МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего образования

«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»

ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ

Кафедра компьютерной инженерии и моделирования

**Симулятор шифровальной машины «Энигма»**

Курсовая работа

по дисциплине «Программирование»

студента 1 курса группы ИВТ-б-о-192(2)

Давкуш Максим Константинович

направления подготовки 09.03.01 «Информатика и вычислительная техника»

|  |  |  |
| --- | --- | --- |
| Научный руководитель  ассистент кафедры компьютерной инженерии и моделирования | \_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) | Чабанов В.В. |

Симферополь, 2020

**РЕФЕРАТ**

Объектами исследования являются изучение принципа работы шифровальной машины «Энигма».

Цель работы заключается в усвоении и закреплении основ, а также углублении знаний в области программирования.

**О**ГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc34004330)

[ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ 5](#_Toc34004331)

[1.1 Цель проекта 5](#_Toc34004332)

[1.2 Существующие аналоги 5](#_Toc34004333)

[ГЛАВА 2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ 6](#_Toc34004336)

[2.1 Анализ инструментальных средств 6](#_Toc34004337)

[2.2 Описание алгоритмов 6](#_Toc34004338)

[ГЛАВА 3 ТЕСТИРОВАНИЕ ПРОГРАММЫ 8](#_Toc34004341)

[3.1 Тестирование исходного кода 8](#_Toc34004342)

[3.2 Тестирование интерфейса пользователя и юзабилити 8](#_Toc34004343)

[ГЛАВА 4 ПЕРСПЕКТИВЫ ДАЛЬНЕЙШЕГО РАЗВИТИЯ ПРОЕКТА 9](#_Toc34004344)

[4.1 Перспективы технического развития 9](#_Toc34004345)

[4.2 Перспективы монетизации 9](#_Toc34004346)

[ЗАКЛЮЧЕНИЕ 10](#_Toc34004347)

[ЛИТЕРАТУРА 11](#_Toc34004348)

[ПРИЛОЖЕНИЕ 1 КОД ОСНОВНЫХ МОДУЛЕЙ ПРОЕКТА 12](#_Toc34004349)

ВВЕДЕНИЕ

Цель курсовой работы – усвоение и закрепление основ, а также углубление знаний в области программирования, приобретения дополнительных практических навыков в работе с шифром. При выполнении курсовой работы осуществляются все этапы создания программного продукта, от постановки задачи до практической реализации, сопровождающейся документацией и инструкциями по его использованию.

В процессе выполнения курсовой осваиваются навыки самостоятельного поиска и использования специальной литературы, каталогов, справочников, стандартов. Тематика заданий для курсовой работы может быть дополнена, расширена, увязана с решением актуальных научно-исследовательских задач.

ГЛАВА 1  
ПОСТАНОВКА ЗАДАЧИ

**1.1 Цель проекта**

Реализовать симулятор шифровальной машины «Энигма» позволяющий шифровать и расшифровывать сообщения.

**1.2 Существующие аналоги**

В приложении Play Market существует аналоги шифровальной машины.

ГЛАВА 2  
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

## 2.1 Анализ инструментальных средств

В своей работе не были использованы библиотеки, модули, фреймворки и другие программные средства, применяемые в процессе выполнения курсовой работы.

## 2.2 Описание алгоритмов

Для решения поставленной задачи воспользуемся алгоритмом, представленным на рисунке:

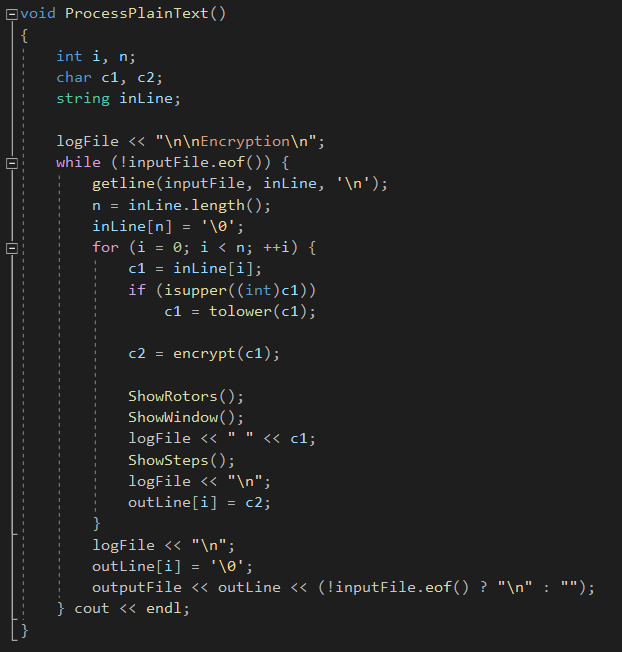


Рис. 1 – Функция «ProcessPlainText»

## Строки из входного файла обрабатываются по одной за раз. Символ новой строки ('\n') в конце каждой строки перезаписывается тернарным нулем. Каждый символ из строки преобразуется в нижний регистр (при необходимости), и его шифрование сохраняется в выходной строке.

