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-221  «2» мая 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | Гордов Р.С. |
|  |  |  |
| Проверил:  Старший преподаватель кафедры ВС  «3» мая 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | Фульман В.О. |

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_»

Новосибирск 2023

**ОГЛАВЛЕНИЕ**

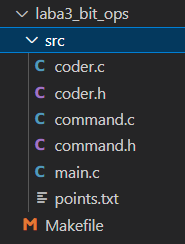
[**ЗАДАНИЕ** 3](#_Toc133940450)

[**ВЫПОЛНЕНИЕ РАБОТЫ** 4](#_Toc133940451)

[**ПРИЛОЖЕНИЕ** 12](#_Toc133940452)

# **ЗАДАНИЕ**

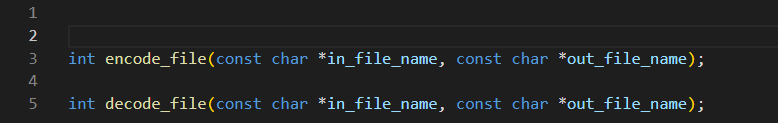
Разработать приложение для кодирования и декодирования чисел в формате utf-8. Со следующей структурой проекта.

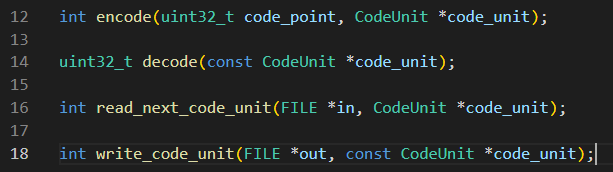


**ВЫПОЛНЕНИЕ РАБОТЫ**

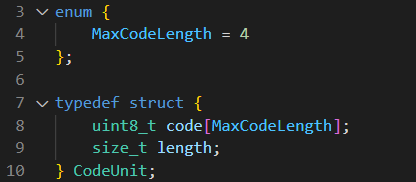
Файлы с расширением .h будут содержать прототипы функций.

command.h:

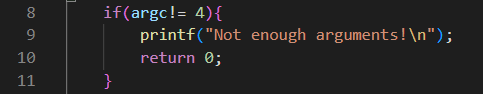
coder.h:

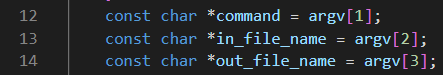


Файл coder.h, также будет содержать структуру кодируемой единицы:



В файле main.c будем проверять количество аргументов.



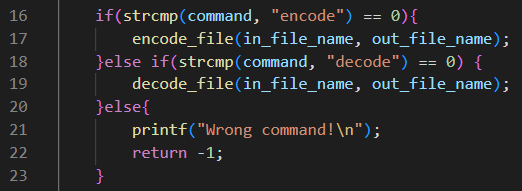


Первый аргумент – команда, которую будет выполнять приложение.

Второй аргумент - входной файл (кодируемый или декодируемый).

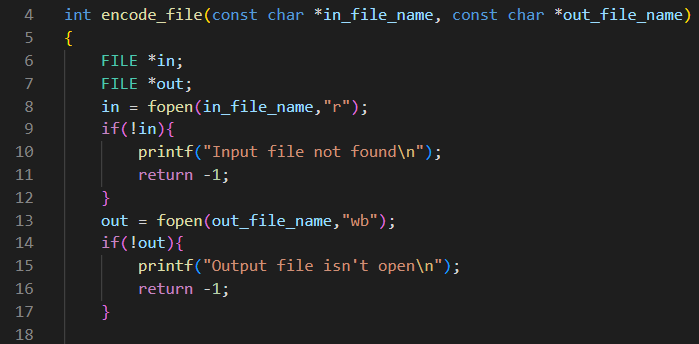
Третий аргумент - выходной файл.

Далее лексикографически проверяем команды и вызываем соответствующую функцию.

