

**Федеральное государственное автономное образовательное учреждение
высшего образования**

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. Тихонова

Департамент электронной инженерии

ОТЧЕТ

О ПРАКТИЧЕСКОЙ РАБОТЕ

по дисциплине «Системное программирование»

«Работа с памятью. Указатели, массивы, битовые поля, объединения»

Вариант 11

Студент гр. БИБ201

Шадрунов Алексей

Дата выполнения: 14 ноября 2022 г.

Преподаватель:

Морозов В. И.

«___» _____ 2022 г.

Москва, 2022

Содержание

1	Цели работы	3
2	Ход работы	3
2.1	Структура программы	3
2.1.1	main	3
2.1.2	str2char	4
2.1.3	print_help	4
2.1.4	AirportEntry	4
2.1.5	AirportUnion	5
2.1.6	xor_byte	5
2.1.7	process	5
2.2	Работа программы	5
2.2.1	-e	5
2.2.2	-d	6
2.2.3	-et	6
2.2.4	-d	6
2.3	Исключения	7
2.3.1	Пользовательский ввод	7
2.3.2	Нет доступа к файлу при чтении	7
2.3.3	Неправильные аргументы	8
3	Выводы о проделанной работе	9
	Приложение 1	10

1. Цели работы

Реализовать программу, выполняющую ввод, сериализацию в текстовый или бинарный файл, а также десериализацию и вывод в консоль пользователю информации об аэропорте.

2. Ход работы

В ходе работы я написал программу, реализующую необходимый функционал, на языке C.

2.1. Структура программы

2.1.1. main

Работа программы начинается с функции **main**. Функция принимает на вход аргументы из командной строки: флаги, выбирающие режим работы программы, и путь к файлу. Допустимые флаги:

- **-e** используется для включения режима кодирования;
- **-d** используется для включения режима декодирования;
- флаг **-t** позволяет использовать текстовый файл для хранения структуры.

Примеры запуска программы:

```
./hw1 -e file.bin # encrypt to file.bin
./hw1 -t file.bin # read from file.bin
./hw1 -et file.txt # use text file file.txt
./hw1 -et file.txt
```

Разберем последовательность работы.

- Сначала функция проверяет число переданных аргументов (должно быть три или четыре, иначе исключение).
- Далее функция определяет режим работы. Обязательно должно быть указано, кодирование или декодирование следует выполнять. Также определяется режим работы с файлом. Сохраняется путь к файлу.

- Если происходит кодирование, то далее программа вызывает функцию **get_data_from_user** для ввода данных от пользователя.
- Выделяется динамическая память для хранения преобразованных данных перед записью в файл.
- Вызывается функция **process**, которая копирует байты из структуры в буфер для преобразованных данных, применяя к каждому байту функцию **xor_byte**.
- Далее буфер записывается в файл. Если файл текстовый, то каждое значение байта заменяется соответствующим строковым представлением по таблице замен. Если же файл бинарный, запись производится непосредственно.
- Если выбран режим декодирования, то последовательность действий обратная: сначала создаётся пустая структура и буфер для ввода информации из файла. Затем из текстового файла считываются единицы и нули для образования байта, после чего конвертируются в **char**; из бинарного файла чтение происходит напрямую в буфер. Далее буфер подаётся на вход функции **process**, которая расшифровывает и записывает данные в структуру. Результат выводится на экран.
- В конце все ресурсы освобождаются.

2.1.2. str2char

Функция переводит строковую двоичную запись байта в **char**.

2.1.3. print_help

Печатает справку о том, как пользоваться программой (какие аргументы допустимы).

2.1.4. AirportEntry

Структура для хранения данных об аэропорте. Состоит из двух полей:

- **char name[128]** — имя аэропорта, максимальная длина 127 символов.
- **char letters[3]** — буквенный код аэропорта.

2.1.5. AirportUnion

Объединение для доступа к данным об аэропорте побайтно. Состоит из двух полей:

- **struct AirportEntry ent** — структура;
- **char arr[sizeof(struct AirportEntry) / sizeof(char)]** — байтовый массив, используется для байтового доступа к данным структуры.

2.1.6. xor_byte

Функция принимает на вход **char**, к которому применяет шифрующее или дешифрующее преобразование — побитовое исключающее ИЛИ с числом **01010101**.

