

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра вычислительных систем

ОТЧЕТ
по практической работе 3
по дисциплине «Программирование»

Выполнил:
студент гр. ИВ-222
«16» мая 2023 г.

Очнев А.Д.

Проверил:
Старший преподаватель кафедры
вычислительных систем
«17» мая 2023 г.

Фульман В.О.

Оценка «_____»

Новосибирск 2023

ОГЛАВЛЕНИЕ

ЗАДАНИЕ	3
ВЫПОЛНЕНИЕ РАБОТЫ	6
ПРИЛОЖЕНИЕ	15

ЗАДАНИЕ

Задание 1.

Разработайте приложение, которое генерирует 1000000 случайных чисел и записывает их в два бинарных файла. В файл `uncompressed.dat` запишите числа в несжатом формате, в файл `compressed.dat` — в формате `varint`. Сравните размеры файлов.

Реализуйте чтение чисел из двух файлов. Добавьте проверку: последовательности чисел из двух файлов должны совпадать.

Задание 2.

Разработать приложение для кодирования и декодирования чисел в формате UTF-8.

Файлы `command.h` и `command.c` :

```
int encode_file(const char *in_file_name, const char *out_file_name)
```

Кодирование текстового файла в бинарный, реализация команды `encode`.

Параметры:

- `char *in_file_name (const)` –
путь ко входному текстовому файлу
- `char *out_file_name (const)` –
путь к выходному бинарному файлу

Результат: 0 в случае успеха, -1 в случае ошибки

```
int decode_file(const char *in_file_name, const char *out_file_name)
```

Декодирование бинарного файла, реализация команды `decode`.

Параметры:

- `char *in_file_name (const)` –
путь ко входному бинарному файлу
- `char *out_file_name (const)` –
путь к выходному текстовому файлу

Результат: 0 в случае успеха, -1 в случае ошибки

Файлы `coder.h` и `coder.c` :

```
int encode(uint32_t code_point, CodeUnits *code_units)
```

Параметры:

- `code_point` – число, которое необходимо закодировать
- `code_unit` – выходной параметр, результат кодирования

Результат: 0, если кодирование успешно, -1 иначе

```
uint32_t decode(const CodeUnit *code_unit)
```

Допущение: `code_unit` - корректный код, полученный с помощью функции `read_next_code_unit`.

Параметры: `code_unit` – закодированное представление числа

Return `code_point`:

результат декодирования

```
int read_next_code_unit(FILE *in, CodeUnits *code_units)
```

Считывает последовательность `code_units` из потока `in`. Implementation note: если считываемый `code_unit` битый, то функция пропускает байты до тех пор, пока не найдет корректный лидирующий байт.

Результат: 0 в случае успеха, -1 в случае ошибки или EOF

```
int write_code_unit(FILE *out, const CodeUnit *code_unit)
```

ВЫПОЛНЕНИЕ РАБОТЫ

Задание 1.

Переносим данные нам функции `encode_varint`, `decode_varint` и `generate_number`.

```
#include <assert.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#define N 1000000

size_t encode_varint(uint32_t value, uint8_t *buf) {
    assert(buf != NULL);
    uint8_t *cur = buf;
    while (value >= 0x80) {
        const uint8_t byte = (value & 0x7f) | 0x80;
        *cur = byte;
        value >>= 7;
        ++cur;
    }
    *cur = value;
    ++cur;
    return cur - buf;
}

uint32_t decode_varint(const uint8_t **bufp) {
    const uint8_t *cur = *bufp;
    uint8_t byte = *cur++;
    uint32_t value = byte & 0x7f;
    size_t shift = 7;
    while (byte >= 0x80) {
        byte = *cur++;
        value += (byte & 0x7f) << shift;
        shift += 7;
    }
    *bufp = cur;
    return value;
}

uint32_t generate_number() {
    const int r = rand();
    const int p = r % 100;
    if (p < 90) {
        return r % 128;
    }
    if (p < 95) {
        return r % 16384;
    }
    if (p < 99) {
        return r % 2097152;
    }
    return r % 268435455;
}
```

