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 «10» мая 2023 г

Проверил:

Старший преподаватель Кафедры ВС

«17» мая 2023 г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Денисов М. А.

Фульман В.О.

Оценка **«**\_\_\_\_\_\_\_\_\_\_\_\_\_**»**

Новосибирск 2023

**ОГЛАВЛЕНИЕ**

[ЗАДАНИЕ 3](#_Toc35593781)

[ВЫПОЛНЕНИЕ РАБОТЫ 5](#_Toc35593782)

[ПРИЛОЖЕНИЕ 14](#_Toc35593783)

# **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.

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

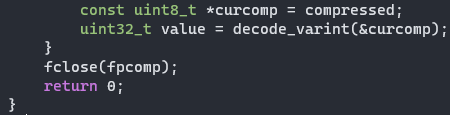
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | #include <assert.h>  #include <stddef.h>  #include <stdint.h>  **size\_t** **encode\_varint**(**uint32\_t** value, **uint8\_t**\* buf)  {  assert(buf != 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;  } |

|  |  |
| --- | --- |
| 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 | #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**;  } |

# 2. Разработать приложение для кодирования и декодирования чисел по описанному выше алгоритму. (UTF-8)

**ВЫПОЛНЕНИЕ РАБОТЫ**

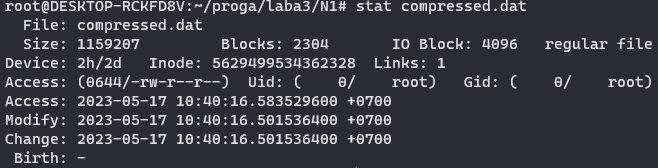
Задание 1



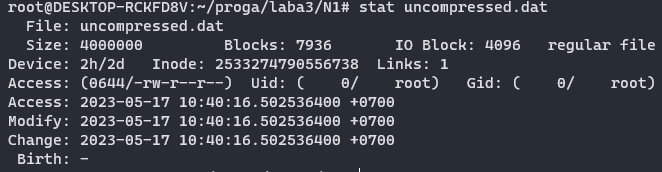
Создадим 2 двоичных файла для записи сжатых и не сжатых чисел. В цикле будем проверять каждое декодированное число на равность с изначальным числом, подовавшиемся в encode. Если они не равны, выводим ошибку.

Далее, в цикле while можем вывести все числа на экран, хранившееся в файле compressed, без единой потери.

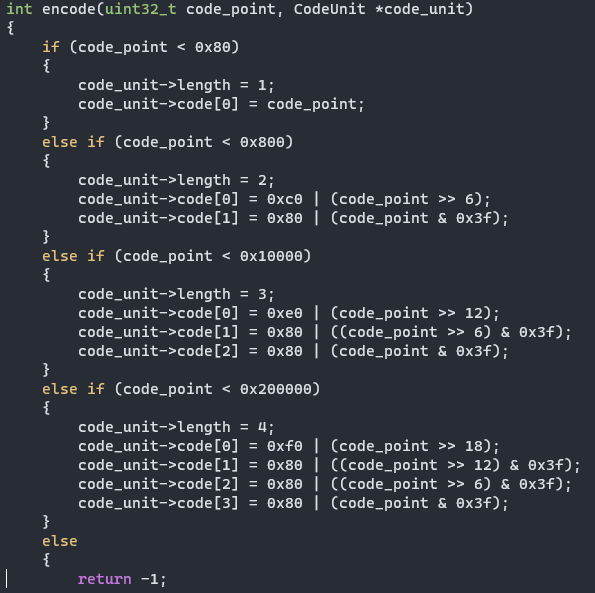
Размер файла compressed.dat



Размер файла uncompressed.dat



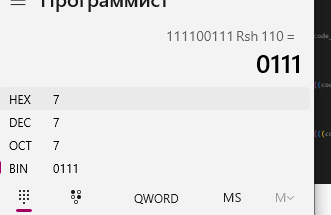
Задание 2

**

Функция encode.

В каждом условии проверяем, сколько число будет занимать байт в представлении UTF-8, если более одного, приводим число к шаблону кодирования и запоминаем длину, то есть, сколько байт потребовалось для записи такого числа.