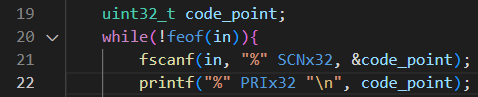


Функции encode\_file и decode\_file будут содержаться в файле command.c

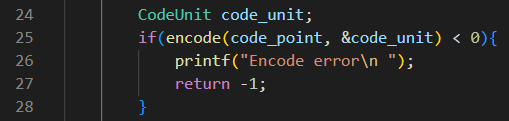
Функция encode\_file открывает оба файла, первый – для чтения, второй – для бинарной записи.



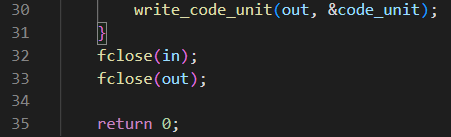
Построчно, до конца файла считываем шестнадцатеричные числа и записываем в 4х байтовую переменную.



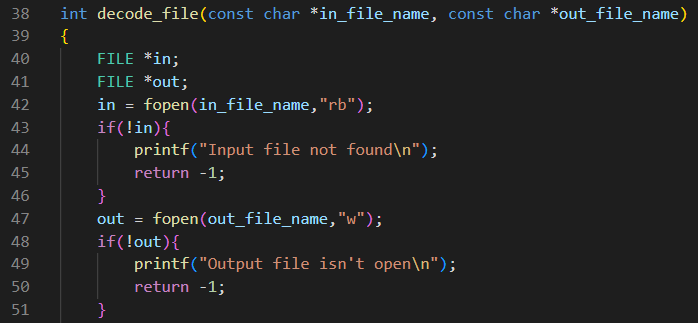
Создаем экземпляр структуры и вызываем функцию encode, которая будет записывать закодированное число в структуру.



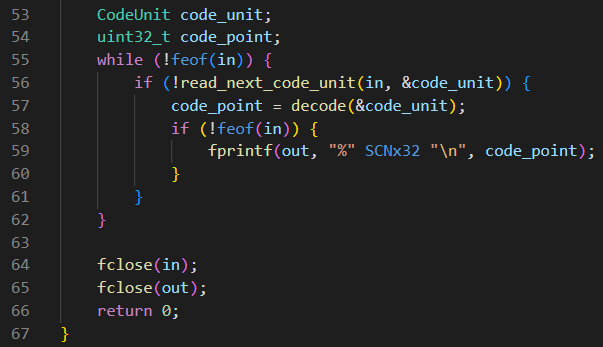
Далее записываем закодированную единицу в выходной файл.



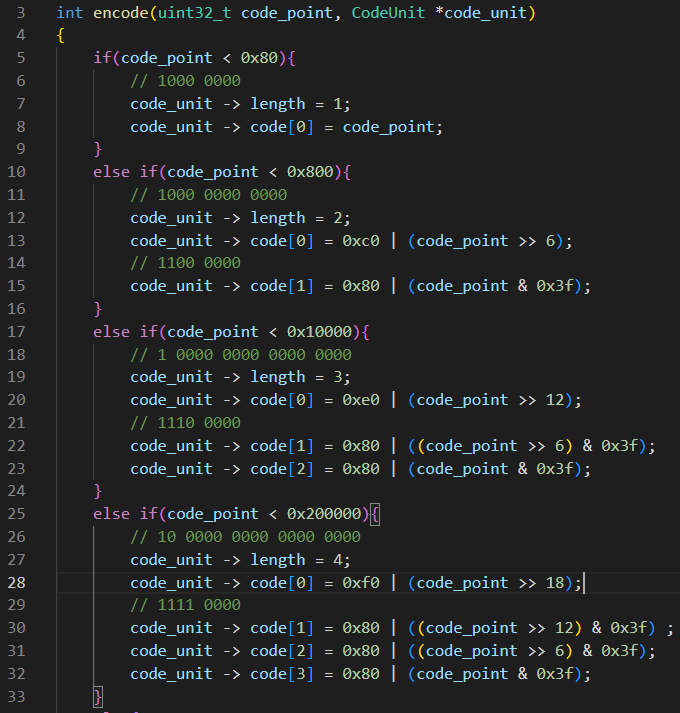
Функция decode\_file открывает оба файла, первый – для бинарного чтения, второй – для записи.