В функции `main` открываем файлы с ключами `wb` для работы с файлами в бинарном формате. Проверяем существуют ли файлы. Создаём массив с незакодированными элементами и заполняем его псевдослучайными числами. Записываем эти числа функцией

fwrite в файл uncompressed.dat. Создаём массивы buf, для хранения одного закодированного числа и массив compressed, для хранения всех закодированных чисел. Создаём указатель на массив compressed. В size сохраняем количество байт, которое занимает число после кодирования. Во вложенном цикле побайтово записываем в массив compressed закодированное число. Записываем все закодированные числа в файл compressed.dat. Закрываем файлы, освобождаем память, выделенную под массивы, после чего вызываем функцию для проверки на корректность кодирования.

```
int main() {
    char *unc = "uncompressed.dat";
    FILE *uncomp = fopen(unc, "wb");
    char *c = "compressed.dat";
    FILE *comp = fopen(c, "wb");
    if (!uncomp || !comp) {
        printf("Файл не найден\n");
        return -1;
    }
    uint32_t *uncompressed = malloc(N * sizeof(uint32_t));
    for (int i = 0; i < N; i++) {
        uncompressed[i] = generate_number();
    }
    fwrite(uncompressed, sizeof(uint32_t), N, uncomp);
    uint8_t buf[4];
    uint8_t *compressed = malloc(N * 4);
    uint8_t *point = compressed;
    size_t size;
    for (int i = 0; i < N; i++) {
        size = encode_varint(uncompressed[i], buf);
        for (int j = 0; j < size; j++) {
            *compressed = buf[j];
            compressed++;
        }
    }
    fwrite(point, sizeof(uint8_t), compressed - point, comp);
    free(point);
    free(uncompressed);
    fclose(comp);
    fclose(uncomp);
    check(c, unc);
    return 0;
}
```

В функции check открываем файлы с ключами rb для чтения файлов в бинарном формате. Проверяем существуют ли файлы. Перемещаем каретку в файле compressed с помощью функции fseek и в переменную size записываем размер файла, после чего перемещаем каретку обратно. Создаём массивы comprCheck и uncompCheck, заполняем их сжатыми и несжатыми числами с помощью функции fread. Создаём указатель на массив comprCheck. В цикле декодируем числа и сравниваем их со значениями массива с незакодированными числами, если совпадает, то инкрементируем счетчик. Закрываем файлы, освобождаем память, выделенную под массивы. Если изначальное количество чисел совпадает со счётчиком, то возвращаем 0.

```

int check(char *c, char *unc) {
    FILE *uncomp = fopen(unc, "rb");
    FILE *comp = fopen(c, "rb");
    if (!uncomp || !comp) {
        printf("Файл не найден\n");
        return -1;
    }
    fseek(comp, 0, SEEK_END);
    int size = ftell(comp);
    fseek(comp, 0, SEEK_SET);
    uint8_t *compCheck = malloc(size);
    fread(compCheck, 1, size, comp);
    uint32_t *uncompCheck = malloc(N * 4);
    fread(uncompCheck, 4, N, uncomp);
    const uint8_t *point = compCheck;
    int count = 0;
    for (int i = 0; i < N; i++) {
        if (decode_varint(&point) == uncompCheck[i]) {
            count++;
        }
    }
    fclose(comp);
    fclose(uncomp);
    free(uncompCheck);
    free(compCheck);
    if (count == N) {
        printf("Кодирование успешно\n");
        return 0;
    }
    return -1;
}

