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. ИС-242
«5» мая 2023 г. _____ /Макиенко В.С./

Проверил:
Ст. Преподаватель
«12» мая 2023 г. _____ /Фульман В.О./

Оценка «_____»

Новосибирск 2020

ОГЛАВЛЕНИЕ

ЗАДАНИЕ

Задание №1:

Необходимо разработать приложение, которое генерирует 1000000 случайных чисел и записывает их в два бинарных файла. В файл uncompressed.dat нужно записать числа в не□сжатом формате, в файл compressed.dat — в формате varint. Вывести коэффициент сжатия для данных файлов. Также необходимо реализовать чтение чисел из двух файлов. Добавите проверку: последовательности чисел из двух файлов должны совпадать. В работе нужно использовать следующие варианты функций кодирования и декодирования:

```
#include <assert.h>
#include <stddef.h>
#include <stdint.h>

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;
}
```

Использование формата varint наиболее эффективно в случаях, когда подавляющая доля чисел имеет небольшие значения. Для выполнения работы использую функцию генерации случайных чисел:

```c
#include <stdint.h>

/*
 * Диапазон              Вероятность
 * --------------------- -----------
 * [0; 128)              90%
 * [128; 16384)          5%
 * [16384; 2097152)      4%
 * [2097152; 268435455) 1%
 */
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;
}
```

Задание №2: Разработать приложение для кодирования и декодирования чисел в кодировки UTF-8. Запуск программы должен осуществляться через аргументы командной строки.

ВЫПОЛНЕНИЕ РАБОТЫ

Задание 1:

Функция encode_varint:

```
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;
}
```

С помощью побитовых операций мы кодируем число. После выполнения функция возвращает количество байт которое понадобилось для кодировки числа. В buf мы храним уже закодированное число и value где храниться число без кодировки.

Функция decode_varint:

```
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;
}
```

Функция принимает на вход указатель на массив с закодированными числами. В самой функции выполняются побитовые операции необходимые для декодирования числа. В конце работы функции перезаписывается

указатель на массив с закодированными числами, указатель хранит адрес следующего закодированного числа. Функция возвращает декодированное число.

Функция main:

```
int main()
{
    FILE *uncompWrite = fopen("uncompressed.dat", "wb");
    FILE *compWrite = fopen("compressed.dat", "wb");

    for (int i = 0; i < 1000000; i++)
    {
        uint32_t value = generate_number();
        fwrite(&value, sizeof(value), 1, uncompWrite);
        uint8_t buf[4] = {};
        size_t size = encode_varint(value, buf);
        fwrite(buf, sizeof(*buf), size, compWrite);
    }
    fclose(uncompWrite);
    fclose(compWrite);
```

В начале основной функции мы создаем два файла для записи чисел. Далее мы генерируем по одному случайному числу пока их не будет 1000000 и записываем их в файл с несжатыми числами, после кодируем эти числа и получаем сжатые числа, которые записываем во второй файл для сжатых чисел. После того как записаны все числа закрываем файлы.

```
FILE *uncompRead = fopen("uncompressed.dat", "rb");
FILE *compRead = fopen("compressed.dat", "rb");

fseek(compRead, 0, SEEK_END);
size_t endfile = ftell(compRead);
fseek(compRead, 0, SEEK_SET);

fseek(uncompRead, 0, SEEK_END);
size_t uncomp_size = ftell(uncompRead);
fseek(uncompRead, 0, SEEK_SET);

uint8_t compressed[endfile];
fread(compressed, sizeof(*compressed), endfile, compRead);
uint32_t uncompressed[1000000];
fread(uncompressed, sizeof(*uncompressed), 1000000, uncompRead);
```

После открываем снова эти файлы для чтения. В переменную endfile сохраняем количество байт в сжатом файле, а в переменную uncomp_size

количество байт в несжатом файле. Далее создаем массив compressed размером endfile и записываем туда все закодированные числа в сжатом файле. Тоже самое делаем для с несжатыми числами, сохраняем их в массив uncompressed

```
    const uint8_t *cur_comp = compressed;
    int i = 0;
    while (cur_comp < compressed + endfile)
    {
        if (uncompressed[i] != decode_varint(&cur_comp))
        {
            printf("Numbers with index %d not equal\n", i);
            i = 0;
            break;
        }
        i++;
    }

    if (i)
    {
        printf("All numbers from two files are equal\n");
    }

    fclose(uncompRead);
    fclose(compRead);

    double k = (double)uncomp_size / endfile;
    printf("Compression ratio = %f\n", k);

    return 0;
}
```

Далее мы декодируем сжатые числа и сравниваем их с соответствующими им не сжатыми числами. Если все числа равны, то выводим сообщение об успешном завершении программы и коэффициент сжатия, в противном случае выводим сообщение об ошибке с индексом числа, которое не совпало со своей сжатой копией.