Она будет вызывать функцию считывания единицы кодирования и декодирования и записывать декодированное значение.



Посмотрим, как выполняется кодирование.



Функция encode принимает 4х байтовую переменную и структуру единицы кодирования, в которую мы будем записывать сколько байт нам нужно для кодирования и само закодированное число.

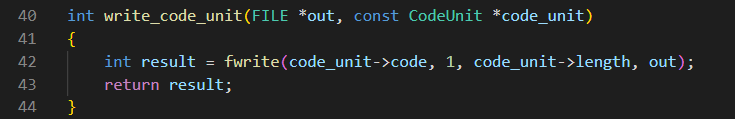
Первый байт будет содержать информацию о количестве байт, необходимых для записи числа, а последующие будут начинаться на 10.

Например, если входное значение можно записать в 21 байт (оно меньше 0x2000000), то записываем в длину 4.

Первый байт содержит будет содержать 4 единицы. И 4 значащих бита, которые идут сначала нашего значения, а значит нам побитно сдвинуть на 18. С помощью побитного ИЛИ соединяем в один байт.

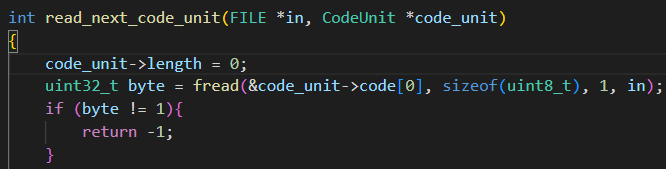
Последующие байты будут начинаться на 10. Будем каждый раз сдвигать на 6 и примерять маску 00 111111 (0x3f).

Из массива code записываем байты length раз в выходной файл.

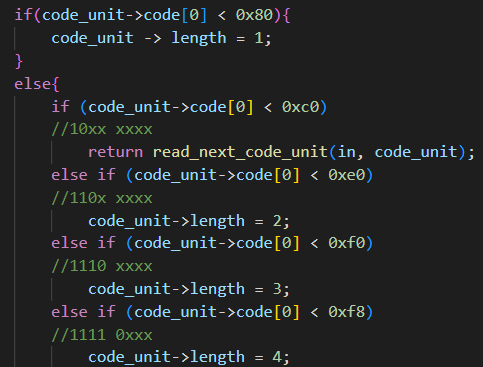


Посмотрим, как выполняется декодирование.

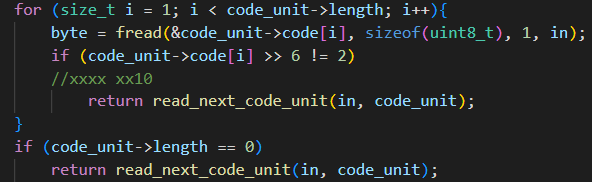
Считываем байт.



По нему определяем количество последующих байт.



Считываем последующие и проверяем начинаются ли они на 10.



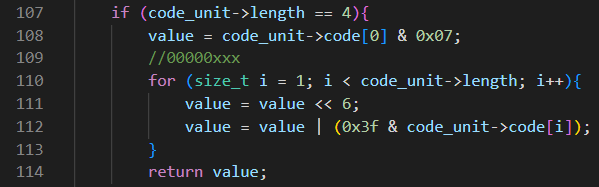
Если байт не начинается с единиц, длина будет нулевой, и мы продолжим считывать следующие байты.

Как происходит декодирование.

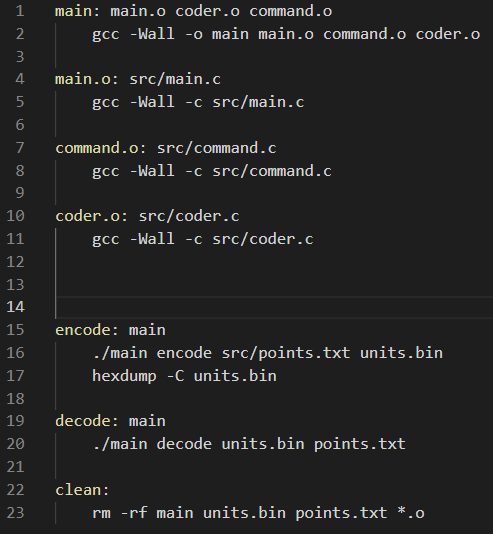


Принимаем единицу кодирования.