```

С помощью команды stat проверяем размеры файлов и определяем коэффициент сжатия, он равен 3,45.

```

Файл: compressed.dat
Размер: 1159207      Блоков: 2272      Блок В/В: 4096      обычный файл
Устройство: 0/36      Инода: 218669      Ссылки: 1
Доступ: (0644/-rw-r--r--)  Uid: ( 1000/   artem)  Gid: ( 1000/   artem)
Контекст: unconfined_u:object_r:user_home_t:s0
Доступ:      2023-05-16 11:01:01.889897947 -0400
Модифицирован: 2023-05-16 11:01:01.869897130 -0400
Изменён:      2023-05-16 11:01:01.869897130 -0400
Создан:       2023-05-15 07:16:42.119391198 -0400
[artem@fedora ex_first]$ stat uncompressed.dat
Файл: uncompressed.dat
Размер: 4000000      Блоков: 7816      Блок В/В: 4096      обычный файл
Устройство: 0/36      Инода: 218670      Ссылки: 1
Доступ: (0644/-rw-r--r--)  Uid: ( 1000/   artem)  Gid: ( 1000/   artem)
Контекст: unconfined_u:object_r:user_home_t:s0
Доступ:      2023-05-16 11:01:01.894898151 -0400
Модифицирован: 2023-05-16 11:01:01.870897170 -0400
Изменён:      2023-05-16 11:01:01.870897170 -0400
Создан:       2023-05-15 07:16:47.397606706 -0400

```

Задание 2.

Создаём заголовочные файлы `coder.h` и `command.h`, где объявляем прототипы функций. В `coder.h` также создаём прототип структуры `CodeUnits`, где поле `code` хранит в себе число записанное побайтово, а поле `length` длину числа.

```
#pragma once

enum {
    MaxCodeLength = 4
};

typedef struct {
    uint8_t code[MaxCodeLength];
    size_t length;
} CodeUnits;

int encode(uint32_t code_point, CodeUnits *code_units);
uint32_t decode(const CodeUnits *code_unit);
int read_next_code_unit(FILE *in, CodeUnits *code_units);
int write_code_unit(FILE *out, const CodeUnits *code_unit);

#pragma once

int encode_file(const char *in_file_name, const char *out_file_name);
int decode_file(const char *in_file_name, const char *out_file_name);
```

В файле `main` получаем четыре аргумента при вызове программы, где первый - команда кодировки или декодировки, второй - файл, из которого мы считываем, третий - файл, в который записываем. Проверяем корректность аргументов и в зависимости от команды вызываем функции `encode_file` или `decode_file`.


```

#include <stdio.h>
#include <string.h>

#include "coder.h"
#include "command.h"

int main(int argc, char *argv[]) {
    if (argc != 4) {
        printf("Неверное количество аргументов\n");
        return -1;
    }
    if (strcmp(argv[1], "decode") == 0) {
        decode_file(argv[2], argv[3]);
    } else if (strcmp(argv[1], "encode") == 0) {
        encode_file(argv[2], argv[3]);
    } else {
        printf("Неверная команда, используйте encode/decode\n");
        return -1;
    }
    return 0;
}

```

В файле command.c реализованы функции кодирования и декодирования всего файла. В функции encode_file открываем файлы и проверяем, существуют ли они. В цикле, пока не закончится файл, считываем число в переменную hex. Если кодировка прошла успешно, то записываем его в файл функцией write_code_unit. Закрываем файлы.

```

int encode_file(const char *in_file_name, const char *out_file_name) {
    FILE *input = fopen(in_file_name, "r");
    FILE *output = fopen(out_file_name, "wb");
    if (!input || !output) {
        printf("Файл не найден\n");
        return -1;
    }

    uint32_t hex;
    CodeUnits codeunit;
    while (fscanf(input, "%x" SCNx32, &hex) == 1) {
        if (encode(hex, &codeunit) == 0) {
            write_code_unit(output, &codeunit);
        } else {
            return -1;
        }
    }
    fclose(input);
    fclose(output);
    printf("Успешно\n");
    return 0;
}

```

В функции decode_file открываем файлы и проверяем, существуют ли они. Перемещаем

каретку во входном файле с помощью функции `fseek` и в переменную `size` записываем размер файла, после чего перемещаем каретку обратно. Пока каретка не дойдёт до конца файла, читаем последовательность `codeunit` с помощью функции `read_next_code_unit`. Декодируем числа с помощью функции `decode` и записываем с помощью функции `fprintf` в файл. Закрываем файлы.

```
int decode_file(const char *in_file_name, const char *out_file_name) {
    FILE *input = fopen(in_file_name, "rb");
    FILE *output = fopen(out_file_name, "w");
    if (!input || !output) {
        printf("Файл не найден\n");
        return -1;
    }

    fseek(input, 0, SEEK_END);
    int size = ftell(input);
    fseek(input, 0, SEEK_SET);
    while (ftell(input) != size) {
        CodeUnits codeunit;
        if (read_next_code_unit(input, &codeunit) == 0) {
            fprintf(output, "%" PRIx32 "\n", decode(&codeunit));
        } else {
            continue;
        }
    }
    fclose(input);
    fclose(output);
    printf("Успешно\n");
    return 0;
}
```

В файле `coder.c` реализованы функции кодирования и декодирования одного числа, а также функции записи и чтения из файлов.