После обработки входной строки выходная строка записывается в файл шифрования. Файл журнала обновляется с информацией о процессе шифрования.

ГЛАВА 3  
ТЕСТИРОВАНИЕ ПРОГРАММЫ

**3.1 Тестирование исходного кода**

Тест не проводился.

**3.2 Тестирование интерфейса пользователя и юзабилити**

Интерфейс пользователя отсутствует.

ГЛАВА 4  
ПЕРСПЕКТИВЫ ДАЛЬНЕЙШЕГО РАЗВИТИЯ ПРОЕКТА

**4.1 Перспективы технического развития**

Есть возможность реализации Юзер Интерфейса, благодаря которому упрощалось использование самой машины. В проекте был замечен недочет с вводом определенного количества символов в одну строку(а именно 256 символов), в будущем ее надо будет устранить.

**4.2 Перспективы монетизации**

Перспективы монетизации не преследуются.

ЗАКЛЮЧЕНИЕ

В результате были получены знания работы шифровальной машины «Энигма».

А также они были реализованы на языке программировании С++. Также ко всему прошло оттачивание уже имеющихся навыков и приобретение новых во время выполнения работы со средой программирования, создания самого кода и работы с разными источниками информации.

ЛИТЕРАТУРА

1. Энигма [https://ru.wikipedia.org/wiki](https://ru.wikipedia.org/wiki/%D0%AD%D0%BD%D0%B8%D0%B3%D0%BC%D0%B0)
2. Алгоритм Энигмы <https://habr.com/ru/post/217331/>
3. Как работала шифровальная машина «Энигма» <http://war-only.com/kak-rabotala-shifrovalnaya-mashina-enigma.html>
4. Криптоанализ «Энигмы» <https://habr.com/ru/post/269519/>
5. Оформление выпускной квалификационной работы на соискание квалификационного уровня «Магистр» («Бакалавр»): методические рекомендации. / сост. Бержанский В.Н., Дзедолик И.В., Полулях С.Н. – Симферополь: КФУ им. В.И.Вернадского, 2017. – 31 с.

**ПРИЛОЖЕНИЕ 1  
КОД ОСНОВНЫХ МОДУЛЕЙ ПРОЕКТА**

#include <stdio.h>

#include <cstring>

#include <cctype>

#include <vector>

#include <string>

#include <iostream>

#include "en.h"

#include <fstream>

using namespace std;

int mRotors, // Количество роторов, размещенных в машине

// (1-based: 1-4)

mSteps; // Фактическое количество шагов шифрования

// = 2\*mRotors + 2 (plugboard) + 1 (reflector)

int RotPos[Nrotors]; // Ротор помещен в каждое положение

char window[Nrotors], // Символы в окне

Iwindow[Nrotors]; // Начальные значения в 'window' (для сброса)

string RotWiring[Nrotors]; // Обмотки ротора

char RotNotch[Nrotors]; // Положения переключения Ротора

int RotNumber[Nrotors]; // Какой физический Ротор (t,1-8,b,g)

string reflector, // Проводка рефлектора

plugboard; // Провода в коммутационной панели

int ReflType; // Используемый рефлектор

char step[Nsteps]; // Массив для хранения шагов шифрования

// Файлы и переменные для настройки и отчетности

#define Nline 255

fstream inputFile, outputFile, logFile;

char outLine[Nline]; // выходная линия

void InitEnigma() // Инициализация по умолчанию

{

int i;

mRotors = 3;

mSteps = (mRotors << 1) + 3;

plugboard = PLUGBOARD;

for (i = 0; i <= mRotors; ++i) {

RotWiring[i] = ROTOR[i];

RotNotch[i] = NOTCH[i];

RotNumber[i] = i;

Iwindow[i] = window[i] = 'a';

}

reflector = REFLECTOR[1];

ReflType = 1;

}

void TryUserSetup()

// Попытка инициализации из пользовательского файла

{

inputFile.open("esetup", ios::in);

if(inputFile.is\_open())