2.1.7. process

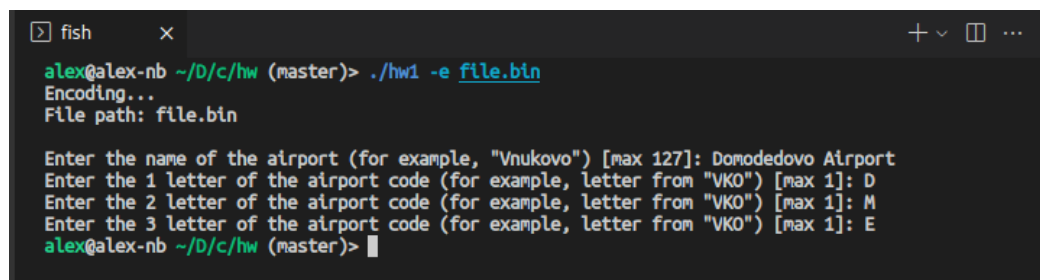
Функция принимает на вход указатели на два буфера (массива байт) – входной и выходной, их размеры, а также функцию обратного вызова и записывает результат применения функции к элементам входного буфера на соответствующие позиции выходного буфера.

2.2. Работа программы

Продемонстрируем различные режимы работы программы.

2.2.1. -e

Здесь мы ввели данные об аэропорте Домодедово и сохранили их в файл file.bin (Рисунок 1):



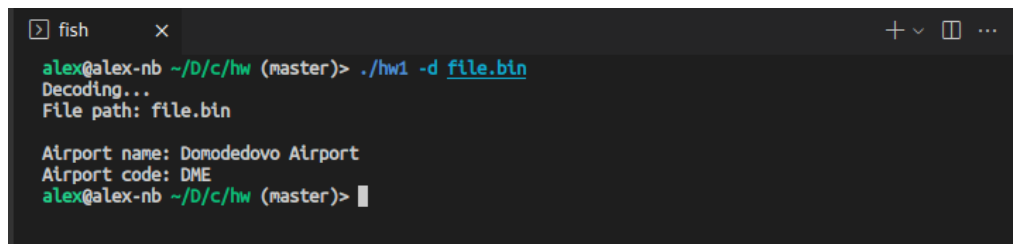
```
fish x
alex@galex-nb ~/D/c/hw (master)> ./hw1 -e file.bin
Encoding...
File path: file.bin

Enter the name of the airport (for example, "Vnukovo") [max 127]: Domodedovo Airport
Enter the 1 letter of the airport code (for example, letter from "VKO") [max 1]: D
Enter the 2 letter of the airport code (for example, letter from "VKO") [max 1]: M
Enter the 3 letter of the airport code (for example, letter from "VKO") [max 1]: E
alex@galex-nb ~/D/c/hw (master)>
```

Рисунок 1 – Работа в режиме бинарного шифрования

2.2.2. -d

Расшифруем данные об аэропорте Домодедово из файла file.bin (Рисунок 2):



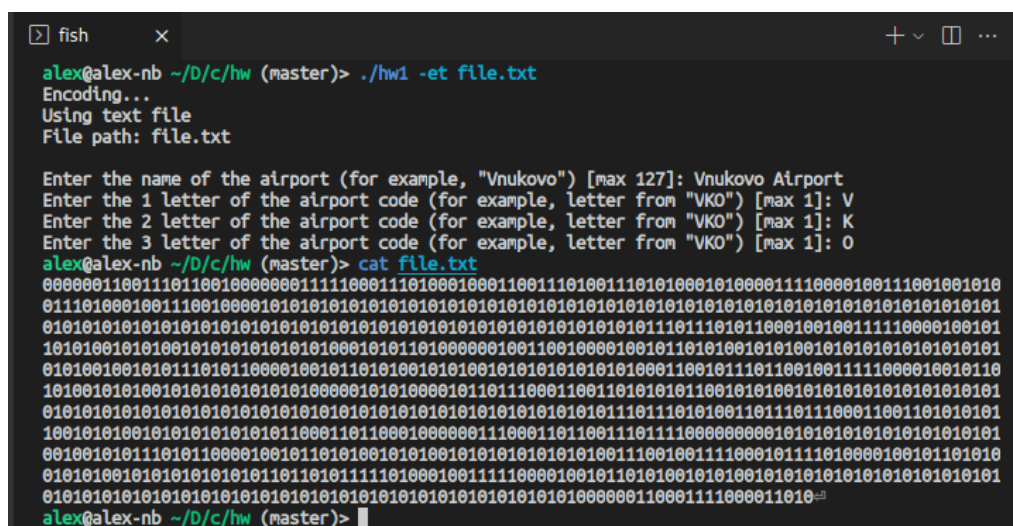
```
fish
alex@alex-nb ~/D/c/hw (master) > ./hw1 -d file.bin
Decoding...
File path: file.bin

Airport name: Domodedovo Airport
Airport code: DME
alex@alex-nb ~/D/c/hw (master) > █
```

