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](#_30j0zll)

[ВЫПОЛНЕНИЕ РАБОТЫ 6](#_1fob9te)

[ПРИЛОЖЕНИЕ 15](#_tyjcwt)

## **ЗАДАНИЕ**

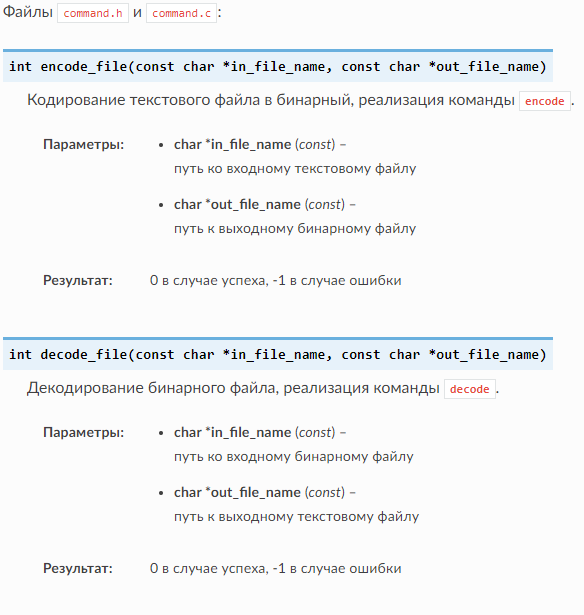
**Задание 1.**

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.

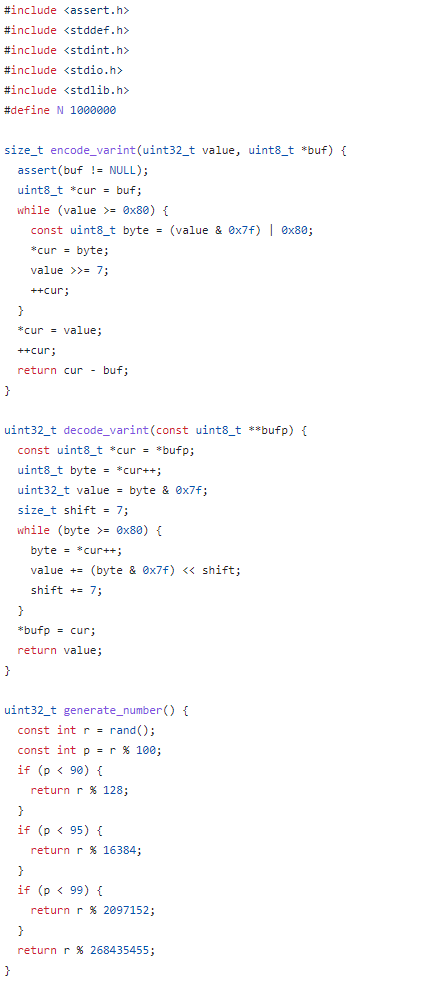




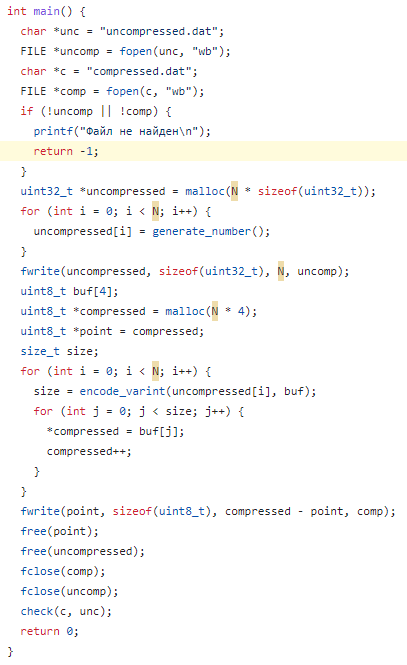
## **ВЫПОЛНЕНИЕ РАБОТЫ**

**Задание 1.**

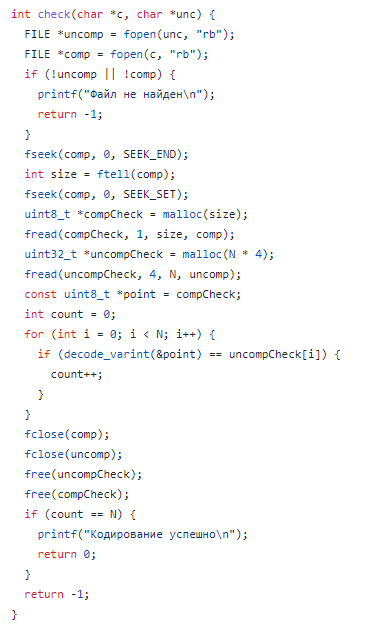
Переносим данные нам функции encode\_varint, decode\_varint и generate\_number.