{

SetPlugboard();

SetRotorsAndReflector();

inputFile.close();

}

}

void ProcessFile( const char\* inFname,

const char\* encFname,

const char\* logFname)

{

if (OpenFiles(inFname,

encFname,

logFname)) {

SetRotorPositions();

ReportMachine();

ProcessPlainText();

CloseFiles();

}

}

void reset()

{

for (int i = 1; i <= mRotors; ++i)

window[i] = Iwindow[i];

}

void main()

{

InitEnigma();

TryUserSetup();

ProcessFile("plain", "encrypt", "elog");

reset();

ProcessFile("encrypt", "decrypt", "dlog");

}

void SetPlugboard() // сделайте соединения на плате штепсельной вилки

{

int i, n, x;

char p1, p2, ch;

string inLine;

// Прочитайте строку, содержащую пары букв, соответствующие парам штекеров,

// соединенных проводом. Предполагается, что длина линии будет четной.

inputFile >> inLine;

inLine[inLine.length() - 1] = '\0';

n = inLine.length();

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

p1 = inLine[i];

p2 = inLine[i + 1];

x = index(p1);

if ((ch = plugboard[x]) != p1) { // занят? -> разъединить

plugboard[index(ch)] = ch;

plugboard[x] = p1;

}

plugboard[x] = p2; // подключить

x = index(p2);

if ((ch = plugboard[x]) != p2) { // занят? -> разъединить

plugboard[index(ch)] = ch;

plugboard[x] = p1;

}

plugboard[x] = p1; //подключить

}

}

// Инициализация из файла 'esetup' (step 2)

void SetRotorsAndReflector()

{

int i, n, rotor, rotPos;

char ch, ringPos;

string inLine;

// Читать фактическое количество роторов "mRotors", рассчитаете

// количество шагов шифрования "mSteps" и чтение строк "mRotors",

// каждый из них содержит три символа, обозначающих:

//

// (a) rotor ID (1-8,b,g)

// (b) rotor position (1-mRotors), and

// (c) ring character (position)

inputFile >> inLine;

mRotors = ChrToInt(inLine[0]);

if (mRotors > 4)

mRotors = 4;

mSteps = (mRotors << 1) + 3;

for (i = 1; i <= mRotors; ++i) {

inputFile >> inLine;

ch = inLine[0];

if (isdigit((int)ch))

rotor = ChrToInt(ch);

else {

ch = tolower(ch);

rotor = ch == 'b' ? 9

: ch == 'g' ? 10 : 0;

}

rotPos = ChrToInt(inLine[1]);

ringPos = inLine[2];

Iwindow[rotPos] = window[rotPos] = ringPos;

PlaceRotor(rotPos, rotor);

}

// Прочитайте строку, содержащую обозначение отражателя (t,b,c,B,C)

inputFile >> inLine;

ch = inLine[0];

switch (ch) {

case 't': n = 0; break; case 'b': n = 1; break;

case 'c': n = 2; break; case 'B': n = 3; break;

case 'C': n = 4; break; default: n = 0; break;

}

reflector = REFLECTOR[n];

ReflType = i;

}

int OpenFiles(const char\* inFname,

const char\* encFname,

const char\* logFname)

{

inputFile.open(inFname, ios::in);

outputFile.open(encFname, ios::out);

logFile.open(logFname, ios::out);

return inputFile.is\_open() && outputFile.is\_open() && logFile.is\_open();

}

void CloseFiles()

{

inputFile.close();

outputFile.close();

logFile.close();

}

void ReportMachine()

{

logFile << "Plugboard mappings:\n";

logFile << ROTOR[0] << "\n";

logFile << plugboard << "\n";

logFile << "\nRotor wirings:\n";

logFile << "position rotor ring setting notch sequence\n";

for (int i = mRotors; i >= 1; --i)

logFile << i << RotNumber[i] << window[i] << RotNotch[i] << RotWiring[i];

logFile << "\nreflector" << ReflType << reflector << "\n";

logFile << "\nrotors:\n";

ShowRotors();

}

void ShowRotors()

{

int i, j, k;

string Rwiring;

for (i = mRotors; i >= 1; --i) {

logFile << i << ": ";

Rwiring = RotWiring[i];

k = RotPos[i];

for (j = 0; j < k; ++j)

logFile << Rwiring[j];

logFile << "->";

for (j = k; j < Nchars; ++j)

logFile << Rwiring[j];

logFile << "\n";

}

}

int index(char c)

{

// c in alphabet

int i = 0;

while ((i < Nchars) && (c != alphabet[i]))

