Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт по лабораторной работе №1 по курсу «Защита информации»

Тема Шифровальная машина «Энигма»

Студент Ву Хай Данг.

Группа ИУ7И-72Б

Оценка (баллы)

Преподаватели Чиж И. С.

Москва — 2024 г.

Введение

Шифровальная машина «Энигма» — одна из самых известрых шифровальнных машин, использовавшихся для шифрования и расшифровывания секретных сообщений.

Целью данной работы является реализация в виде программы на языке программирования C аналога шифровальной машины «Энигма», обеспечеие шифрования и расшифровки файла.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1) изучить алгоритм работы шифровальной машины «Энигма»;
2) реализовать алгоритм работы шифровальной машины «Энигма» в виде программы;
3) протестировать разработанную программу;
4) описать и обосновать полученные результаты в отчёте о выполненной лабораторной работе.

1 Аналитическая часть

В этом разделе будут рассмотрены классический алгоритм работы шифровальной машины «Энигма», а также её вариант, использованный во время Второй мировой войны.

Шифровальная машина «Энигма» состоит из следующих деталей: роторы, входное колесо, рефлектор, а также коммутационная панель.

1.1 Роторы

«Энигма» предназначена для шифрации сообщений, написанных на английском языке. Ротор — прикреплённый к шестерёнке с 26 зубцами (по одному на каждую букву алфавита) элемент, предназначенный для преобразования одной буквы в другую.

В разное время в разных реализациях «Энигмы» использовалось разное количество роторов. Во время Второй мировой войны использовались 3 ротора, причём всего было 10 роторов, преобразовывающих буквы в соответствии с таблицей 1.1.

Таблица 1.1 – Преобразования роторов «Энигмы»

Ротор	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
I	E	K	M	F	L	G	D	Q	V	Z	N	T	O	W	Y	H	X	U	S	P	A	I	B	R	C	J
II	A	J	D	K	S	I	R	U	X	B	L	H	W	T	M	C	Q	G	Z	N	P	Y	F	V	O	E
III	B	D	F	H	J	L	C	P	R	T	X	V	Z	N	Y	E	I	W	G	A	K	M	U	S	Q	O
IV	E	S	O	V	P	Z	J	A	Y	Q	U	I	R	H	X	L	N	F	T	G	K	D	C	M	W	B
V	V	Z	B	R	G	I	T	Y	U	P	S	D	N	H	L	X	A	W	M	J	Q	O	F	E	C	K
VI	J	P	G	V	O	U	M	F	Y	Q	B	E	N	H	Z	R	D	K	A	S	X	L	I	C	T	W
VII	N	Z	J	H	G	R	C	X	M	Y	S	W	B	O	U	F	A	I	V	L	P	E	K	Q	D	T
VIII	F	K	Q	H	T	L	X	O	C	B	J	S	P	D	Z	R	A	M	E	W	N	I	U	Y	G	V
IX	L	E	Y	J	V	C	N	I	X	W	P	B	Q	M	D	R	T	A	K	Z	G	F	U	H	O	S
X	F	S	O	K	A	N	U	E	R	H	M	B	T	I	Y	C	W	L	Q	P	Z	X	V	G	J	D

1.2 Входное колесо

Входное колесо — элемент, позволяющий выставить роторы в необходимые значения. В физической машине было 3 отверстия, позволяющих про-

сматривать, в каком состоянии находится каждый ротор. Положения роторов является ключевым для процесса шифрования, поскольку в зависимости от них одно и то же сообщение будет зашифровано по-разному и будет требовать сответствующих начальных значений роторов для дешифрации.