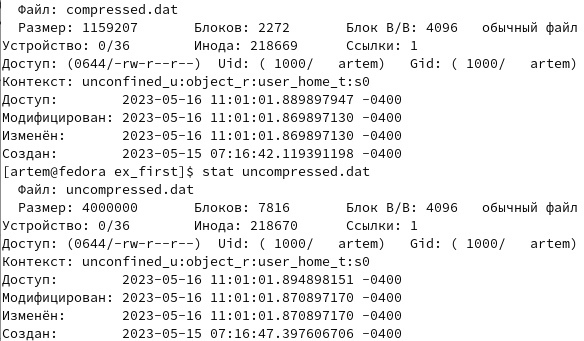
В функции main открываем файлы с ключами wb для работы с файлами в бинарном формате. Проверяем существуют ли файлы. Создаём массив с незакодированными элементами и заполняем его псевдослучайными числами. Записываем эти числа функцией fwrite в файл uncompressed.dat. Создаём массивы buf, для хранения одного 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.



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 compCheck и uncompCheck, заполняем их сжатыми и несжатыми числами с помощью функции fread. Создаём указатель на массив compCheck. 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.

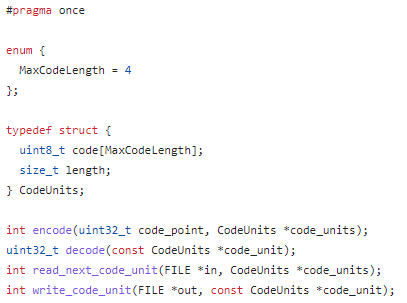


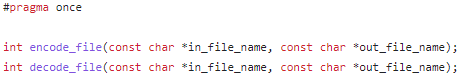
С помощью команды stat проверяем размеры файлов и определяем коэффициент сжатия, он равен 3,45.



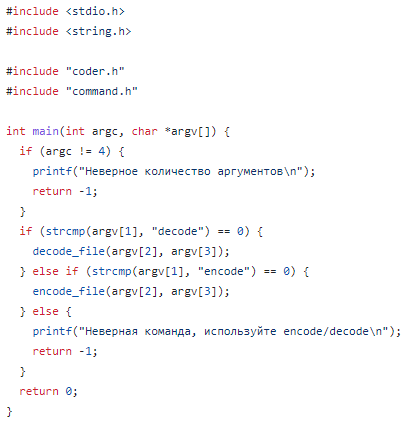
**Задание 2.**

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 длинну числа.



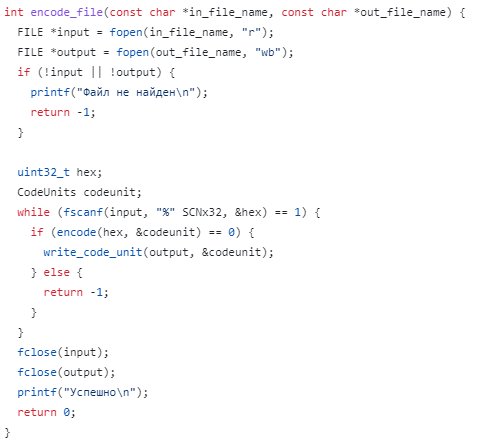


В файле main получаем четыре аргумента при вызове программы, где первый - команда кодировки или декодировки, второй - файл, из которого мы считываем, третий - файл, в который записываем. Проверяем коректность аргументов и в завизимоти от команды вызываем функции encode\_file или decode\_file.



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.

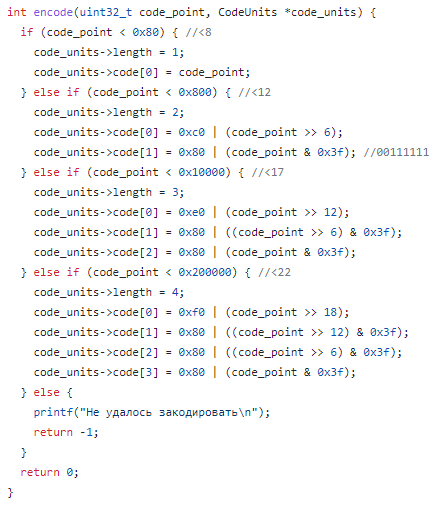


В функции decode\_file открываем файлы и проверяем, существуют ли они. Перемещаем каретку во входном файле с помощью функции fseek и в переменную size записываем размер файла, после чего перемещаем каретку обратно. Пока каретка не дойдёт до конца файла, читаем последовательность codeunit с помощью функции read\_next\_code\_unit. Декодируем числа с помощью функции decode и записываем с помощью функции fprintf в файл. Закрываем файлы.