В функции `encode` рассматриваем 4 случая: если длина двоичного числа меньше 8, для его записи достаточно 1 байта; если меньше 12, то 2 байт; если меньше 17, то 3 байт; если меньше 22, то 4 байт. В поле структуры `length` записываем количество байт, которое число будет занимать после декодирования. В массив `code` в каждый элемент побайтово записываем число.

```

int encode(uint32_t code_point, CodeUnits *code_units) {
    if (code_point < 0x80) { //<8
        code_units->length = 1;
        code_units->code[0] = code_point;
    } else if (code_point < 0x800) { //<12
        code_units->length = 2;
        code_units->code[0] = 0xc0 | (code_point >> 6);
        code_units->code[1] = 0x80 | (code_point & 0x3f); //00111111
    } else if (code_point < 0x10000) { //<17
        code_units->length = 3;
        code_units->code[0] = 0xe0 | (code_point >> 12);
        code_units->code[1] = 0x80 | ((code_point >> 6) & 0x3f);
        code_units->code[2] = 0x80 | (code_point & 0x3f);
    } else if (code_point < 0x200000) { //<22
        code_units->length = 4;
        code_units->code[0] = 0xf0 | (code_point >> 18);
        code_units->code[1] = 0x80 | ((code_point >> 12) & 0x3f);
        code_units->code[2] = 0x80 | ((code_point >> 6) & 0x3f);
        code_units->code[3] = 0x80 | (code_point & 0x3f);
    } else {
        printf("Не удалось закодировать\n");
        return -1;
    }
    return 0;
}

```

В функции `decode` рассматриваем 4 случая, в зависимости от первого байта, декодируем число, соответствующее ему количеству байт.

```

uint32_t decode(const CodeUnits *code_unit) {
    if (code_unit->code[0] < 0x80) {
        return code_unit->code[0];
    } else if (code_unit->code[0] < 0xE0) { //<11100000
        return ((code_unit->code[0] & 0x1f) << 6) | (code_unit->code[1] & 0x3f); //00111111
    } else if (code_unit->code[0] < 0xF0) { //<11110000
        return ((code_unit->code[0] & 0x0f) << 12) | //00001111
            ((code_unit->code[1] & 0x3f) << 6) | (code_unit->code[2] & 0x3f); //00111111
    } else if (code_unit->code[0] < 0xF8) { //<11111000
        return ((code_unit->code[0] & 0x07) << 18) | //00000111
            ((code_unit->code[1] & 0x3f) << 12) | //00111111
            ((code_unit->code[2] & 0x3f) << 6) | (code_unit->code[3] & 0x3f); //00111111
    } else {
        printf("Не удалось декодировать\n");
        return -1;
    }
}

```

В функции `read_next_code_unit` читаем первый байт, в зависимости от его первых бит, обозначающих сколько байт далее относится к тому же числу, заполняем массив `code`. Так же проверяем с помощью масок на битые байты.

```

int read_next_code_unit(FILE *in, CodeUnits *code_units) {
    fread(&(code_units->code[0]), 1, 1, in);
    if (code_units->code[0] < 0x80) {
        code_units->length = 1;
        return 0;
    } else if (code_units->code[0] < 0xE0) { //11100000
        code_units->length = 2;
        fread(&(code_units->code[1]), 1, 1, in);
        if (code_units->code[1] >= 0x80 && code_units->code[1] <= 0xBF) { //
            return 0;
        } else {
            return -1;
        }
    } else if (code_units->code[0] < 0xF0) {
        int count = 0;
        code_units->length = 3;
        for (int i = 1; i < 3; i++) {
            fread(&(code_units->code[i]), 1, 1, in);
            if (code_units->code[i] >= 0x80 && code_units->code[i] <= 0xBF) {
                count++;
            }
        }
        if (count == 2) {
            return 0;
        } else {
            return -1;
        }
    } else if (code_units->code[0] < 0xF8) {
        int count = 0;
        code_units->length = 4;
        for (int i = 1; i < 4; i++) {
            fread(&(code_units->code[i]), 1, 1, in);
            if (code_units->code[i] >= 0x80 && code_units->code[i] <= 0xBF) {
                count++;
            }
        }
        if (count == 3) {
            return 0;
        } else {
            return -1;
        }
    } else {
        return -1;
    }
}