1.3 Рефлектор

Рефлектор — элемент, попарно соединяющий контакты последнего ротора, тем самым направляя ток обратно на последний ротор. Так, после этого электрический сигнал пойдёт в обратном направлении, пройдя через все роторы повторно. Во время Второй мировой войны было создано 2 рефлектора, представленных в таблице

Таблица 1.2 – Преобразования роторов «Энигмы»

Рефлектор	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
I	F	V	P	J	I	A	O	Y	E	D	R	Z	X	W	G	C	T	K	U	Q	S	B	N	M	H	L
II	Y	R	U	H	Q	S	L	D	P	X	N	G	O	K	M	I	E	B	F	Z	C	W	V	J	A	T

1.4 Коммутационная панель

Коммутационная панель позволяет оператору шифровальной машины варьировать содержимое проводов, попарно соединяющих буквы английского алфавита. Эффект состоял в том, чтобы усложнить работу машины, не увеличивая число роторов. Так, если на коммутационной панели соединены буквы 'A' и 'Z', то каждая буква 'A', проходящая через коммутационную панель, будет заменена на 'Z' и наоборот. Сигналы попадали на коммутационную панель 2 раза: в начале и в конце обработки отдельного символа.

2 Конструкторская часть

В этом разделе будут представлены описания используемых типов данных, а также требования к программе.

В этом разделе представлена схема алгоритма шифровальной машины «Энигма».

2.1 Разработка алгоритмов

На рисунке 2.1 приведена схема работы шифровальной машины Энигма.

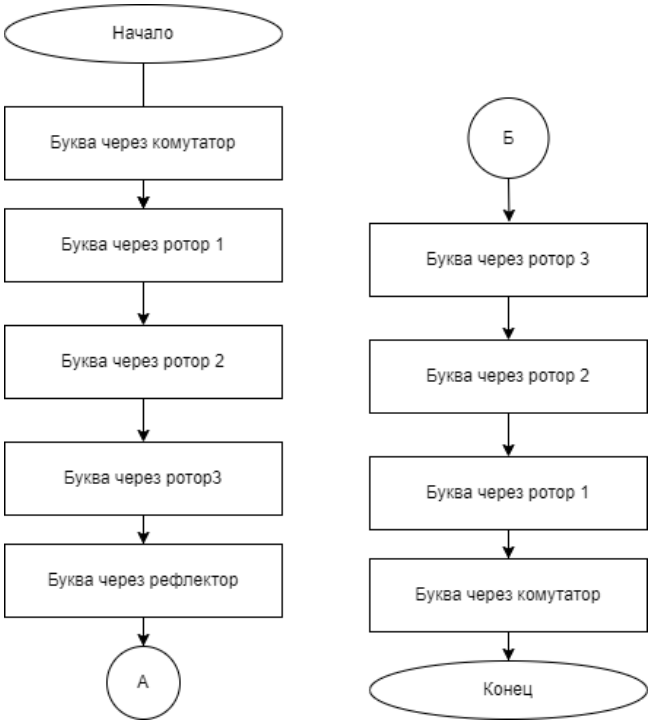

Рисунок 2.1 – Схема работы шифровальной машина Энигма

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги реализаций алгоритма шифрования машины «Энигма».

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования C. Данный язык удоволетворяет поставленным критериям по средствам реализации.

3.2 Реализация алгоритма

В листингах 3.1 представлена реализация алгоритма шифрования машины «Энигма».