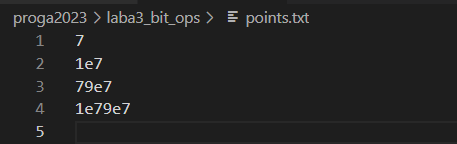
Если число было записано в 4 байта, на первый байт надеваем маску 00000111, записываем 3 информационных бита, последующие байты записываем побитно сдвинув наше декодированное значение на 6, надеваем маску и объединяем с помощью побитного ИЛИ.



Создадим Makefile для автоматической сборки.



Входной файл:



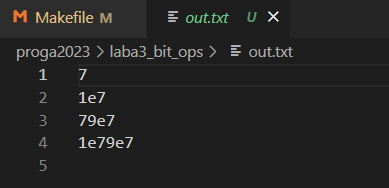
Кодируем.





Декодируем.





# **ПРИЛОЖЕНИЕ**

<https://github.com/Herroa/proga2023>

**Makefile**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | **main:** main.o coder.o command.o  gcc -Wall -o main main.o command.o coder.o  main.o: src/main.c  gcc -Wall -c src/main.c  command.o: src/command.c  gcc -Wall -c src/command.c  coder.o: src/coder.c  gcc -Wall -c src/coder.c  **encode:** main  ./main encode src/points.txt units.bin  hexdump -C units.bin  **decode:** main  ./main decode units.bin points.txt  **clean:**  rm -rf main units.bin points.txt \*.o |

**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 | #include <stdio.h>  #include <string.h>  #include "coder.h"  #include "command.h"  **int** **main**(**int** argc, **char** \*argv[])  {  **if**(argc!= **4**){  printf("Not enough arguments!**\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**){  encode\_file(in\_file\_name, out\_file\_name);  }**else** **if**(strcmp(command, "decode") == **0**) {  decode\_file(in\_file\_name, out\_file\_name);  }**else**{  printf("Wrong command!**\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  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 | #include "coder.h"  **int** **encode**(**uint32\_t** code\_point, CodeUnit \*code\_unit)  {  **if**(code\_point < **0x80**){  // 1000 0000  code\_unit -> length = **1**;  code\_unit -> code[**0**] = code\_point;  }  **else** **if**(code\_point < **0x800**){  // 1000 0000 0000  code\_unit -> length = **2**;  code\_unit -> code[**0**] = **0xc0** | (code\_point >> **6**);  // 1100 0000  code\_unit -> code[**1**] = **0x80** | (code\_point & **0x3f**);  }  **else** **if**(code\_point < **0x10000**){  // 1 0000 0000 0000 0000  code\_unit -> length = **3**;  code\_unit -> code[**0**] = **0xe0** | (code\_point >> **12**);  // 1110 0000  code\_unit -> code[**1**] = **0x80** | ((code\_point >> **6**) & **0x3f**);  code\_unit -> code[**2**] = **0x80** | (code\_point & **0x3f**);  }  **else** **if**(code\_point < **0x200000**){  // 10 0000 0000 0000 0000  code\_unit -> length = **4**;  code\_unit -> code[**0**] = **0xf0** | (code\_point >> **18**);  // 1111 0000  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**;  }  **int** **write\_code\_unit**(**FILE** \*out, **const** CodeUnit \*code\_unit)  {  **int** result = fwrite(code\_unit->code, **1**, code\_unit->length, out);  **return** result;  }  **int** **read\_next\_code\_unit**(**FILE** \*in, CodeUnit \*code\_unit)  {  code\_unit->length = **0**;  **size\_t** byte = fread(&code\_unit->code[**0**], **sizeof**(**uint8\_t**), **1**, in);  **if** (byte != **1**){  **return** -**1**;  }  **if**(code\_unit->code[**0**] < **0x80**){  code\_unit -> length = **1**;  }  **else**{  **if** (code\_unit->code[**0**] < **0xc0**)  //10xx xxxx  **return** read\_next\_code\_unit(in, code\_unit);  **else** **if** (code\_unit->code[**0**] < **0xe0**)  //110x xxxx  code\_unit->length = **2**;  **else** **if** (code\_unit->code[**0**] < **0xf0**)  //1110 xxxx  code\_unit->length = **3**;  **else** **if** (code\_unit->code[**0**] < **0xf8**)  //1111 0xxx  code\_unit->length = **4**;  **for** (**size\_t** i = **1**; i < code\_unit->length; i++){  byte = fread(&code\_unit->code[i], **sizeof**(**uint8\_t**), **1**, in);  **if** (code\_unit->code[i] >> **6** != **2**)  //xxxx xx10  **return** read\_next\_code\_unit(in, code\_unit);  }  **if** (code\_unit->length == **0**)  **return** read\_next\_code\_unit(in, code\_unit);  }  **return** **0**;  }  **uint32\_t** **decode**(**const** CodeUnit \*code\_unit)  {  **uint32\_t** value = **0**;  **if** (code\_unit->length == **1**){  value = code\_unit->code[**0**];  **return** value;  }  **if** (code\_unit->length == **2**){  value = code\_unit->code[**0**] & **0x1f**;  // 000x xxxx  value = value << **6**;  value = value | (code\_unit->code[**1**] & **0x3f**);  // 00xx xxxx  **return** value;  }  **if** (code\_unit->length == **3**){  value = code\_unit->code[**0**] & **0x0f**;  //0000xxxx  **for** (**size\_t** i = **1**; i < code\_unit->length; i++){  value = value << **6**;  value = value | (**0x3f** & code\_unit->code[i]);  //00xx xxxx  }  **return** value;  }  **if** (code\_unit->length == **4**){  value = code\_unit->code[**0**] & **0x07**;  //00000xxx  **for** (**size\_t** i = **1**; i < code\_unit->length; i++){  value = value << **6**;  value = value | (**0x3f** & code\_unit->code[i]);  }  **return** value;  }  **return** **0**;  } |