Рисунок 2 – Работа в режиме бинарного расшифрования

2.2.3. -et

Зашифруем данные об аэропорте Внуково в текстовый файл file.txt. Содержимое файла показано на экране (Рисунок 3):



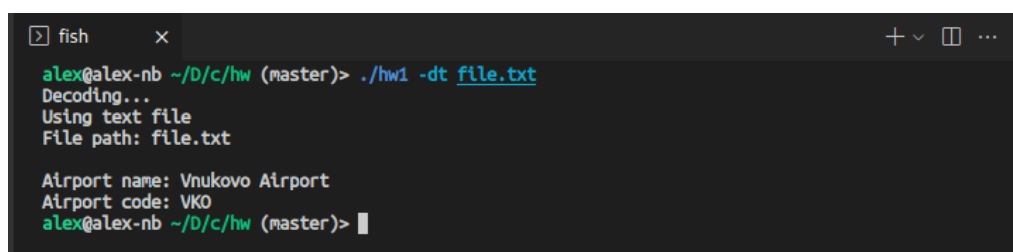
```
fish
alex@alex-nb ~/D/c/hw (master) > ./hw1 -et file.txt
Encoding...
Using text file
File path: file.txt

Enter the name of the airport (for example, "Vnukovo") [max 127]: Vnukovo Airport
Enter the 1 letter of the airport code (for example, letter from "VKO") [max 1]: V
Enter the 2 letter of the airport code (for example, letter from "VKO") [max 1]: K
Enter the 3 letter of the airport code (for example, letter from "VKO") [max 1]: O
alex@alex-nb ~/D/c/hw (master) > cat file.txt
0000001100111011001000000011111000111010001000110011101001110101000101000011110000100111001001010
01110100010011100100001010101010101010101010101010101010101010101010101010101010101010101010101
01010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010
10101001010100010101010101010101010101010101010101010101010101010101010101010101010101010101010
01010010010101110101100001001010101001010101010101010101010101010101010101010101010101010101010
10100101010010101010101010000010101000010110111000110011010101010010100101010101010101010101010
01010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010
10010101001010101010101010100011011000100000011100011011001110111000000000101010101010101010101
00100101011101011000010010110101001010100101010101010101010101010101010101010101010101010101010
01010100101010101010101011011010111101000100111100001001011010100101010010101010101010101010101
01010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010
alex@alex-nb ~/D/c/hw (master) > █
```

Рисунок 3 – Работа в режиме текстового зашифрования

2.2.4. -dt

Расшифруем данные из текстового файла file.txt (Рисунок 4):



```
fish
alex@alex-nb ~/D/c/hw (master) > ./hw1 -dt file.txt
Decoding...
Using text file
File path: file.txt

Airport name: Vnukovo Airport
Airport code: VKO
alex@alex-nb ~/D/c/hw (master) > █
```

Рисунок 4 – Работа в режиме текстового расшифрования

2.3. Исключения

2.3.1. Пользовательский ввод

Пользовательский ввод реализован надёжно: подающаяся на вход последовательность считывается в динамическую память до первого символа перевода строки/конца файла, после чего символы, выходящие за границы диапазона структуры, обрезаются. Далее считываются три символа для использования в качестве кода аэропорта. На рисунке 5 показана работа в случае длинной последовательности:



```
fish
alex@alex-nb ~/D/c/hw (master)> ./hw1 -e file.bin
Encoding...
File path: file.bin

Enter the name of the airport (for example, "Vnukovo") [max 127]: Very Very Long Airport Name in
some exotic country Very Very Long Airport Name in some exotic country Very Very Long Airport Nam
e in some exotic country Very Very Long Airport Name in some exotic country Very Very Long Airpor
t Name in some exotic country
Enter the 1 letter of the airport code (for example, letter from "VKO") [max 1]: Very Very Long A
irport Name in some exotic country Very Very Long Airport Name in some exotic country Very Very L
ong Airport Name in some exotic country Very Very Long Airport Name in some exotic country
Enter the 2 letter of the airport code (for example, letter from "VKO") [max 1]: Enter the 3 lett
er of the airport code (for example, letter from "VKO") [max 1]:
alex@alex-nb ~/D/c/hw (master)> ./hw1 -d file.bin
Decoding...
File path: file.bin

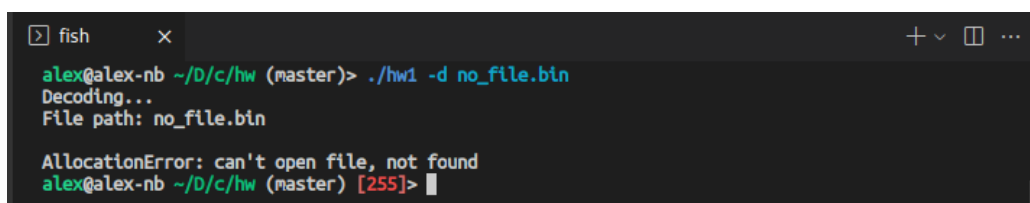
Airport name: Very Very Long Airport Name in some exotic country Very Very Long Airport Name in s
ome exotic country Very Very Long Airport Na
Airport code: Ver
alex@alex-nb ~/D/c/hw (master)>
```