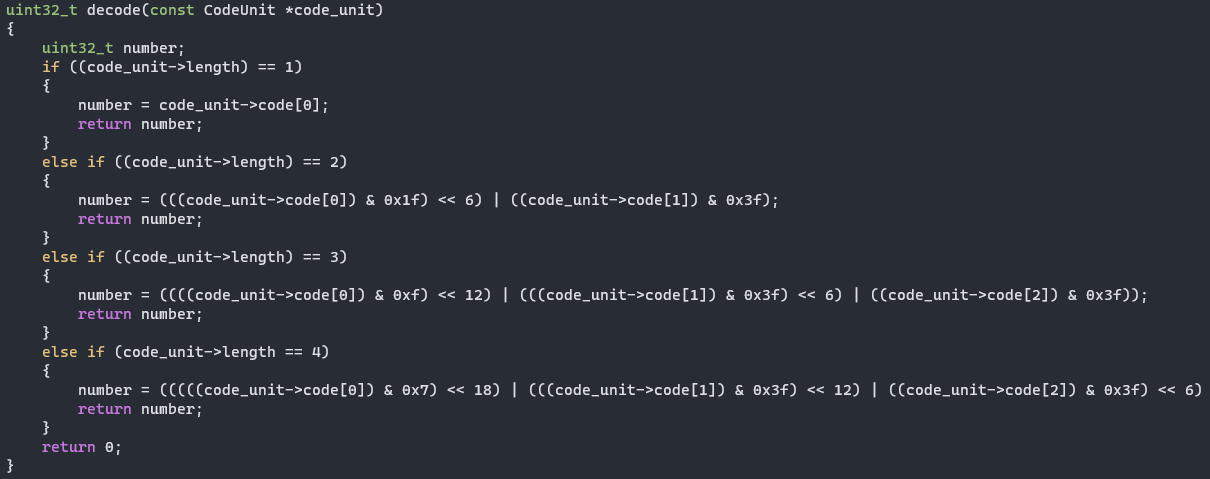
Посмотрим на одном примере, допустим у нас число 2 байта – 1e7. Сдвинем биты вправо на 6



И сделаем побитовое или с числом с0, выглядит оно в двоичном виде 1100 0000. Первый байт такого числа будет выглядеть так.

Получаем то, что и должны были по шаблону.

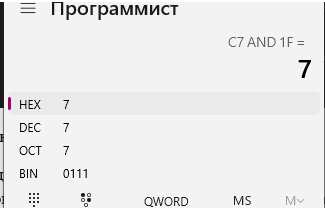
Со вторым байтом аналогично, только другие числа, для получения закодированного второго байта.

**

Функция decode.

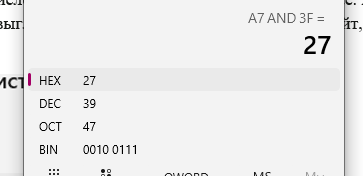
Создам переменную number, где в последствии будет хранится декодированное число. В условиях ветвление провереям, сколько байт занимает закодированное число – длина. После, проводим побитовые операции и приводим число к исходному виду.

Сделаем пример на одном числе, том же, которым и пользовались в encode. 2 байта закодированного числа 1e7 выглядит так – С7 A7. Декодируем первый байт, сначала проводим операцию “и”.