```

В функции write_code_unit записываем массив code в выходной файл.

```

int write_code_unit(FILE *out, const CodeUnits *code_units) {
    return fwrite(code_units->code, 1, code_units->length, out);
}

```

Проверяем, корректно ли программа кодирует.

```

[artem@fedora ex_second]$ xxd -b output.bin
00000000: 11100000 10101010 10000011
[artem@fedora ex_second]$ hexdump -C output.bin
00000000  e0 aa 83
00000003

```

ПРИЛОЖЕНИЕ

ex1.c

```
1  #include <assert.h>
2  #include <stddef.h>
3  #include <stdint.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #define N 1000000
7
8
9  size_t encode_varint(uint32_t value, uint8_t *buf) {
10     assert(buf != NULL);
11     uint8_t *cur = buf;
12     while (value >= 0x80) {
13         const uint8_t byte = (value & 0x7f) | 0x80;
14         *cur = byte;
15         value >>= 7;
16         ++cur;
17     }
18     *cur = value;
19     ++cur;
20     return cur - buf;
21 }
22
23
24 uint32_t decode_varint(const uint8_t **bufp) {
25     const uint8_t *cur = *bufp;
26     uint8_t byte = *cur++;
27     uint32_t value = byte & 0x7f;
28     size_t shift = 7;
29     while (byte >= 0x80) {
30         byte = *cur++;
31         value += (byte & 0x7f) << shift;
32         shift += 7;
33     }
34     *bufp = cur;
35     return value;
36 }
37
38
39 uint32_t generate_number() {
40     const int r = rand();
41     const int p = r % 100;
42     if (p < 90) {
43         return r % 128;
44     }
45     if (p < 95) {
46         return r % 16384;
47     }
48     if (p < 99) {
49         return r % 2097152;
50     }
51     return r % 268435455;
52 }
53
54
55
56
```

```

57 int check(char *c, char *unc) {
58     FILE *uncomp = fopen(unc, "rb");
59     FILE *comp = fopen(c, "rb");
60     if (!uncomp || !comp) {
61         printf("Файл не найден\n");
62         return -1;
63     }
64     fseek(comp, 0, SEEK_END);
65     int size = ftell(comp);
66     fseek(comp, 0, SEEK_SET);
67     uint8_t *compCheck = malloc(size);
68     fread(compCheck, 1, size, comp);
69     uint32_t *uncompCheck = malloc(N * 4);
70     fread(uncompCheck, 4, N, uncomp);
71     const uint8_t *point = compCheck;
72     int count = 0;
73     for (int i = 0; i < N; i++) {
74         if (decode_varint(&point) == uncompCheck[i]) {
75             count++;
76         }
77     }
78     fclose(comp);
79     fclose(uncomp);
80     free(uncompCheck);
81     free(compCheck);
82     if (count == N) {
83         printf("Кодирование успешно\n");
84         return 0;
85     }
86     return -1;
87 }
88
89 int main() {
90     char *unc = "uncompressed.dat";
91     FILE *uncomp = fopen(unc, "wb");
92     char *c = "compressed.dat";
93     FILE *comp = fopen(c, "wb");
94     if (!uncomp || !comp) {
95         printf("Файл не найден\n");
96         return -1;
97     }
98     uint32_t *uncompressed = malloc(N * sizeof(uint32_t));
99     for (int i = 0; i < N; i++) {
100         uncompressed[i] = generate_number();
101     }
102     fwrite(uncompressed, sizeof(uint32_t), N, uncomp);
103     uint8_t buf[4];
104     uint8_t *compressed = malloc(N * 4);
105     uint8_t *point = compressed;
106     size_t size;
107     for (int i = 0; i < N; i++) {
108         size = encode_varint(uncompressed[i], buf);
109         for (int j = 0; j < size; j++) {
110             *compressed = buf[j];
111             compressed++;
112         }
113     }
114 }