Листинг 3.1 – Реализация алгоритма шифрования машины «Энигма»

```
1  #include <stdio.h>
2  #include <stdint.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  #define ALPHABET_SIZE 16
7
8  const int8_t ALPHABET[ALPHABET_SIZE] = {0, 1, 2, 3, 4, 5, 6, 7, 8,
       9, 10, 11, 12, 13, 14, 15};
9
10 void my_memcpy(int8_t *dst, const int8_t *src, int len)
11 {
12     for (int i = 0; i < len; i++)
13     {
14         dst[i] = src[i];
15     }
16 }
17
18 int8_t find(const int8_t* nums, int8_t num, int size) {
```

```
19      for (int i = 0; i < size; i++) {
20          if (num == nums[i])
21          return i;
22      }
23      return -1;
24  }
25
26  // Rotor struct and methods
27  typedef struct {
28      int8_t wiring[ALPHABET_SIZE];
29  } Rotor;
30
31  void initRotor(Rotor* rotor, const int8_t* wiring, int position) {
32      for (int i = position; i < ALPHABET_SIZE; i++) {
33          rotor->wiring[i - position] = wiring[i];
34      }
35      for (int i = 0; i < position; i++) {
36          rotor->wiring[ALPHABET_SIZE - position + i] = wiring[i];
37      }
38  }
39
40  int8_t rotorEncryptForward(Rotor* rotor, int8_t c) {
41      int index = find(rotor->wiring, c, ALPHABET_SIZE);
42      return ALPHABET[index];
43  }
44
45  int8_t rotorEncryptBackward(Rotor* rotor, int8_t c) {
46      int index = find(ALPHABET, c, ALPHABET_SIZE);
47      return rotor->wiring[index];
48  }
49
50  void rotorRotate(Rotor* rotor) {
51      int8_t tmp = rotor->wiring[0];
52      for (int i = 0; i < ALPHABET_SIZE - 1; i++) {
53          rotor->wiring[i] = rotor->wiring[i + 1];
54      }
55      rotor->wiring[ALPHABET_SIZE - 1] = tmp;
56  }
57
58  // Commutation Panel struct and methods
59  typedef struct {
```

```
60      int8_t wiring[ALPHABET_SIZE];
61      int8_t originalWiring[ALPHABET_SIZE];
62  } CommutationPanel;
63
64  void initCommutationPanel(CommutationPanel* panel, const int8_t*
        wiring) {
65      my_memcpy(panel->originalWiring, wiring, ALPHABET_SIZE);
66      my_memcpy(panel->wiring, wiring, ALPHABET_SIZE);
67      for (int i = 0; i < ALPHABET_SIZE; i += 2) {
68          int8_t temp = panel->wiring[i];
69          panel->wiring[i] = panel->wiring[i + 1];
70          panel->wiring[i + 1] = temp;
71      }
72  }
73
74  int8_t commutate(CommutationPanel* panel, int8_t c) {
75      int idx = find(panel->originalWiring, c, ALPHABET_SIZE);
76      if (idx != -1) {
77          return panel->wiring[idx];
78      }
79      return c; // No change if not found
80  }
81
82  // Reflector struct and methods
83  typedef struct {
84      int8_t wiring[ALPHABET_SIZE];
85  } Reflector;
86
87  void initReflector(Reflector* reflector, const int8_t* wiring) {
88      my_memcpy(reflector->wiring, wiring, ALPHABET_SIZE);
89      for (int i = 0; i < ALPHABET_SIZE; i += 2) {
90          int8_t temp = reflector->wiring[i];
91          reflector->wiring[i] = reflector->wiring[i + 1];
92          reflector->wiring[i + 1] = temp;
93      }
94  }
95
96  int8_t reflect(Reflector* reflector, int8_t c) {
97      return reflector->wiring[find(ALPHABET, c, ALPHABET_SIZE)];
98  }
99
```

```
100 // Enigma Machine structure
101 typedef struct {
102     Rotor* rotors;
103     int rotorCount;
104     Reflector reflector;
105     CommutationPanel panel;
106     int cnt;
107 } Enigma;
108 
109 void initEnigma(Enigma* enigma, Rotor* rotors, int rotorCount,
        Reflector reflector, CommutationPanel panel) {
110     enigma->rotors = rotors;
111     enigma->rotorCount = rotorCount;
112     enigma->reflector = reflector;
113     enigma->panel = panel;
114     enigma->cnt = 0;
115 }