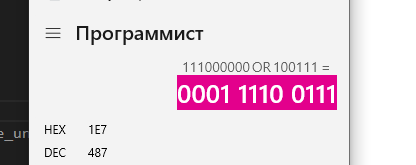
**

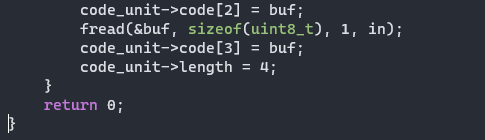
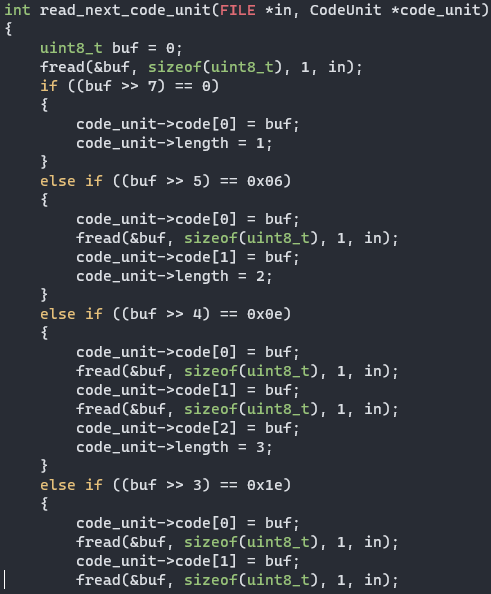
После,сдвигаем влево данное число на 6, получаем 0001 1100 0000.

Теперь, работаем со вторым байтом, получаем 0010 0111.



Теперь, проведем побитовое или, получим 0001 1110 0111, что и есть 1E7 в hex.

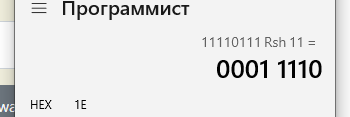




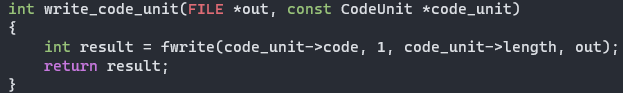
Функция read\_next\_code\_unit.

Считывает последовательно code\_unit из потока in.

Создаем буфер – buf, и сразу читаем в него один бит, после проверяем, какой длине подходит наш байт по шаблону. Если он таковым является, то каждый байт по отдельности записываем в code\_unit – code последовательно, а иначе просто пропускает байт.

Пример: первый байт у закодированного 4 байта числа начинается на 11110. Сдвинув на право в 3, можем увидеть, тот ли у нас байт лежит на данный момент в буфере. У числа 1e79e7 первый закодированный байт равен F7, сдвинув его на 3 вправо в двоичной системе исчисления, увидим, что это байт числа, закодированного четырьмя байтами. 

Он и должен равняться числу 1E.



Функция write\_code\_unit

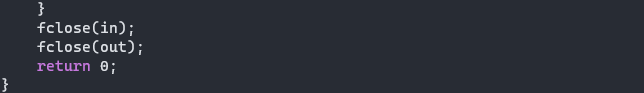
Записывает в файл байты закодированного числа.



Функция encode\_file

Записывает в файл закодированное число.

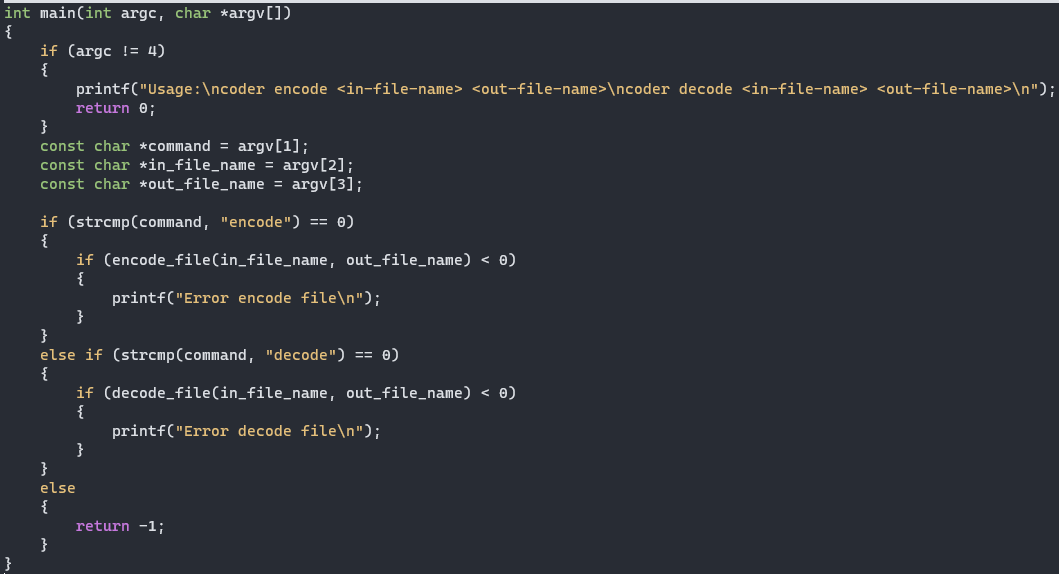
При открытии файлов, проверяем, открылись ли они корректно, после, идем в цикле до конца файла, считываем числа и кодируем их. Если encode отработала ошибочно, выводим ошибку. Если все прошло успешно, записываем в файл out закодированное число.

