управления используются следующие ключи:\n");

printf("\t\t -h --help вызов помощи\n");

printf("\t\t -d --debug [true/false] [originalFile] вывод отладочной информации\n");

printf("\t\t -e --encrypt [inputFile outputFile] шифрование файла\n");

printf("\t\t -de --decrypt [inputFile outputFile] дешифровка файла\n");

}

bool isKey(wchar\_t\* arg) {

bool flag = false;

flag = lstrcmpW(arg, L"-e") ? false : true;

flag = lstrcmpW(arg, L"--encrypt") ? false : true;

flag = lstrcmpW(arg, L"-de") ? false : true;

flag = lstrcmpW(arg, L"--decrypt") ? false : true;

flag = lstrcmpW(arg, L"-h") ? false : true;

flag = lstrcmpW(arg, L"--help") ? false : true;

flag = lstrcmpW(arg, L"-d") ? false : true;

flag = lstrcmpW(arg, L"--debug") ? false : true;

return flag;

}

bool isPresent(wchar\_t\* arg) {

if (arg != NULL) {

return true; //проверка наличия аргумента

}

wcout << L"Не передан аргумент. Справка -h|--help" << endl;

exit(666);

}

bool isKeyValid(wchar\_t\* arg[], int paramsCount) {

isPresent(arg[0]);

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

if (isPresent(arg[i])) { //аргумент команды должен быть представлен и не должен являться ключем

isKey(arg[i]);

}

}

return true;

}

BIT GetBit(Block block, INT numBit)

{

return ((block & (1 << numBit)) != 0);

}

BIT SetBit(Block block, INT numBit)

{

return (block | (1 << numBit));

}

int unsetBit(Block block, int numBit)

{

return (block & ~(1 << numBit));

}

int cryptBlock(Block block)

{

Block tempBlock = 0;

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

{

int tempBit = GetBit(block, i);

if (tempBit == 1)

tempBlock = SetBit(tempBlock, key[i]);

}

return tempBlock;

}

int decryptBlock(Block block)

{

Block tempBlock = 0;

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

{

int tempBit = GetBit(block, key[i]);

if (tempBit == 1)

tempBlock = SetBit(tempBlock, i);

}

return tempBlock;

}

BOOL encryptFile(LPCWSTR fileName)

{

HANDLE hSourceFile, hDestFile;

DWORD dwReaded = 0, dwWrote = 0;

Block block = 0;

BOOL bResult = FALSE;

hSourceFile = CreateFile(fileName, GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hSourceFile == INVALID\_HANDLE\_VALUE)

{

printf("Error CreateFile! Error = %ld\n", GetLastError());

return FALSE;

}

hDestFile = CreateFile(TEXT("encrypted"), GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hDestFile == INVALID\_HANDLE\_VALUE)

{

printf("Error CreateFile! Error = %ld\n", GetLastError());

return -1;

}

DWORD dwSizeH = 0, dwSizeL = 0;

dwSizeL = GetFileSize(hSourceFile, &dwSizeH);

LONGLONG fileLength = ((LONGLONG)dwSizeH \* ((LONGLONG)MAXDWORD + 1)) + dwSizeL;

if (!WriteFile(hDestFile, &fileLength, sizeof(LONGLONG), &dwWrote, NULL))

{

printf("Error WriteFile! error = %ld\n", GetLastError());

return FALSE;

}

do

{

memset(&block, 0, BLOCK\_SIZE);

if (!(bResult = ReadFile(hSourceFile, &block, BLOCK\_SIZE, &dwReaded, NULL)))

{

printf("Error ReadFile! error = %ld\n", GetLastError());

return FALSE;

}

if (dwReaded)

{

block = cryptBlock(block);

if (!WriteFile(hDestFile, &block, BLOCK\_SIZE, &dwWrote, NULL))

{

printf("Error WriteFile! error = %ld\n", GetLastError());

return FALSE;

}

}

} while (bResult && dwReaded);

CloseHandle(hSourceFile);

CloseHandle(hDestFile);

return TRUE;

}

BOOL decryptFile(LPCWSTR fileName)

{

HANDLE hSourceFile, hDestFile;

DWORD dwReaded = 0, dwWrote = 0;

