Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра вычислительных систем

**ОТЧЕТ**

по практической работе 2

по дисциплине «**Программирование**»

|  |  |  |
| --- | --- | --- |
| Выполнил:  студент гр. ИС-242  «21» апреля 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | /Любицкий М.Е./ |
|  |  |  |
| Проверил:  Старший преподаватель кафедры ВС  «21» апреля 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | /Фульман В.О./ |

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_»

Новосибирск 2023

**ОГЛАВЛЕНИЕ**

[ОГЛАВЛЕНИЕ 2](#_Toc132937465)

[ЗАДАНИЕ 3](#_Toc132937466)

[ВЫПОЛНЕНИЕ РАБОТЫ 5](#_Toc132937467)

[ПРИЛОЖЕНИЕ 16](#_Toc132937468)

# **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 несжатом 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:

|  |  |
| --- | --- |
| 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 | **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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | /\*  \* 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:**

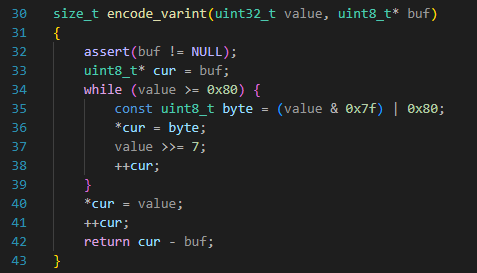
Разработать приложение для кодирования и декодирования чисел в кодировки UTF-8.

Запуск программы должен осуществляться через аргументы командной строки.

# **ВЫПОЛНЕНИЕ РАБОТЫ**

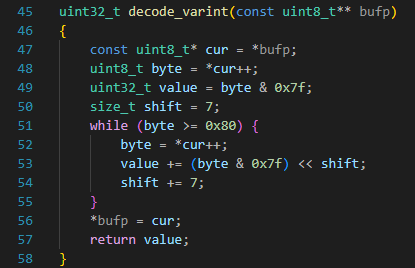
**Задание 1:**

**Функция encode\_varint:**



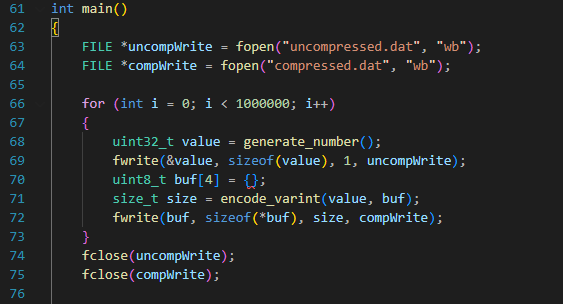
Функция принимает на вход число, которое мы собираемся закодировать и буфер куда сохраним уже закодированное число. В самой функции выполняются побитовые операции необходимые для кодирования числа. Функция возвращает количество байт, потребовавшееся для закодированного числа.

**Функция decode\_varint:**

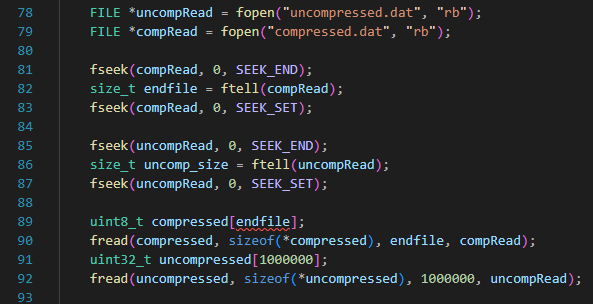


Функция принимает на вход указатель на массив с закодированными числами. В самой функции выполняются побитовые операции необходимые для декодирования числа. В конце работы функции перезаписывается указатель на массив с закодированными числами, указатель хранит адрес следующего закодированного числа. Функция возвращает декодированное число.

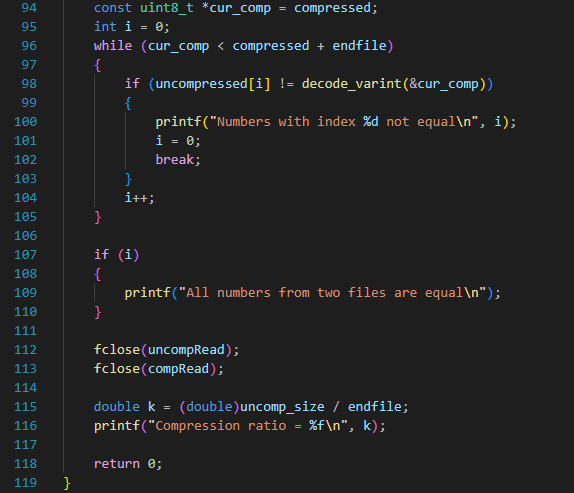
**Функция main:**



В начале основной функции мы создаем два файловых потока для записи чисел. Далее мы генерируем 1000000 случайных чисел 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.