Запустим программу

```
linux@DESKTOP-U3072BL:~/Progga/ex1$ ./main
All numbers from two files are equal
Compression ratio = 3.450635
```

Она сработала без ошибок и выдала что коэффициент сжатия 3.45

Задание №2:

Функция main:

Запуск программы начинается с файла main.c в него поступает 3 аргумента (комманда, файл из которого брать данные, файл в который записать обработанные данные). Далее исходя из первого аргумента мы выполняем кодирование или декодирование и работаем с файлами из второго и третьего аргумента.

```c
int main(int argc, char *argv[])
{
    if (argc != 4)
    {
        printf("EROR: Wrong number of arguments\n");
        return 1;
    }

    const char *command = argv[1];
    const char *in_file_name = argv[2];
    const char *out_file_name = argv[3];

    if (strcmp(command, "encode") == 0)
    {
        encode_file(in_file_name, out_file_name);
    }
    else if (strcmp(command, "decode") == 0)
    {
        decode_file(in_file_name, out_file_name);
    }
    else
    {
        printf("EROR: Wrong command\n");
        return 1;
    }
    return 0;
}
```

Функция encode:

На вход передаются два параметра code_point – число, которое мы хотим закодировать, и указатель на структуру Code_Units, куда мы сохраним закодированное число.

```
int encode(uint32_t code_point, CodeUnits *code_units)
{
    if (code_point < 0x80)
    {
        code_units->length = 1;
        code_units->code[0] = code_point;
    }
    else if (code_point < 0x800)
    {
        code_units->length = 2;
        code_units->code[0] = 0xc0 | (code_point >> 6);
        code_units->code[1] = 0x80 | (code_point & 0x3f);
    }
    else if (code_point < 0x10000)
    {
        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)
    {
        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
    {
        return -1;
    }

    return 0;
}
```


В функции, при помощи битовых операций определяется сколько байт потребуется для закодированного числа и происходит кодирование чисел по определенным правилам. Функция возвращает 0, если кодирование прошло успешно, иначе -1

Функция write_code_unit:

Функция принимает на вход указатель на выходной поток, куда мы хотим записать закодированное число и указатель на структуру Code_Units с закодированным числом. 10 Функция возвращает количество записанных байт в файл.

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

Функция encode_file:

```
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)
    {
        printf("ERROR: Wrong source to input file\n");
        return -1;
    }
    if (!output)
    {
        fclose(input);
        return -1;
    }

    fseek(input, 0, SEEK_END);
    size_t end_of_input_file_stream = ftell(input);
    fseek(input, 0, SEEK_SET);
```

```
    while(ftell(input) < end_of_input_file_stream)
    {
        uint32_t code_point;
        fscanf(input, "%" SCNx32, &code_point);

        CodeUnits code_unit;
        if (encode(code_point, &code_unit))
        {
            printf("ERROR: Failed to encode number\n");
            return -1;
        }
        write_code_unit(output, &code_unit);
    }

    fclose(input);
    fclose(output);
    return 0;
}
```

В функции мы из файла с числами (которые будем кодировать) считываем по одному шестнадцатеричному числу, далее они кодируются при помощи функции encode и записываются в выходной файл при помощи функции write_code_unit.

Функция read_next_code_unit:

На вход функции передается указатель на входной поток, откуда будут считаны закодированные числа и указатель на структуру Code_Units куда мы сохраним считанное закодированное число.

```c
int read_next_code_unit(FILE *in, CodeUnits *code_units)
{
    code_units->length = 0;
    while (code_units->length == 0 && !feof(in))
    {
        uint8_t *buf = code_units->code;
        fread(buf, sizeof(uint8_t), 1, in);
        if ((*buf & 0x80) == 0x00)
        {
            code_units->length = 1;
        }
        else if ((*buf >> 5) == 0x06)
        {
            code_units->length = 2;
            buf++;
            fread(buf, sizeof(uint8_t), 1, in);
        }
        else if ((*buf >> 4) == 0x0e)
        {
            code_units->length = 3;
            buf++;
            fread(buf, sizeof(uint8_t), 2, in);
        }
        else if ((*buf >> 3) == 0x1e)
        {
            code_units->length = 4;
            buf++;
            fread(buf, sizeof(uint8_t), 3, in);
        }
    }
    return 0;
}
```

Функция считывает по одному байту с файла до тех пор, пока не встретится корректный старший байт. При помощи битовых операций проверяется корректность старшего байта, а после до записываются остальные байты закодированного числа. Функция возвращает 0 в случае успеха, иначе -1.