Block block = 0;

hSourceFile = CreateFile(fileName, GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hSourceFile == INVALID\_HANDLE\_VALUE)

{

printf("Error CreateFile! Error = %ld\n", GetLastError());

return FALSE;

}

hDestFile = CreateFile(TEXT("decrypted"), GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hDestFile == INVALID\_HANDLE\_VALUE)

{

printf("Error CreateFile! Error = %ld\n", GetLastError());

return -1;

}

LONGLONG sourceLength;

if (!ReadFile(hSourceFile, &sourceLength, sizeof(LONGLONG), &dwReaded, NULL))

{

printf("Error ReadFile! error = %ld\n", GetLastError());

return FALSE;

}

while (sourceLength > 0)

{

memset(&block, 0, BLOCK\_SIZE);

if (!ReadFile(hSourceFile, &block, BLOCK\_SIZE, &dwReaded, NULL))

{

printf("Error ReadFile! error = %ld\n", GetLastError());

return FALSE;

}

if (dwReaded)

{

block = decryptBlock(block);

if (!WriteFile(hDestFile, &block, sourceLength > BLOCK\_SIZE ? BLOCK\_SIZE : (DWORD)sourceLength, &dwWrote, NULL))

{

printf("Error WriteFile! error = %ld\n", GetLastError());

return FALSE;

}

}

sourceLength -= BLOCK\_SIZE;

}

CloseHandle(hSourceFile);

CloseHandle(hDestFile);

if (debug) {

string command = "FC ";

char\* temp;

int len = 0;

int totalLen = 3;

len = lstrlenW(TEXT("decrypted"));

totalLen += len;

temp = (char\*)malloc(sizeof(char) \* len);

wcstombs(temp, TEXT("decrypted"), len);

command.append(temp);

command = command.substr(0, totalLen);

free(temp);

command.append(" ");

totalLen++;

command = command.substr(0, totalLen);

len = lstrlenW(original);

totalLen += len;

temp = (char\*)malloc(sizeof(char) \* len);

wcstombs(temp, original, len);

command.append(temp);

command = command.substr(0, totalLen);

free(temp);

system(command.data());

}

return TRUE;

}

int \_tmain(int argc, wchar\_t\* argv[])

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

argsParser(argc, argv);

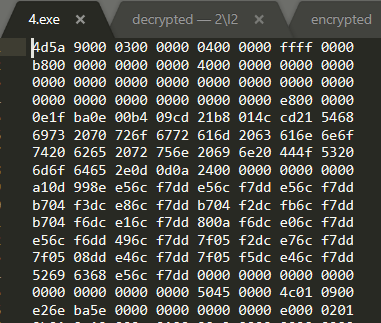
system("pause");

return 0;

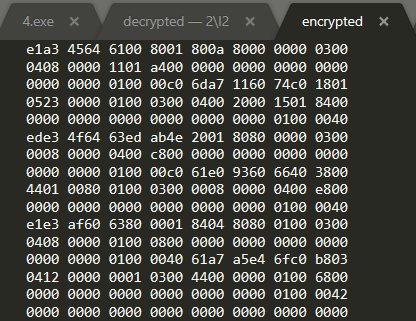
}

**Результат работы.**

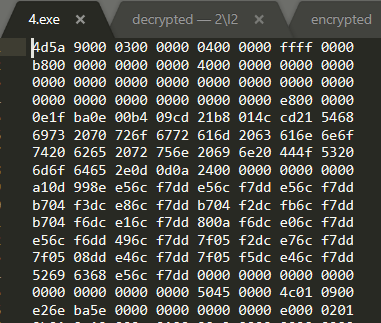
Результат работы программы показан на рисунках 1-5.



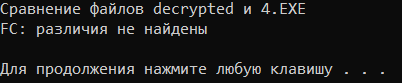
**Рисунок 1 - Исходный файл.**



**Рисунок 2 - Результат после шифрования.**



**Рисунок 3 - Результат после дешифрования.**



**Рисунок 4 - Результат проверки файлов.**

Исходный текст файла совпал с результатом дешифрования.

**Вывод.**

Научились разрабатывать программы шифрования/дешифрования текста с использованием перестановочного шифра.