Рисунок 5 – Длинная входная последовательность

Таким образом, любой ввод корректно обрабатывается.

2.3.2. Нет доступа к файлу при чтении

Проверим, что будет, если заставить программу произвести расшифрование несуществующего файла (Рисунок 6):



```
fish
alex@alex-nb ~/D/c/hw (master)> ./hw1 -d no_file.bin
Decoding...
File path: no_file.bin

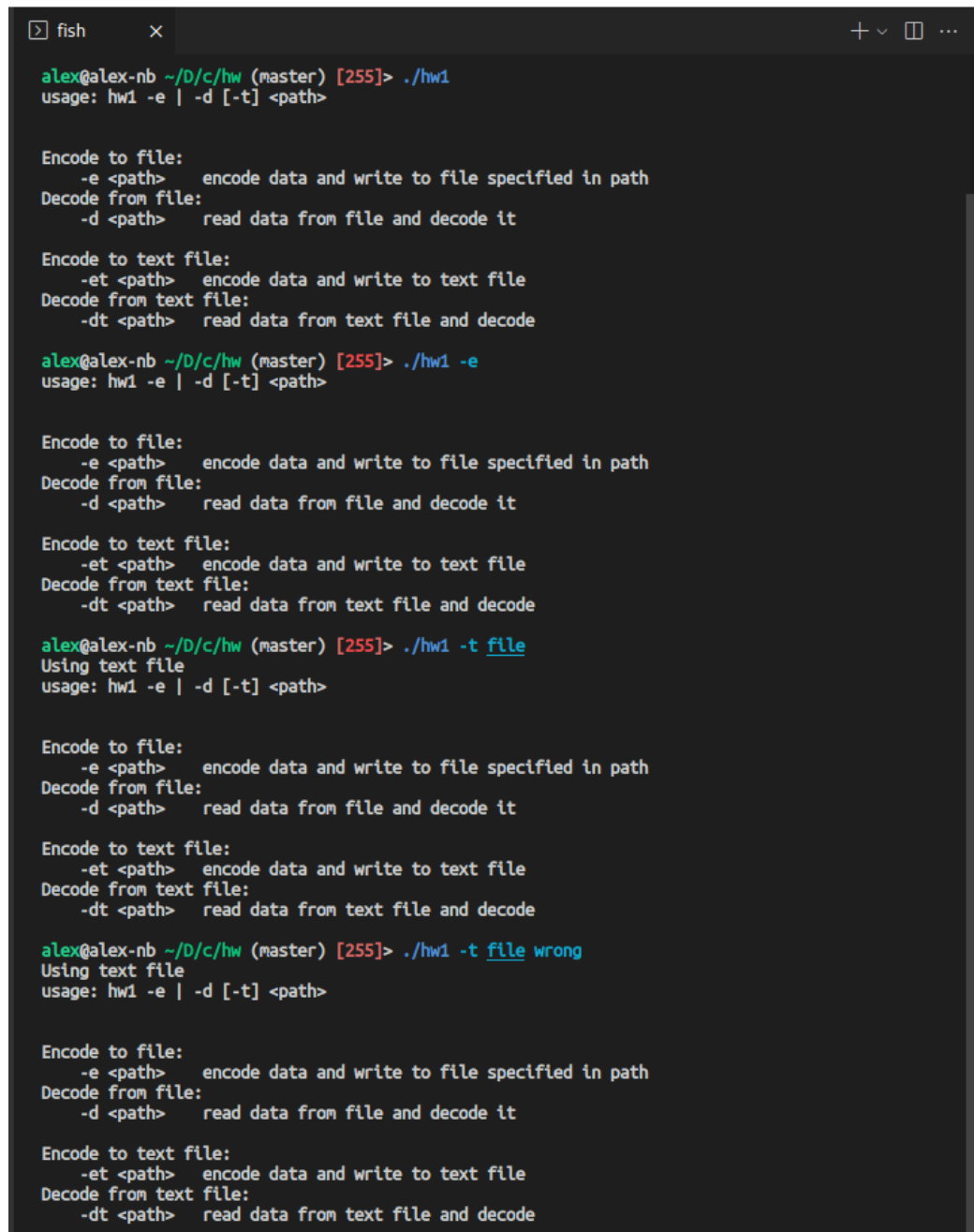
AllocationError: can't open file, not found
alex@alex-nb ~/D/c/hw (master) [255]>
```