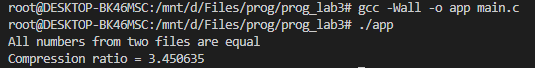


После открываем снова эти файлы для чтения. В переменную endfile сохраняем количество байт в сжатом файле, а в переменную uncomp\_size количество байт в несжатом файле. Далее создаем массив compressed размером endfile и записываем туда все закодированные числа в сжатом файле. Тоже самое делаем для с несжатыми числами, сохраняем их в массив uncompressed.



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 и посмотрим на результат ее работы:

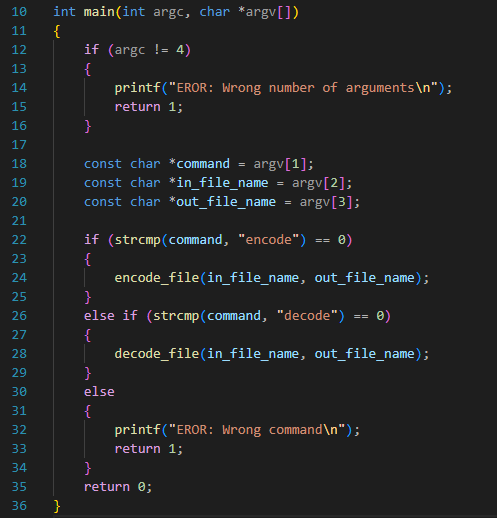


В итоге мы видим, что все числа прошли проверку и коэффициент сжатия составил 3.45.

**Задание 2:**

**Функция main:**

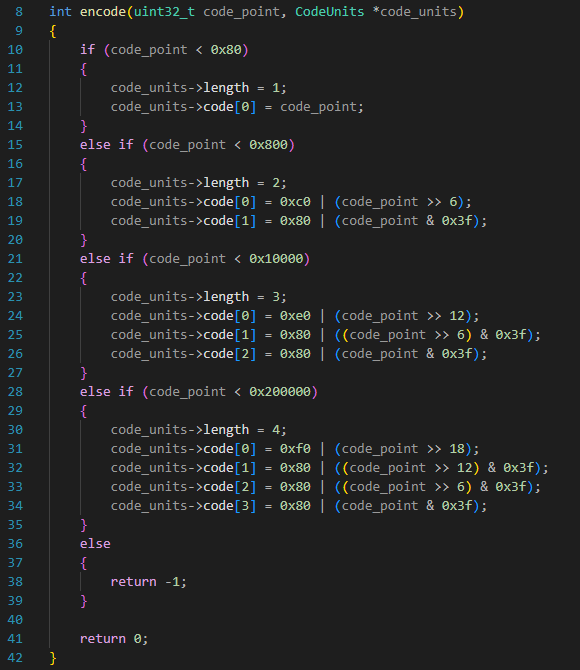
Запуск программы осуществляется при помощи аргументов командной строки для того чтобы закодировать файл нужно указать команду (encode или decode), название входного файла и название выходного файла.



В основной функции проверяется количество аргументов и правильность написанной команды. Если аргументы указаны верно, то происходит выполнение функции encode\_file или decode\_file, в зависимости от указанной команды.

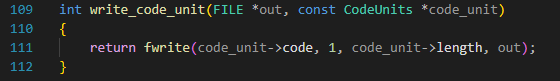
**Функция encode:**

На вход передаются два параметра code\_point – число, которое мы хотим закодировать, и указатель на структуру Code\_Units, куда мы сохраним закодированное число.