**

Функция decode\_file

При открытии файлов, проверяем, открылись ли они корректно, после запоминаем конец файла и в цикле идем до конца файла, если функция вернула 0, значит байт найден и функция отработала успешно, в другом случае просто идем к новому байту, декодируем число и записываем его в файл, если при декодировании произошла ошибка, возвращаем ошибку.

Функция main



Если при запуске менее четырех аргументов, программа не отрабатывает и выводит ошибку, если первый аргумент равен encode или decode, то вызываем функция encode\_file, или decode\_file , если функция вернула -1, то выведем ошибку.

# **ПРИЛОЖЕНИЕ**

Исходный код;

**Utf-8.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 | #include <stdio.h>  #include <string.h>  #include "coder.h"  #include "command.h"  **int** **main**(**int** argc, **char** \*argv[])  {  **if** (argc != **4**)  {  printf("Usage:**\n**coder encode <in-file-name> <out-file-name>**\n**coder decode <in-file-name> <out-file-name>**\n**");  **return** **0**;  }  **const** **char** \*command = argv[**1**];  **const** **char** \*in\_file\_name = argv[**2**];  **const** **char** \*out\_file\_name = argv[**3**];  **if** (strcmp(command, "encode") == **0**)  {  **if** (encode\_file(in\_file\_name, out\_file\_name) < **0**)  {  printf("Error encode file**\n**");  }  }  **else** **if** (strcmp(command, "decode") == **0**)  {  **if** (decode\_file(in\_file\_name, out\_file\_name) < **0**)  {  printf("Error decode file**\n**");  }  }  **else**  {  **return** -**1**;  }  } |