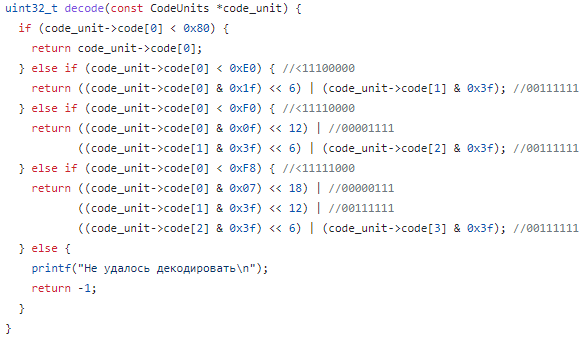


В файле coder.c реализованы функции кодирования и декодирования одного числа, а также функции записи и чтения из файлов.

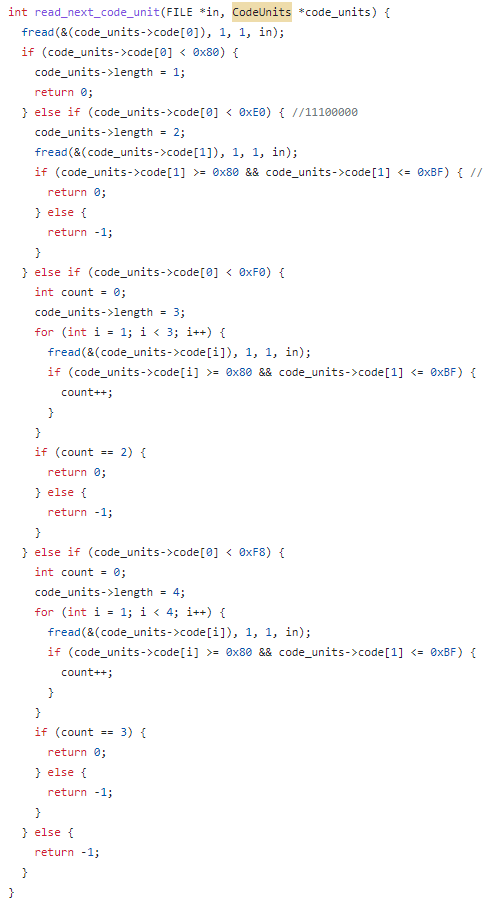
В функции encode рассматриваем 4 случая: если длина двоичного числа меньше 8, для его записи достаточно 1 байта; если меньше 12, то 2 байт; если меньше 17, то 3 байт; если меньше 22, то 4 байт. В поле структуры length записываем количество байт, которое число будет занимать после декодирования. В массив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



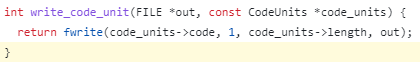
В функции decode рассматриваем 4 случая, в зависимости от первого байта, декодируем число, соответствующее ему количеству байт.



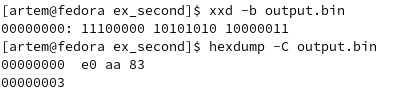
В функции read\_next\_code\_unit читаем первый байт, в зависимости от его первых бит, обозначающих сколько байт далее относится к тому же числу, заполняем массив code. Так же проверяем с помощью масок на битые байты.



В функции write\_code\_unir записываем массив code в выходной файл.



Проверяем, корректно ли программа кодирует.



## 

## **ПРИЛОЖЕНИЕ**

**ex1.c**

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133 | #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**; }  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**; }  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**; } |
| --- | --- |

**main.c**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | #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**; } |

**coder.c**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123 | #include <inttypes.h> #include <stdio.h>  #include "coder.h" #include "command.h"  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**; }  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**;  } }  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**) { // в отрезке от 10000000 до 10111111  **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[**1**] <= **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[**1**] <= **0xBF**) {  count++;  }  }  **if** (count == **3**) {  **return** **0**;  } **else** {  **return** -**1**;  }  } **else** {  **return** -**1**;  } }  int **write\_code\_unit**(**FILE** \*out, **const** CodeUnits \*code\_units) {  **return** fwrite(code\_units->code, **1**, code\_units->length, out); } |

**command.c**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57 | #include "coder.h" #include "command.h"  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, "%" 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**; }  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**; } |

**coder.h**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | #include <stdint.h> #include <stdio.h> #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); |

**command.h**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | #include <stdint.h> #include <stdio.h> #include <inttypes.h> #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); int **write\_code\_unit**(**FILE** \*out, **const** CodeUnits \*code\_unit); |