Рисунок 6 – Нет файла

Перед открытием на чтение проверяется, что файл существует, в противном случае пользователю выводится сообщение об ошибке.

2.3.3. Неправильные аргументы

Если нет нужных аргументов, программа не будет работать (Рисунок 7):



```
fish
alex@galex-nb ~/D/c/hw (master) [255]> ./hw1
usage: hw1 -e | -d [-t] <path>

Encode to file:
  -e <path>  encode data and write to file specified in path
Decode from file:
  -d <path>  read data from file and decode it

Encode to text file:
  -et <path> encode data and write to text file
Decode from text file:
  -dt <path> read data from text file and decode

alex@galex-nb ~/D/c/hw (master) [255]> ./hw1 -e
usage: hw1 -e | -d [-t] <path>

Encode to file:
  -e <path>  encode data and write to file specified in path
Decode from file:
  -d <path>  read data from file and decode it

Encode to text file:
  -et <path> encode data and write to text file
Decode from text file:
  -dt <path> read data from text file and decode

alex@galex-nb ~/D/c/hw (master) [255]> ./hw1 -t file
Using text file
usage: hw1 -e | -d [-t] <path>

Encode to file:
  -e <path>  encode data and write to file specified in path
Decode from file:
  -d <path>  read data from file and decode it

Encode to text file:
  -et <path> encode data and write to text file
Decode from text file:
  -dt <path> read data from text file and decode

alex@galex-nb ~/D/c/hw (master) [255]> ./hw1 -t file wrong
Using text file
usage: hw1 -e | -d [-t] <path>

Encode to file:
  -e <path>  encode data and write to file specified in path
Decode from file:
  -d <path>  read data from file and decode it

Encode to text file:
  -et <path> encode data and write to text file
Decode from text file:
  -dt <path> read data from text file and decode
```

Рисунок 7 – Неправильные аргументы

Перед открытием на чтение проверяется, что файл существует, в противном случае пользователю выводится сообщение об ошибке.

Также в программе есть обработка ещё некоторых исключений, но они не возникают.

3. Выводы о проделанной работе

В ходе работы я реализовал программу, выполняющую ввод, сериализацию в текстовый или бинарный файл, а также десериализацию и вывод в консоль пользователю информации об аэропорте. Для переключения режимов работы используются аргументы командной строки.