++i;

return i;

}

int ChrToInt(char c)

{

// '0' <= c <= '9'

return (int)(c - '0');

}

void PlaceRotor(int position, int r)

// комплект проводов одного Ротора

{

RotWiring[position] = ROTOR[r];

RotNotch[position] = NOTCH[r];

RotNumber[position] = r;

}

void SetRotorPositions()

{

int i, j, k;

string Rwiring;

char ch;

for (i = 1; i <= mRotors; ++i) {

j = RotNumber[i];

ch = window[j];

Rwiring = RotWiring[j];

k = 0;

while (Rwiring[k] != ch)

++k;

RotPos[j] = k;

}

}

int mod(int n, int modulus) // простая функция по модулю

{

while (n >= modulus)

n -= modulus;

while (n < 0)

n += modulus;

return n;

}

void ProcessPlainText()

{

int i, n;

char c1, c2;

string inLine;

logFile << "\n\nEncryption\n";

while (!inputFile.eof()) {

getline(inputFile, inLine, '\n');

n = inLine.length();

inLine[n] = '\0';

for (i = 0; i < n; ++i) {

c1 = inLine[i];

if (isupper((int)c1))

c1 = tolower(c1);

c2 = encrypt(c1);

ShowRotors();

ShowWindow();

logFile << " " << c1;

ShowSteps();

logFile << "\n";

outLine[i] = c2;

}

logFile << "\n";

outLine[i] = '\0';

outputFile << outLine << (!inputFile.eof() ? "\n" : "");

} cout << endl;

}

void ShowWindow()

{

int i;

for (i = mRotors; i >= 1; --i)

logFile << window[i] << " ";

logFile << " ";

}

void ShowSteps()

{

int i;

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

logFile << " -> " << step[i];

}

char encrypt(char c)

{

int i, r;

turn(); // перемещение роторов

i = 0; // пройти через:

step[i++] = plugboard[index(c)]; // коммутационная панель

for (r = 1; r <= mRotors; ++r)

step[i++] = RtoLpath(step[i - 1], r); // путь справа налево

step[i++] = reflector[index(step[i - 1])]; // отражатель

for (r = mRotors; r >= 1; --r) // путь слева направо

step[i++] = LtoRpath(step[i - 1], r);

step[i] = plugboard[index(step[i - 1])]; // коммутационная панель

return step[i];

}

void turn() // определите, какие роторы должны вращаться

{

int doit[Nrotors], n;

string r1 = RotWiring[1]; string r2 = RotWiring[2]; string r3;

if (mRotors > 3)

r3 = RotWiring[3];

// вычислите ширину шага для каждого ротора

doit[1] = 1;

for (int i = 2; i <= mRotors; ++i)

doit[i] = 0;

if ((RotNotch[1] == r1[RotPos[1]])

||

(RotNotch[2] == r2[RotPos[2]])) // двойной шаг

doit[2] = 1;

if (RotNotch[2] == r2[RotPos[2]])

doit[3] = 1;

if (mRotors > 3) {

if (RotNotch[3] == r3[RotPos[3]])

doit[4] = 1;

}

// поворот роторов " одновременно"

for (n = 1; n <= mRotors; ++n)

TurnRot(n, doit[n]);

}

void TurnRot(int n, int width) // Поверните Ротор "n" "width" раз

{ // обертывание вокруг, если это необходимо

string r;

if (width > 0) {

RotPos[n] = mod(RotPos[n] + width, Nchars);

r = RotWiring[n];

window[n] = r[RotPos[n]];

}

}

// Трансформация по пути справа налево через роторы

char RtoLpath(char c, int r) // преобразование символа "с" Ротором "r"

{

int n, offset, idx, ret;

string CurRotor;

CurRotor = RotWiring[r];

n = index(c);

offset = index(CurRotor[RotPos[r]]);

idx = mod(n + offset, Nchars);

ret = mod(index(CurRotor[idx]) - offset, Nchars);

return alphabet[ret];

}

// Трансформация на пути слева направо через роторы

char LtoRpath(char c, int r) // преобразование символа "с" Ротором "r"

{

int n, m, offset, idx, newchar;

string CurRotor;

CurRotor = RotWiring[r];

n = index(c);

offset = index(CurRotor[RotPos[r]]);

newchar = alphabet[mod(n + offset, Nchars)];

m = 0;

while (m < Nchars && CurRotor[m] != newchar)

++m;

idx = mod(m - offset, Nchars);

return alphabet[idx];

}