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:**

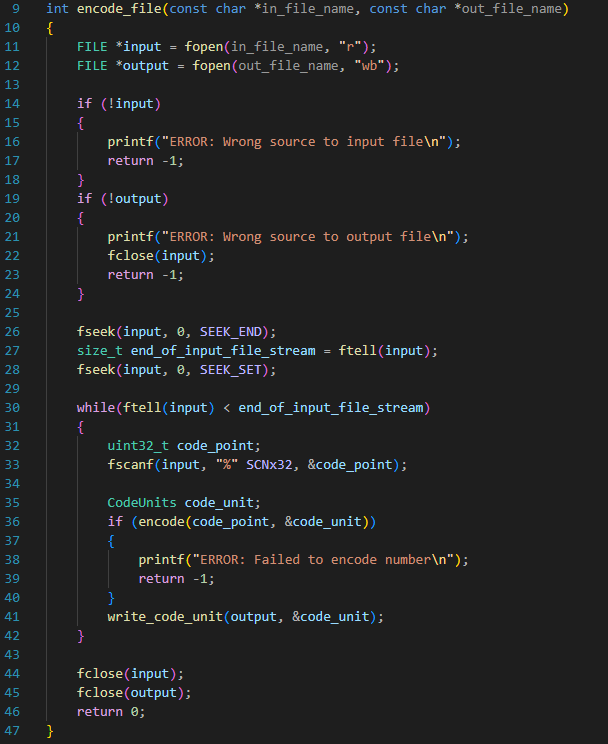


Функция принимает на вход указатель на выходной поток, куда мы хотим записать закодированное число и указатель на структуру Code\_Unitsс закодированным числом.

Функция возвращает количество записанных байт в файл.

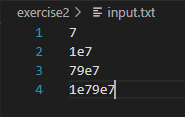
**Функция encode\_file:**

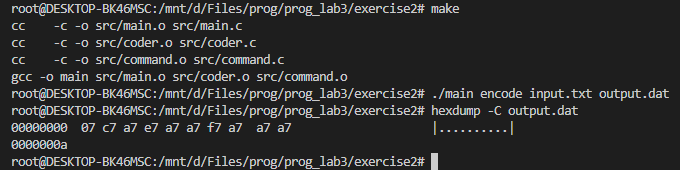
Функция принимает на вход две строки, одна с названием входного файлового потока, другая с названием выходного файлового потока.



В функции с входного потока считываем по одному шестнадцатеричные числа, далее они кодируются при помощи функции encode и записываются в выходной файл при помощи функции write\_code\_unit.

Запишем несколько шестнадцатеричных чисел в файл и посмотрим на работу функции:

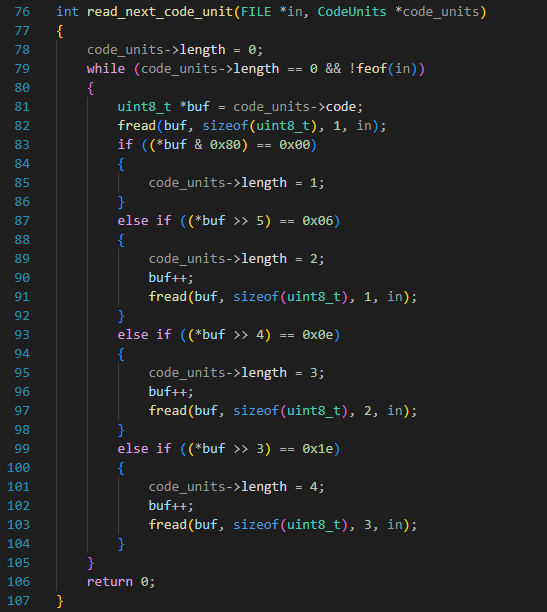




После завершения работы программы мы видим, что в двоичном файле появились наши закодированные числа.

**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.

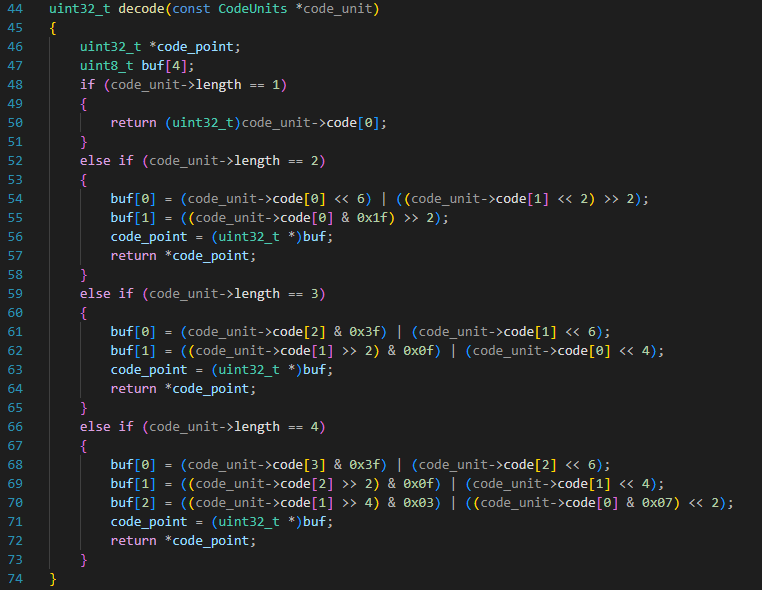


Функция считывает по одному байту с файла до тех пор, пока не встретится корректный лидирующий байт. При помощи битовых операций проверяется корректность лидирующего байта, а после до записываются остальные байты закодированного числа.