**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 | #include <stdio.h>  #include <inttypes.h>  #include "command.h"  #include "coder.h"  **int** **encode\_file**(**const** **char** \*in\_file\_name, **const** **char** \*out\_file\_name)  {  **FILE** \*in;  **FILE** \*out;  in = fopen(in\_file\_name, "r");  **if** (!in)  {  **return** -**1**;  }  out = fopen(out\_file\_name, "wb");  **if** (!out)  {  **return** -**1**;  }  **while** (!feof(in))  {  **uint32\_t** code\_point;  CodeUnit code\_unit;  fscanf(in, "%" SCNx32, &code\_point);  **if** (encode(code\_point, &code\_unit) < **0**)  {  printf("Error encode function**\n**");  **return** -**1**;  }  write\_code\_unit(out, &code\_unit);  }  fclose(in);  fclose(out);  **return** **0**;  }  **int** **decode\_file**(**const** **char** \*in\_file\_name, **const** **char** \*out\_file\_name)  {  **FILE** \*in;  **FILE** \*out;  in = fopen(in\_file\_name, "r");  **if** (!in)  {  **return** -**1**;  }  out = fopen(out\_file\_name, "wb");  **if** (!out)  {  **return** -**1**;  }  CodeUnit code\_unit;  fseek(in, **0**, SEEK\_SET);  fseek(in, **0**, SEEK\_END);  **long** end\_symbol = ftell(in);  fseek(in, **0**, SEEK\_SET);  **while** ((ftell(in)) != (end\_symbol))  {  **if** ((read\_next\_code\_unit(in, &code\_unit)) == **0**)  {  **uint32\_t** num = decode(&code\_unit);  **if** (decode(&code\_unit) < **0**)  {  printf("Error decode function**\n**");  **return** -**1**;  }  fprintf(out, "%" PRIx32 "**\n**", num);  }  }  fclose(in);  fclose(out);  **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  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 | #include <stdio.h>  #include <inttypes.h>  #include "coder.h"  #include "command.h"  **int** **encode**(**uint32\_t** code\_point, CodeUnit \*code\_unit)  {  **if** (code\_point < **0x80**)  {  code\_unit->length = **1**;  code\_unit->code[**0**] = code\_point;  }  **else** **if** (code\_point < **0x800**)  {  code\_unit->length = **2**;  code\_unit->code[**0**] = **0xc0** | (code\_point >> **6**);  code\_unit->code[**1**] = **0x80** | (code\_point & **0x3f**);  }  **else** **if** (code\_point < **0x10000**)  {  code\_unit->length = **3**;  code\_unit->code[**0**] = **0xe0** | (code\_point >> **12**);  code\_unit->code[**1**] = **0x80** | ((code\_point >> **6**) & **0x3f**);  code\_unit->code[**2**] = **0x80** | (code\_point & **0x3f**);  }  **else** **if** (code\_point < **0x200000**)  {  code\_unit->length = **4**;  code\_unit->code[**0**] = **0xf0** | (code\_point >> **18**);  code\_unit->code[**1**] = **0x80** | ((code\_point >> **12**) & **0x3f**);  code\_unit->code[**2**] = **0x80** | ((code\_point >> **6**) & **0x3f**);  code\_unit->code[**3**] = **0x80** | (code\_point & **0x3f**);  }  **else**  {  **return** -**1**;  }  **return** **0**;  }  **uint32\_t** **decode**(**const** CodeUnit \*code\_unit)  {  **uint32\_t** number;  **if** ((code\_unit->length) == **1**)  {  number = code\_unit->code[**0**];  **return** number;  }  **else** **if** ((code\_unit->length) == **2**)  {  number = (((code\_unit->code[**0**]) & **0x1f**) << **6**) | ((code\_unit->code[**1**]) & **0x3f**);  **return** number;  }  **else** **if** ((code\_unit->length) == **3**)  {  number = ((((code\_unit->code[**0**]) & **0xf**) << **12**) | (((code\_unit->code[**1**]) & **0x3f**) << **6**) | ((code\_unit->code[**2**]) & **0x3f**));  **return** number;  }  **else** **if** (code\_unit->length == **4**)  {  number = (((((code\_unit->code[**0**]) & **0x7**) << **18**) | (((code\_unit->code[**1**]) & **0x3f**) << **12**) | ((code\_unit->code[**2**]) & **0x3f**) << **6**) | ((code\_unit->code[**3**]) & **0x3f**));  **return** number;  }  **return** **0**;  }  **int** **read\_next\_code\_unit**(**FILE** \*in, CodeUnit \*code\_unit)  {  **uint8\_t** buf = **0**;  fread(&buf, **sizeof**(**uint8\_t**), **1**, in);  **if** ((buf >> **7**) == **0**)  {  code\_unit->code[**0**] = buf;  code\_unit->length = **1**;  }  **else** **if** ((buf >> **5**) == **0x06**)  {  code\_unit->code[**0**] = buf;  fread(&buf, **sizeof**(**uint8\_t**), **1**, in);  code\_unit->code[**1**] = buf;  code\_unit->length = **2**;  }  **else** **if** ((buf >> **4**) == **0x0e**)  {  code\_unit->code[**0**] = buf;  fread(&buf, **sizeof**(**uint8\_t**), **1**, in);  code\_unit->code[**1**] = buf;  fread(&buf, **sizeof**(**uint8\_t**), **1**, in);  code\_unit->code[**2**] = buf;  code\_unit->length = **3**;  }  **else** **if** ((buf >> **3**) == **0x1e**)  {  code\_unit->code[**0**] = buf;  fread(&buf, **sizeof**(**uint8\_t**), **1**, in);  code\_unit->code[**1**] = buf;  fread(&buf, **sizeof**(**uint8\_t**), **1**, in);  code\_unit->code[**2**] = buf;  fread(&buf, **sizeof**(**uint8\_t**), **1**, in);  code\_unit->code[**3**] = buf;  code\_unit->length = **4**;  }  **return** **0**;  }  **int** **write\_code\_unit**(**FILE** \*out, **const** CodeUnit \*code\_unit)  {  **int** result = fwrite(code\_unit->code, **1**, code\_unit->length, out);  **return** result;  } |