Приложение 1

```
1 // HW 1, Shadrinov Aleksei
2
3 /* Название аэропорта и список из трёх символьных значений
4  - его обозначения на латинице например(, SVOили
5  DME). Шифрование
6  : к каждому байту применяется операция побитового
7  исключающего ИЛИ с двоичным числом
8  01010101. Расшифровывание: аналогично шифрованию
9  .
10
11 how to run:
12 ./hw1 -e file.bin
13 ./hw1 -d file.bin
14
15 */
16
17 #include <stdio.h>
18 #include <stdlib.h>
19 #include <stdbool.h>
20 #include <unistd.h>
21
22 /**
23  * LUT to convert unsigned char to string
24  */
25 static char *encoding_bytes[] =
26     {"00000000", "00000001", "00000010", "00000011", "00000100", "00000101",
27     "00000110", "00000111",
28     "00001000", "00001001", "00001010", "00001011", "00001100", "00001101",
29     "00001110", "00001111",
30     "00010000", "00010001", "00010010", "00010011", "00010100", "00010101",
31     "00010110", "00010111",
32     "00011000", "00011001", "00011010", "00011011", "00011100", "00011101",
33     "00011110", "00011111",
34     "00100000", "00100001", "00100010", "00100011", "00100100", "00100101",
35     "00100110", "00100111",
36     "00101000", "00101001", "00101010", "00101011", "00101100", "00101101",
37     "00101110", "00101111",
38     "00110000", "00110001", "00110010", "00110011", "00110100", "00110101",
39     "00110110", "00110111",
40     "00111000", "00111001", "00111010", "00111011", "00111100", "00111101",
41     "00111110", "00111111",
42     "01000000", "01000001", "01000010", "01000011", "01000100", "01000101",
43     "01000110", "01000111",
44     "01001000", "01001001", "01001010", "01001011", "01001100", "01001101",
45     "01001110", "01001111",
46     "01010000", "01010001", "01010010", "01010011", "01010100", "01010101",
47     "01010110", "01010111",
48     "01011000", "01011001", "01011010", "01011011", "01011100", "01011101",
49     "01011110", "01011111",
50     "01100000", "01100001", "01100010", "01100011", "01100100", "01100101",
51     "01100110", "01100111",
52     "01101000", "01101001", "01101010", "01101011", "01101100", "01101101",
53     "01101110", "01101111",
54     "01110000", "01110001", "01110010", "01110011", "01110100", "01110101",
55     "01110110", "01110111",
56     "01111000", "01111001", "01111010", "01111011", "01111100", "01111101",
57     "01111110", "01111111",
58     "10000000", "10000001", "10000010", "10000011", "10000100", "10000101",
59     "10000110", "10000111",
60     "10001000", "10001001", "10001010", "10001011", "10001100", "10001101",
61     "10001110", "10001111",
62     "10010000", "10010001", "10010010", "10010011", "10010100", "10010101",
63     "10010110", "10010111",
64     "10011000", "10011001", "10011010", "10011011", "10011100", "10011101",
65     "10011110", "10011111",
66     "10100000", "10100001", "10100010", "10100011", "10100100", "10100101",
67     "10100110", "10100111",
68     "10101000", "10101001", "10101010", "10101011", "10101100", "10101101",
69     "10101110", "10101111",
70     "10110000", "10110001", "10110010", "10110011", "10110100", "10110101",
71     "10110110", "10110111",
72     "10111000", "10111001", "10111010", "10111011", "10111100", "10111101",
73     "10111110", "10111111",
74     "11000000", "11000001", "11000010", "11000011", "11000100", "11000101",
75     "11000110", "11000111",
76     "11001000", "11001001", "11001010", "11001011", "11001100", "11001101",
77     "11001110", "11001111",
78     "11010000", "11010001", "11010010", "11010011", "11010100", "11010101",
79     "11010110", "11010111",
80     "11011000", "11011001", "11011010", "11011011", "11011100", "11011101",
81     "11011110", "11011111",
82     "11100000", "11100001", "11100010", "11100011", "11100100", "11100101",
83     "11100110", "11100111",
84     "11101000", "11101001", "11101010", "11101011", "11101100", "11101101",
85     "11101110", "11101111",
86     "11110000", "11110001", "11110010", "11110011", "11110100", "11110101",
87     "11110110", "11110111",
88     "11111000", "11111001", "11111010", "11111011", "11111100", "11111101",
89     "11111110", "11111111"};
```