```

```
128     }
129     fwrite(point, sizeof(uint8_t), compressed - point, comp);
130     free(point);
131     free(uncompressed);
132     fclose(comp);
133     fclose(uncomp);
    check(c, unc);
    return 0;
}
```

main.c

```
1  #include <stdio.h>
2  #include <string.h>
3
4  #include "coder.h"
5  #include "command.h"
6
7  int main(int argc, char *argv[]) {
8      if (argc != 4) {
9          printf("Неверное количество аргументов\n");
10         return -1;
11     }
12     if (strcmp(argv[1], "decode") == 0) {
13         decode_file(argv[2], argv[3]);
14     } else if (strcmp(argv[1], "encode") == 0) {
15         encode_file(argv[2], argv[3]);
16     } else {
17         printf("Неверная команда, используйте encode/decode\n");
18         return -1;
19     }
20     return 0;
21 }
22 }
```


coder.c

```
1  #include <inttypes.h>
2  #include <stdio.h>
3
4
5  #include "coder.h"
6  #include "command.h"
7
8  int encode(uint32_t code_point, CodeUnits *code_units) {
9      if (code_point < 0x80) { //<8
10         code_units->length = 1;
11         code_units->code[0] = code_point;
12     } else if (code_point < 0x800) { //<12
13         code_units->length = 2;
14         code_units->code[0] = 0xc0 | (code_point >> 6);
15         code_units->code[1] = 0x80 | (code_point & 0x3f); //00111111
16     } else if (code_point < 0x10000) { //<17
17         code_units->length = 3;
18         code_units->code[0] = 0xe0 | (code_point >> 12);
19         code_units->code[1] = 0x80 | ((code_point >> 6) & 0x3f);
20         code_units->code[2] = 0x80 | (code_point & 0x3f);
21     } else if (code_point < 0x200000) { //<22
22         code_units->length = 4;
23         code_units->code[0] = 0xf0 | (code_point >> 18);
24         code_units->code[1] = 0x80 | ((code_point >> 12) & 0x3f);
25         code_units->code[2] = 0x80 | ((code_point >> 6) & 0x3f);
26         code_units->code[3] = 0x80 | (code_point & 0x3f);
27     } else {
28         printf("Не удалось закодировать\n");
29         return -1;
30     }
31     return 0;
32 }
33
34 uint32_t decode(const CodeUnits *code_unit) {
35     if (code_unit->code[0] < 0x80) {
36         return code_unit->code[0];
37     } else if (code_unit->code[0] < 0xE0) { //<11100000
38         return ((code_unit->code[0] & 0x1f) << 6) | (code_unit->code[1] &
39 0x3f); //00111111
40     } else if (code_unit->code[0] < 0xF0) { //<11110000
41         return ((code_unit->code[0] & 0x0f) << 12) | //00001111
42 ((code_unit->code[1] & 0x3f) << 6) | (code_unit->code[2] &
43 0x3f); //00111111
44     } else if (code_unit->code[0] < 0xF8) { //<11111000
45         return ((code_unit->code[0] & 0x07) << 18) | //00000111
46 ((code_unit->code[1] & 0x3f) << 12) | //00111111
47 ((code_unit->code[2] & 0x3f) << 6) | (code_unit->code[3] &
48 0x3f); //00111111
49     } else {
50         printf("Не удалось декодировать\n");
51         return -1;
52     }
53 }
54
55 }
```

```

61 int read_next_code_unit(FILE *in, CodeUnits *code_units) {
62     fread(&(code_units->code[0]), 1, 1, in);
63     if (code_units->code[0] < 0x80) {
64         code_units->length = 1;
65         return 0;
66     } else if (code_units->code[0] < 0xE0) { //11100000
67         code_units->length = 2;
68         fread(&(code_units->code[1]), 1, 1, in);
69         if (code_units->code[1] >= 0x80 && code_units->code[1] <= 0xBF)
70             { // в отрезке от 10000000 до 10111111
71                 return 0;
72             } else {
73                 return -1;
74             }
75     } else if (code_units->code[0] < 0xF0) {
76         int count = 0;
77         code_units->length = 3;
78         for (int i = 1; i < 3; i++) {
79             fread(&(code_units->code[i]), 1, 1, in);
80             if (code_units->code[i] >= 0x80 && code_units->code[i] <= 0xBF)
81                 {
82                     count++;
83                 }
84             }
85         if (count == 2) {
86             return 0;
87         } else {
88             return -1;
89         }
90     } else if (code_units->code[0] < 0xF8) {
91         int count = 0;
92         code_units->length = 4;
93         for (int i = 1; i < 4; i++) {
94             fread(&(code_units->code[i]), 1, 1, in);
95             if (code_units->code[i] >= 0x80 && code_units->code[i] <= 0xBF)
96                 {
97                     count++;
98                 }
99             }
100         if (count == 3) {
101             return 0;
102         } else {
103             return -1;
104         }
105     } else {
106         return -1;
107     }
108 }
109
110 int write_code_unit(FILE *out, const CodeUnits *code_units) {
111     return fwrite(code_units->code, 1, code_units->length, out);
112 }
113
114 }
115
116 }
117
118 }
119
120 }
121
122 }
123