Функция decode:

Функция принимает на вход указатель структуру Code_Units куда сохранено число, которое мы хотим декодировать.

```
uint32_t decode(const CodeUnits *code_unit)
{
    uint32_t *code_point;
    uint8_t buf[4];
    if (code_unit->length == 1)
    {
        return (uint32_t)code_unit->code[0];
    }
    else if (code_unit->length == 2)
    {
        buf[0] = (code_unit->code[0] << 6) | ((code_unit->code[1] << 2) >> 2);
        buf[1] = ((code_unit->code[0] & 0x1f) >> 2);
        code_point = (uint32_t *)buf;
        return *code_point;
    }
    else if (code_unit->length == 3)
    {
        buf[0] = (code_unit->code[2] & 0x3f) | (code_unit->code[1] << 6);
        buf[1] = ((code_unit->code[1] >> 2) & 0x0f) | (code_unit->code[0] << 4);
        code_point = (uint32_t *)buf;
        return *code_point;
    }
    else if (code_unit->length == 4)
    {
        buf[0] = (code_unit->code[3] & 0x3f) | (code_unit->code[2] << 6);
        buf[1] = ((code_unit->code[2] >> 2) & 0x0f) | (code_unit->code[1] << 4);
        buf[2] = ((code_unit->code[1] >> 4) & 0x03) | ((code_unit->code[0] & 0x07) << 2);
        code_point = (uint32_t *)buf;
        return *code_point;
    }
}
```

При помощи битовых операций восстанавливается изначальный вид числа. Функция возвращает декодированное число.

Функция decode_file:

Функция принимает на вход две строки, одна с названием входного файлового потока, другая с названием выходного файлового потока.

```c
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)
    {
        printf("ERROR: Wrong source to input file\n");
        return -1;
    }
    if (!output)
    {
        fclose(input);
        return -1;
    }

    fseek(input, 0, SEEK_END);
    size_t end_of_input_file_stream = ftell(input);
    fseek(input, 0, SEEK_SET);

    while(ftell(input) < end_of_input_file_stream)
    {
        CodeUnits code_unit;
        if (read_next_code_unit(input, &code_unit))
        {
            printf("ERROR: Failed reading next code unit. End of file reached\n");
        }
        uint32_t code_point = decode(&code_unit);
        fprintf(output, "%" PRIx32 "\n", code_point);
    }

    fclose(input);
    fclose(output);
}
```

В функции считываются по одному закодированному числу при помощи функции read_next_code_unit. Далее считанное число декодируется функцией decode и записывается в выходной файл.

ПРИЛОЖЕНИЕ

Исходный код с комментариями;

main.c

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

```
49  uint32_t decode_varint(const uint8_t **bufp)
50  {
51      const uint8_t *cur = *bufp;
52      uint8_t byte = *cur++;
53      uint32_t value = byte & 0x7f;
54      size_t shift = 7;
55      while (byte >= 0x80)
56      {
57          byte = *cur++;
58          value += (byte & 0x7f) << shift;
59          shift += 7;
60      }
61      *bufp = cur;
62      return value;
63  }
64
65  int main()
66  {
67      FILE *uncompWrite = fopen("uncompressed.dat", "wb");
68      FILE *compWrite = fopen("compressed.dat", "wb");
69
70      for (int i = 0; i < 1000000; i++)
71      {
72          uint32_t value = generate_number();
73          fwrite(&value, sizeof(value), 1, uncompWrite);
74          uint8_t buf[4] = {};
75          size_t size = encode_varint(value, buf);
76          fwrite(buf, sizeof(*buf), size, compWrite);
77      }
78      fclose(uncompWrite);
79      fclose(compWrite);
80
81      FILE *uncompRead = fopen("uncompressed.dat", "rb");
82      FILE *compRead = fopen("compressed.dat", "rb");
83
84      fseek(compRead, 0, SEEK_END);
85      size_t endfile = ftell(compRead);
86      fseek(compRead, 0, SEEK_SET);
87
88      fseek(uncompRead, 0, SEEK_END);
89      size_t uncomp_size = ftell(uncompRead);
90      fseek(uncompRead, 0, SEEK_SET);
91
92      uint8_t compressed[endfile];
93      fread(compressed, sizeof(*compressed), endfile, compRead);
94      uint32_t uncompressed[1000000];
95      fread(uncompressed, sizeof(*uncompressed), 1000000, uncompRead);
96
97      const uint8_t *cur_comp = compressed;
98      int i = 0;
99      while (cur_comp < compressed + endfile)
100     {
101         if (uncompressed[i] != decode_varint(&cur_comp))
102         {
103             printf("Numbers with index %d not equal\n", i);
104             i = 0;
105             break;
```

```
106             }
107             i++;
108         }
109
110         if (i)
111         {
112             printf("All numbers from two files are equal\n");
113         }
114
115         fclose(uncompRead);
116         fclose(compRead);
117
118         double k = (double)uncomp_size / endfile;
119         printf("Compression ratio = %f\n", k);
120
121         return 0;
122     }
```