33 "00111000", "00111001", "00111010", "00111011", "00111100", "00111101",
 "00111110", "00111111",
 34 "01000000", "01000001", "01000010", "01000011", "01000100", "01000101",
 "01000110", "01000111",
 35 "01001000", "01001001", "01001010", "01001011", "01001100", "01001101",
 "01001110", "01001111",
 36 "01010000", "01010001", "01010010", "01010011", "01010100", "01010101",
 "01010110", "01010111",
 37 "01011000", "01011001", "01011010", "01011011", "01011100", "01011101",
 "01011110", "01011111",
 38 "01100000", "01100001", "01100010", "01100011", "01100100", "01100101",
 "01100110", "01100111",
 39 "01101000", "01101001", "01101010", "01101011", "01101100", "01101101",
 "01101110", "01101111",
 40 "01110000", "01110001", "01110010", "01110011", "01110100", "01110101",
 "01110110", "01110111",
 41 "01111000", "01111001", "01111010", "01111011", "01111100", "01111101",
 "01111110", "01111111",
 42 "10000000", "10000001", "10000010", "10000011", "10000100", "10000101",
 "10000110", "10000111",
 43 "10001000", "10001001", "10001010", "10001011", "10001100", "10001101",
 "10001110", "10001111",
 44 "10010000", "10010001", "10010010", "10010011", "10010100", "10010101",
 "10010110", "10010111",
 45 "10011000", "10011001", "10011010", "10011011", "10011100", "10011101",
 "10011110", "10011111",
 46 "10100000", "10100001", "10100010", "10100011", "10100100", "10100101",
 "10100110", "10100111",
 47 "10101000", "10101001", "10101010", "10101011", "10101100", "10101101",
 "10101110", "10101111",
 48 "10110000", "10110001", "10110010", "10110011", "10110100", "10110101",
 "10110110", "10110111",
 49 "10111000", "10111001", "10111010", "10111011", "10111100", "10111101",
 "10111110", "10111111",
 50 "11000000", "11000001", "11000010", "11000011", "11000100", "11000101",
 "11000110", "11000111",
 51 "11001000", "11001001", "11001010", "11001011", "11001100", "11001101",
 "11001110", "11001111",
 52 "11010000", "11010001", "11010010", "11010011", "11010100", "11010101",
 "11010110", "11010111",

```

53     "11011000", "11011001", "11011010", "11011011", "11011100", "11011101",
    "11011110", "11011111",
54     "11100000", "11100001", "11100010", "11100011", "11100100", "11100101",
    "11100110", "11100111",
55     "11101000", "11101001", "11101010", "11101011", "11101100", "11101101",
    "11101110", "11101111",
56     "11110000", "11110001", "11110010", "11110011", "11110100", "11110101",
    "11110110", "11110111",
57     "11111000", "11111001", "11111010", "11111011", "11111100", "11111101",
    "11111110", "11111111"};
58
59 /**
60  * Converts string with binary representation of char to char
61  */
62 char str2char(char *input)
63 {
64     char res = 0;
65     if (input[7] == '1') res += 1;
66     if (input[6] == '1') res += 2;
67     if (input[5] == '1') res += 4;
68     if (input[4] == '1') res += 8;
69     if (input[3] == '1') res += 16;
70     if (input[2] == '1') res += 32;
71     if (input[1] == '1') res += 64;
72     if (input[0] == '1') res += 128;
73     return res;
74 }
75
76 /**
77  * Prints help message to console
78  */
79 void print_help()
80 {
81     puts("usage: hw1 -e | -d [-t] <path> \n\n");
82     puts("Encode to file:");
83     puts("    -e <path> \t encode data and write to file specified in path");
84     puts("Decode from file:");
85     puts("    -d <path> \t read data from file and decode it \n");
86     puts("Encode to text file:");
87     puts("    -et <path> \t encode data and write to text file");

```

```

88     puts("Decode from text file:");
89     puts("    -dt <path> \t read data from text file and decode \n");
90 }
91
92 /**
93  * Struct to store data about Airport
94  */
95 struct AirportEntry
96 {
97     char name[128];
98     char letters[3];
99 };
100
101 /**
102  * Union to serialise data from structure
103  */
104 union AirportUnion
105 {
106     struct AirportEntry ent;
107     char arr[sizeof(struct AirportEntry) / sizeof(char)];
108 };
109
110 /**
111  * XOR given byte with 01010101
112  */
113 char xor_byte(char input)
114 {
115     if (sizeof(input) != 1)
116     {
117         puts("ValueError: input must be 1 byte long.");
118         exit(-1);
119     }
120     return input ^ 0b01010101;
121     // return input;
122 }
123
124 /**
125  * Copy bytes from input array to output array and apply operation function
126  * to each byte
127  */

```

```

127 int process(char *input, char *output, size_t input_s, size_t output_s,
    char(operation) (char))
128 {
129     if (input_s > output_s)
130         return 1; // Indexerror
131
132     for (int i = 0; i < input_s; i++)
133         output[i] = xor_byte(input[i]); // copy byte after xor_byte
134
135     return 0;
136 }
137
138 /**
139  * Reads data from keyboard, returns union AirportUnion
140  * user input safety: gets line of any length, truncates after 127 characters
141  */
142 union AirportUnion get_data_from_user()
143 {
144     union AirportUnion un;
145     printf("Enter the name of the airport (for example, \"Vnukovo\") [max
    127]: ");
146
147     // get full line
148     size_t input_size = sizeof(un.ent.name);
149     char *init_char = malloc(input_size * sizeof(char));
150     input_size = getline(&init_char, &input_size, stdin);
151
152     // truncate the rest of input if any
153     if (input_size > sizeof(un.ent.name))
154         input_size = sizeof(un.ent.name);
155     for (int i = 0; i < input_size - 1; i++)
156         un.ent.name[i] = init_char[i];
157     un.ent.name[input_size - 1] = '\\0';
158     free(init_char);
159
160     // input three letters
161     for (int i = 0; i < 3; i++)
162     {
163         printf("Enter the %d letter of the airport code (for example, letter
    from \"VKO\") [max 1]: ", i + 1);