```

command.c

```
1  #include "coder.h"
2  #include "command.h"
3
4  int encode_file(const char *in_file_name, const char *out_file_name)
5  {
6      FILE *input = fopen(in_file_name, "r");
7      FILE *output = fopen(out_file_name, "wb");
8      if (!input || !output) {
9          printf("Файл не найден\n");
10         return -1;
11     }
12
13
14     uint32_t hex;
15     CodeUnits codeunit;
16     while (fscanf(input, "%" SCNx32, &hex) == 1) {
17         if (encode(hex, &codeunit) == 0) {
18             write_code_unit(output, &codeunit);
19         } else {
20             return -1;
21         }
22     }
23     fclose(input);
24     fclose(output);
25     printf("Успешно\n");
26     return 0;
27 }
28
29
30
31 int decode_file(const char *in_file_name, const char *out_file_name)
32 {
33     FILE *input = fopen(in_file_name, "rb");
34     FILE *output = fopen(out_file_name, "w");
35     if (!input || !output) {
36         printf("Файл не найден\n");
37         return -1;
38     }
39
40
41     fseek(input, 0, SEEK_END);
42     int size = ftell(input);
43     fseek(input, 0, SEEK_SET);
44     while (ftell(input) != size) {
45         CodeUnits codeunit;
46         if (read_next_code_unit(input, &codeunit) == 0) {
47             fprintf(output, "%" PRIx32 "\n", decode(&codeunit));
48         } else {
49             continue;
50         }
51     }
52     fclose(input);
53     fclose(output);
54     printf("Успешно\n");
55     return 0;
56 }
57 }
```

coder.h

```
1  #include <stdint.h>
2  #include <stdio.h>
3  #pragma once
4
5  enum {
6      MaxCodeLength = 4
7  };
8
9
10 typedef struct {
11     uint8_t code[MaxCodeLength];
12     size_t length;
13 } CodeUnits;
14
15
16 int encode(uint32_t code_point, CodeUnits *code_units);
17 uint32_t decode(const CodeUnits *code_unit);
18 int read_next_code_unit(FILE *in, CodeUnits *code_units);
19 int write_code_unit(FILE *out, const CodeUnits *code_unit);
```

command.h

```
1
2     #include <stdint.h>
3     #include <stdio.h>
4     #include <inttypes.h>
5     #pragma once
6
7
8     int encode_file(const char *in_file_name, const char *out_file_name);
9     int decode_file(const char *in_file_name, const char *out_file_name);
10    int write_code_unit(FILE *out, const CodeUnits *code_unit);
11
```