116 
117 void rotateRotors(Enigma* enigma) {
118     enigma->cnt++;
119     rotorRotate(&enigma->rotors[0]);
120     if (enigma->cnt % ALPHABET_SIZE == 0) {
121         rotorRotate(&enigma->rotors[1]);
122     }
123     if (enigma->cnt % (ALPHABET_SIZE * ALPHABET_SIZE) == 0) {
124         rotorRotate(&enigma->rotors[2]);
125     }
126 }
127 
128 int8_t encryptChar(Enigma* enigma, int8_t p1, int8_t p2) {
129     // Process first part of the byte
130     p1 = commutate(&enigma->panel, p1);
131     for (int i = 0; i < enigma->rotorCount; i++) {
132         p1 = rotorEncryptForward(&enigma->rotors[i], p1);
133     }
134     p1 = reflect(&enigma->reflector, p1);
135     for (int i = enigma->rotorCount - 1; i >= 0; i--) {
136         p1 = rotorEncryptBackward(&enigma->rotors[i], p1);
137     }
138     p1 = commutate(&enigma->panel, p1);
139 
```

```
140     // Process second part of the byte
141     p2 = commutate(&enigma->panel, p2);
142     for (int i = 0; i < enigma->rotorCount; i++) {
143         p2 = rotorEncryptForward(&enigma->rotors[i], p2);
144     }
145     p2 = reflect(&enigma->reflector, p2);
146     for (int i = enigma->rotorCount - 1; i >= 0; i--) {
147         p2 = rotorEncryptBackward(&enigma->rotors[i], p2);
148     }
149     p2 = commutate(&enigma->panel, p2);
150 
151     // Combine both parts into one byte
152     return (p1 << 4) | p2;
153 }
154 
155 char* encryptMessage(Enigma* enigma, const char* message) {
156     int len = strlen(message);
157     char* encryptedMessage = malloc((len + 1) * sizeof(char));
158     for (int i = 0; i < len; i++) {
159         int8_t p1 = (message[i] & 0xf0) >> 4;
160         int8_t p2 = (message[i] & 0x0f);
161         encryptedMessage[i] = encryptChar(enigma, p1, p2);
162         rotateRotors(enigma);
163     }
164     encryptedMessage[len] = '\0';
165     return encryptedMessage;
166 }
167 
168 void encryptFile(Enigma* enigma, const char* inputFile, const char*
    outputFile) {
169     FILE* inFile = fopen(inputFile, "rb");
170     FILE* outFile = fopen(outputFile, "wb");
171 
172     if (!inFile || !outFile) {
173         printf("Cannot␣open␣files!\n");
174         return;
175     }
176 
177     int8_t buffer;
178     while (fread(&buffer, sizeof(int8_t), 1, inFile)) {
179         int8_t p1 = (buffer & 0xf0) >> 4;
```

```
180             int8_t p2 = buffer & 0x0f;
181             buffer = encryptChar(enigma, p1, p2);
182             rotateRotors(enigma);
183             fwrite(&buffer, sizeof(int8_t), 1, outFile);
184         }
185
186         fclose(inFile);
187         fclose(outFile);
188 }
```

3.3 Тестирование

Таблица 3.1 – Функциональные тесты

Входная строка	Выходная строка
Hello world —-	>—- Hello world

Вывод

Были представлены листинги реализаций алгоритма шифрования в машине «Энигма» согласно алгоритму, представленному в первой части, а также проведено тестирование разработанной программы.

Заключение

В результате лабораторной работы были изучены принципы работы шифровальной машины «Энигма», была реализована программа, способная шифровать и дешифровать текстовый файл, позволять настраивать роторы, рефлектор и коммутационную панель.

Были решены следующие задачи:

1) изучен алгоритм работы шифровальной машины «Энигма»;

2) реализован алгоритм работы шифровальной машины «Энигма» в виде программы, обеспечив возможности шифрования и расшифровки текстового файла;

3) полученная программа протестирована, произведена демонстрация того, что во всех случаях сообщение удаётся дешифровать и получить исходное;

4) полученные результаты описаны в отчёте о выполненной лабораторной работе.