```

```

164     scanf(" %c", &un.ent.letters[i]);
165 }
166
167     return un;
168 }
169
170 int main(int argc, char **argv)
171 {
172     // check number of arguments
173     if ((argc != 3) && (argc != 4))
174     {
175         print_help();
176         return -1;
177     }
178
179     // get mode
180     bool encode;
181     bool set_encode = false;
182     bool text = false;
183     int option;
184     while ((option = getopt(argc, argv, "det")) != -1)
185     {
186         switch (option)
187         {
188             case 'd':
189                 puts("Decoding...");
190                 encode = false;
191                 set_encode = true;
192                 break;
193             case 'e':
194                 puts("Encoding...");
195                 encode = true;
196                 set_encode = true;
197                 break;
198             case 't':
199                 puts("Using text file");
200                 text = true;
201                 break;
202             default:
203                 print_help();

```

```

204         return -1;
205     }
206 }
207
208 // e or d must be provided
209 if (!set_encode)
210 {
211     print_help();
212     return -1;
213 }
214 // get file path
215 char *path = argv[argc - 1];
216 printf("File path: %s \n\n", path);
217
218 if (encode)
219 {
220     // get data from user
221     union AirportUnion un = get_data_from_user();
222
223     // output buffer
224     char *output = malloc(sizeof(un) * sizeof(char));
225
226     // write output to buffer
227     if (process(un.arr, output, sizeof(un), sizeof(un), xor_byte))
228     {
229         puts("Error occurred during the process. Indexerror: size of
input must not be greater than size of output");
230         free(output);
231         return -1;
232     }
233
234     // write buffer to file
235     if (text)
236     {
237         FILE *ft = fopen(path, "w+");
238         if (!ft) // error opening file
239         {
240             puts("AllocationError: can't open file");
241             free(output);
242             return -1;

```



```

243     }
244     // write every char as string from LUT
245     for (int i = 0; i < sizeof(un); i++)
246         fputs(encoding_bytes[(unsigned char)output[i]], ft);
247
248     fclose(ft);
249 }
250 else
251 {
252     FILE *fp = fopen(path, "wb");
253     if (!fp) // error opening file
254     {
255         puts("AllocationError: can't open file");
256         free(output);
257         return -1;
258     }
259     // write bytes directly
260     fwrite(output, sizeof(char), sizeof(un), fp);
261     fclose(fp);
262 }
263
264     free(output);
265 }
266 else // decode
267 {
268     // create empty structure
269     union AirportUnion un;
270     char *input = malloc(sizeof(un) * sizeof(char));
271
272     // get encrypted data from file
273     if (access(path, F_OK))
274     {
275         puts("AllocationError: can't open file, not found");
276         free(input);
277         return -1;
278     }
279
280     if (text)
281     {
282         FILE *ft = fopen(path, "r");

```

```

283         if (!ft) // error opening file
284         {
285             puts("AllocationError: can't open file");
286             free(input);
287             return -1;
288         }
289         // read file by chunks of 8 bits
290         size_t chunksize = 8;
291         char buffer[chunksize + 1];
292
293         for (int i = 0; i < sizeof(un); i++)
294         {
295             fseek(ft, SEEK_SET, i * chunksize);           // find position
296             fread(&buffer, sizeof(char), chunksize, ft); // read 8
297             buffer[chunksize] = '\0';                     // terminator
298             // printf("%s \n", buffer);
299             // printf("%u ", (unsigned char)str2char(buffer));
300             input[i] = str2char(buffer); // convert string in buffer to
301             char
302         }
303         fclose(ft);
304     }
305     else
306     {
307         FILE *fp = fopen(path, "rb");
308         if (!fp) // error opening file
309         {
310             puts("AllocationError: can't open file");
311             free(input);
312             return -1;
313         }
314         fread(input, sizeof(char), sizeof(un), fp);
315         fclose(fp);
316     }
317     // decrypt data
318     if (process(input, un.arr, sizeof(un), sizeof(un), xor_byte))
319     {

```

```

319         puts("Error occurred during the process. Indexerror: size of
input must not be greater than size of output");
320         free(input);
321         return -1;
322     }
323
324     printf("Airport name: %s \n", un.ent.name);
325     printf("Airport code: %s \n", un.ent.letters);
326
327     free(input);
328 }
329
330 return 0;
331 }

```