Задание №2:

main.c

```
1     #include <stdio.h>
 2    #include <stdint.h>
 3    #include <stdlib.h>
 4    #include <string.h>
 5    #include <inttypes.h>
 6
 7    #include "coder.h"
 8    #include "command.h"
 9
10    int main(int argc, char *argv[])
11    {
12        if (argc != 4)
13        {
14            printf("EROR: Wrong number of arguments\n");
15            return 1;
16        }
17
18        const char *command = argv[1];
19        const char *in_file_name = argv[2];
20        const char *out_file_name = argv[3];
21
22        if (strcmp(command, "encode") == 0)
23        {
24            encode_file(in_file_name, out_file_name);
25        }
26        else if (strcmp(command, "decode") == 0)
27        {
28            decode_file(in_file_name, out_file_name);
29        }
30        else
31        {
32            printf("EROR: Wrong command\n");
33            return 1;
```

```
34          }
35          return 0;
36      }
```

command.c

```
 1      #include <stdlib.h>
 2      #include <stdio.h>
 3      #include <stdint.h>
 4      #include <inttypes.h>
 5
 6      #include "coder.h"
 7      #include "command.h"
 8
 9      int encode_file(const char *in_file_name, const char *out_file_name)
10      {
11          FILE *input = fopen(in_file_name, "r");
12          FILE *output = fopen(out_file_name, "wb");
13
14          if (!input)
15          {
16              printf("ERROR: Wrong source to input file\n");
17              return -1;
18          }
19          if (!output)
20          {
21              fclose(input);
22              return -1;
23          }
24
25          fseek(input, 0, SEEK_END);
26          size_t end_of_input_file_stream = ftell(input);
27          fseek(input, 0, SEEK_SET);
28
29          while (ftell(input) < end_of_input_file_stream)
30          {
31              uint32_t code_point;
32              fscanf(input, "%" SCNx32, &code_point);
33
34              CodeUnits code_unit;
35              if (encode(code_point, &code_unit))
36              {
37                  printf("ERROR: Failed to encode number\n");
38                  return -1;
39              }
40              write_code_unit(output, &code_unit);
41          }
42
43          fclose(input);
44          fclose(output);
45          return 0;
46      }
47
48      int decode_file(const char *in_file_name, const char *out_file_name)
49      {
```

```
50        FILE *input = fopen(in_file_name, "rb");
51        FILE *output = fopen(out_file_name, "w");
52
53        if (!input)
54        {
55            printf("ERROR: Wrong source to input file\n");
56            return -1;
57        }
58        if (!output)
59        {
60            fclose(input);
61            return -1;
62        }
63
64        fseek(input, 0, SEEK_END);
65        size_t end_of_input_file_stream = ftell(input);
66        fseek(input, 0, SEEK_SET);
67
68        while (ftell(input) < end_of_input_file_stream)
69        {
70            CodeUnits code_unit;
71            if (read_next_code_unit(input, &code_unit))
72            {
73                printf("ERROR: Failed reading next code unit. End of file
74    reached\n");
75            }
76            uint32_t code_point = decode(&code_unit);
77            fprintf(output, "%" PRIx32 "\n", code_point);
78        }
79
80        fclose(input);
81        fclose(output);
    }
```