Функция возвращает 0 в случае успеха, иначе -1.

**Функция decode:**

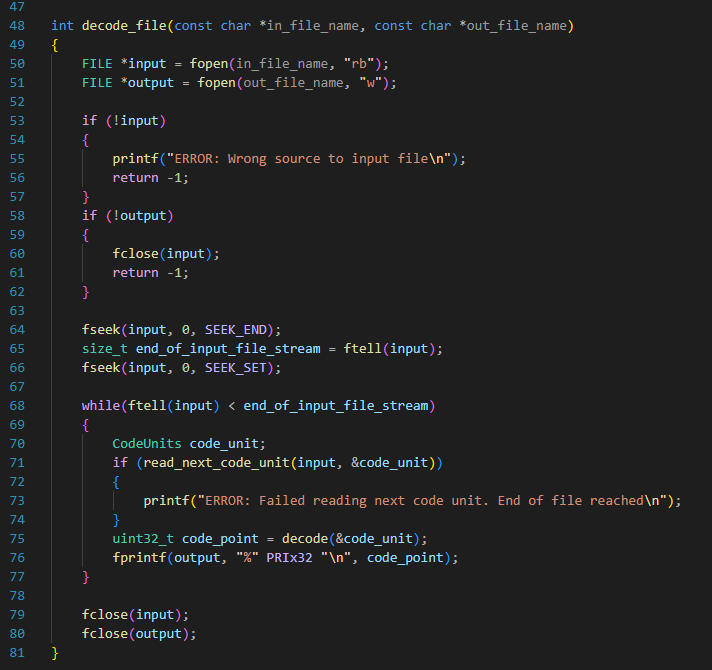
Функция принимает на вход указатель структуру Code\_Units куда сохранено число, которое мы хотим декодировать.



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

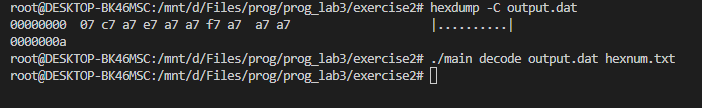
**Функция decode\_file:**

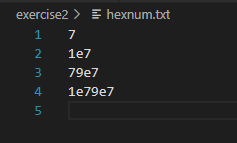
Функция принимает на вход две строки, одна с названием входного файлового потока, другая с названием выходного файлового потока.



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.

Посмотрим, как функция декодирует числа с файла.





Как мы видим, получились те же число что и в самом начале до кодирования. Значит функция работает исправно.

# **ПРИЛОЖЕНИЕ**

Исходный код с комментариями;

Задание 1:

**main.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  81  82  83  84  85  86  87  88  89  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 | #include <assert.h>  #include <stddef.h>  #include <stdint.h>  #include <stdio.h>  #include <stdlib.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**;  }  **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;  }  **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);      **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);  **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**;  } |

Задание 2:

**main.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 | #include <stdio.h>  #include <stdint.h>  #include <stdlib.h>  #include <string.h>  #include <inttypes.h>  #include "coder.h"  #include "command.h"  **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**;  } |

**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  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84 | #include <stdlib.h>  #include <stdio.h>  #include <stdint.h>  #include <inttypes.h>  #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)  {  printf("ERROR: Wrong source to input file**\n**");  **return** -**1**;  }  **if** (!output)  {  printf("ERROR: Wrong source to output file**\n**");  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**;  }  **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)  {  printf("ERROR: Wrong source to output file**\n**");  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);  } |

**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  81  82  83  84  85  86  87  88  89  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 | #include <stdint.h>  #include <stdio.h>  #include <inttypes.h>  #include "coder.h"  #include "command.h"  **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**;  }  **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;  }  }  **int** **read\_next\_code\_unit**(**FILE** \*in, CodeUnits \*code\_units)  {  code\_units->length = **0**;  **while** (code\_units->length == **0**)  {  **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**;  }  **int** **write\_code\_unit**(**FILE** \*out, **const** CodeUnits \*code\_unit)  {  **return** fwrite(code\_unit->code, **1**, code\_unit->length, out);  } |

**command.h**

|  |  |
| --- | --- |
| 1  2 | **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); |

**coder.h**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #include <stdlib.h>  **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); |