**coder.h**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | #include <stdio.h>  #include <inttypes.h>  **enum** {  MaxCodeLength = **4**  };  **typedef** **struct** {  **uint8\_t** code[MaxCodeLength];  **size\_t** length;  } CodeUnit;  **int** **encode**(**uint32\_t** code\_point, CodeUnit \*code\_unit);  **uint32\_t** **decode**(**const** CodeUnit \*code\_unit);  **int** **read\_next\_code\_unit**(**FILE** \*in, CodeUnit \*code\_unit);  **int** **write\_code\_unit**(**FILE** \*out, **const** CodeUnit \*code\_unit); |

**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 | #include "coder.h"  #include "command.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){  printf("Input file not found**\n**");  **return** -**1**;  }  out = fopen(out\_file\_name,"wb");  **if**(!out){  printf("Output file isn't open**\n**");  **return** -**1**;  }  **uint32\_t** code\_point;  **while**(!feof(in)){  fscanf(in, "%" SCNx32, &code\_point);  printf("%" PRIx32 "**\n**", code\_point);  CodeUnit code\_unit;  **if**(encode(code\_point, &code\_unit) < **0**){  printf("Encode error**\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,"rb");  **if**(!in){  printf("Input file not found**\n**");  **return** -**1**;  }  out = fopen(out\_file\_name,"w");  **if**(!out){  printf("Output file isn't open**\n**");  **return** -**1**;  }  CodeUnit code\_unit;  **uint32\_t** code\_point;  **while** (!feof(in)) {  **if** (!read\_next\_code\_unit(in, &code\_unit)) {  code\_point = decode(&code\_unit);  **if** (!feof(in)) {  fprintf(out, "%" SCNx32 "**\n**", code\_point);  }  }  }  fclose(in);  fclose(out);  **return** **0**;  } |

**command.h**

|  |  |
| --- | --- |
| 1  2  3 | **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); |