coder.c

```
1   #include <stdint.h>
2   #include <stdio.h>
3   #include <inttypes.h>
4
5   #include "coder.h"
6   #include "command.h"
7
8   int encode(uint32_t code_point, CodeUnits *code_units)
9   {
10      if (code_point < 0x80)
11      {
12          code_units->length = 1;
13          code_units->code[0] = code_point;
14      }
15      else if (code_point < 0x800)
16      {
17          code_units->length = 2;
18          code_units->code[0] = 0xc0 | (code_point >> 6);
19          code_units->code[1] = 0x80 | (code_point & 0x3f);
```

```
20      }
21      else if (code_point < 0x10000)
22      {
23          code_units->length = 3;
24          code_units->code[0] = 0xe0 | (code_point >> 12);
25          code_units->code[1] = 0x80 | ((code_point >> 6) & 0x3f);
26          code_units->code[2] = 0x80 | (code_point & 0x3f);
27      }
28      else if (code_point < 0x200000)
29      {
30          code_units->length = 4;
31          code_units->code[0] = 0xf0 | (code_point >> 18);
32          code_units->code[1] = 0x80 | ((code_point >> 12) & 0x3f);
33          code_units->code[2] = 0x80 | ((code_point >> 6) & 0x3f);
34          code_units->code[3] = 0x80 | (code_point & 0x3f);
35      }
36      else
37      {
38          return -1;
39      }
40
41      return 0;
42  }
43
44  uint32_t decode(const CodeUnits *code_unit)
45  {
46      uint32_t *code_point;
47      uint8_t buf[4];
48      if (code_unit->length == 1)
49      {
50          return (uint32_t)code_unit->code[0];
51      }
52      else if (code_unit->length == 2)
53      {
54          buf[0] = (code_unit->code[0] << 6) | ((code_unit->code[1] <<
55  2) >> 2);
56          buf[1] = ((code_unit->code[0] & 0x1f) >> 2);
57          code_point = (uint32_t *)buf;
58          return *code_point;
59      }
60      else if (code_unit->length == 3)
61      {
62          buf[0] = (code_unit->code[2] & 0x3f) | (code_unit->code[1] <<
63  6);
64          buf[1] = ((code_unit->code[1] >> 2) & 0x0f) | (code_unit-
65  >code[0] << 4);
66          code_point = (uint32_t *)buf;
67          return *code_point;
68      }
69      else if (code_unit->length == 4)
70      {
71          buf[0] = (code_unit->code[3] & 0x3f) | (code_unit->code[2] <<
72  6);
73          buf[1] = ((code_unit->code[2] >> 2) & 0x0f) | (code_unit-
74  >code[1] << 4);
75          buf[2] = ((code_unit->code[1] >> 4) & 0x03) | ((code_unit-
76  >code[0] & 0x07) << 2);
77          code_point = (uint32_t *)buf;
```

```
78              return *code_point;
79          }
80      }
81
82      int read_next_code_unit(FILE *in, CodeUnits *code_units)
83      {
84          code_units->length = 0;
85          while (code_units->length == 0 && !feof(in))
86          {
87              uint8_t *buf = code_units->code;
88              fread(buf, sizeof(uint8_t), 1, in);
89              if ((*buf & 0x80) == 0x00)
90              {
91                  code_units->length = 1;
92              }
93              else if ((*buf >> 5) == 0x06)
94              {
95                  code_units->length = 2;
96                  buf++;
97                  fread(buf, sizeof(uint8_t), 1, in);
98              }
99              else if ((*buf >> 4) == 0x0e)
100             {
101                 code_units->length = 3;
102                 buf++;
103                 fread(buf, sizeof(uint8_t), 2, in);
104             }
105             else if ((*buf >> 3) == 0x1e)
106             {
107                 code_units->length = 4;
108                 buf++;
109                 fread(buf, sizeof(uint8_t), 3, in);
110             }
111         }
112         return 0;
        }

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

command.h

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

coder.h

```
1   #include <stdlib.h>
2
3   enum
4   {
```

```
 5        MaxCodeLength = 4
 6    };
 7
 8    typedef struct {
 9        uint8_t code[MaxCodeLength];
10        size_t length;
11    } CodeUnits;
12
13    int encode(uint32_t code_point, CodeUnits *code_units);
14    uint32_t decode(const CodeUnits *code_unit);
15    int read_next_code_unit(FILE *in, CodeUnits *code_units);
16    int write_code_unit(FILE *out, const CodeUnits *code_unit);
```