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-241  «16» мая 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | /Бондаренко А.А./ |
|  |  |  |
| Проверил:  ст. преп. кафедры ВС  «\_\_» мая 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | /Фульман В.О./ |

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_»

Новосибирск 2023

**ОГЛАВЛЕНИЕ**

[ЗАДАНИЕ 3](#_Toc135175151)

[Задание 1 3](#_Toc135175152)

[Задание 2 4](#_Toc135175153)

[ВЫПОЛНЕНИЕ РАБОТЫ 5](#_Toc135175154)

[Задание 1 5](#_Toc135175155)

[Задание 2 6](#_Toc135175156)

[ПРИЛОЖЕНИЕ 10](#_Toc135175157)

[Приложение 1 10](#_Toc135175158)

[Приложение 2 12](#_Toc135175159)

# **ЗАДАНИЕ**

В первой части вам предлагается разобрать алгоритм кодирования целых чисел, называемый varint (variable integer). Такой способ кодирования позволяет использовать переменное количество байт для представления целых чисел и благодаря этому обеспечивает компактность данных. Вам нужно разработать приложение для записи и чтения чисел в сыром виде и в формате varint, сравнить эти два способа по эффективности и сделать выводы о применимости предложенного способа кодирования.

Во второй части необходимо самостоятельно реализовать алгоритм кодирования UTF-8. Этот алгоритм решает аналогичную задачу — позволяет кодировать целые числа переменным количеством байт, но используется для кодирования кодов символов и поддерживает обратную совместимость с кодировкой ASCII.

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

В работе мы будем 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  30  31  32  33 | *#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**;**  **}** |

## Задание 2

Разработать приложение для кодирования и декодирования чисел по описанному выше алгоритму.

Предлагается следующая структура проекта:

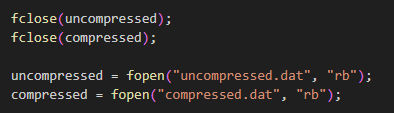
|  |
| --- |
| **.**  **|-- Makefile**  **`-- src**  **|-- coder.c**  **|-- coder.h**  **|-- command.c**  **|-- command.h**  **`-- main.c`** |

**ВЫПОЛНЕНИЕ РАБОТЫ**

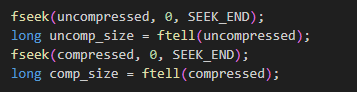
## Задание 1

Приложение random генерирует 1000000 случайных чисел, записывает их в незакодированном виде и в закодированном в два разных бинарных файла.

После файлы закрываются и открываются снова, но уже в режиме чтения, а не записи:



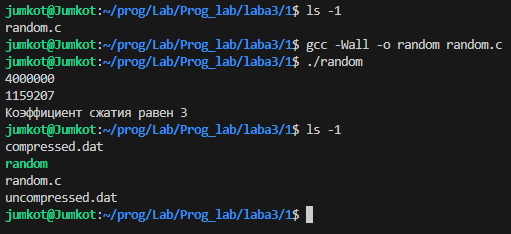
С помощью перемещения указателя внутри файла функция ftell подсчитывает общий размер обоих файлов, чтобы вычислить коэффициент сжатия:



Затем указатели внутри файлов перемещаются к началу файлов, и, используя функцию fread, каждое число из закодированного файла декодируется и сравнивается с изначальным, не кодированным числом. Так реализуется проверка на то, что кодирование прошло корректно и результаты обратной работы совпадают.

Кодирование, декодирование и генерация случайных чисел реализованы с помощью функций, приведённых в Задании 1 (раздел Задание).

Пример результата работы программы:

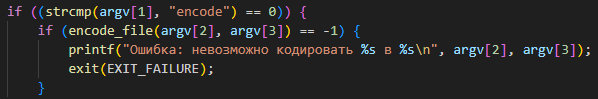


## Задание 2

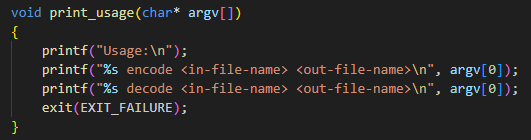
int main(int argc, char\* argv[])

Функция main служит головным “переключателем” в программе: с помощью параметров запуска она вызывает одну из “консольных” функций (argv[1]) и передаёт им имена входных (argv[2]) и выходных (argv[3]) файлов.

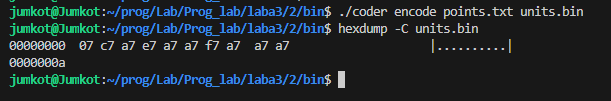
В случае неправильного написания команды программа возвращает -1 и завершает выполнение с ненулевым кодом с помощью функции exit(). Проверка соответствия написания производится с помощью функции strcmp():



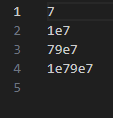
При получении неверного количества аргументов программа выдаёт ошибку по приведённому в лабораторной работе формату и завершает выполнение тем же способом.



Примеры результатов работы программы (при приведённых в примере входных данных):

****





results.txt

int encode(uint32\_t code\_point, CodeUnits\* code\_unit)

В функции encode сначала производится проверка корректности числа – если переданное макросу assert выражение false, то он выведет ошибку в stderr (стандартный поток вывода ошибок).

Далее следует конструкция из if-ов:

1. если полученное число меньше 10000000 в двоичной системе – 7 значащих бит

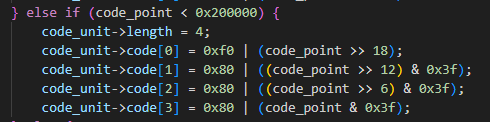
Тогда фактическая длина массива будет 1, а число целиком без преобразований переносится в нулевой по индексу элемент массива.

1. если полученное число меньше 100000 000000 в двоичной системе – 11 значащих бит (т.к. если число кодируется двумя байтами и более, то значащих бит остаётся не более 6 вместо 8 в каждой части)

В таком случае длина будет уже 2. После в нулевой элемент массива заносится исходное число, битово умноженное на 0x0с (110xxxxx) для обозначения количества последующий байт, с происходит сдвиг на 6 бит вправо. Следующий элемент битово умножается на 0x3f (00111111) для преобразования, и сложение с 0x80 для добавления 10 в начале.

1. если полученное число меньше 10000 000000 000000 в двоичной системе – 16 значащих бит
2. если полученное число меньше 1000 000000 000000 000000 в двоичной системе – 21 значащий бит

В третьем и четвёртом else if длина массива будет 3 и 4 соответственно, сдвиг будет на 12 и на 18 бит соответственно (с уменьшением на 6 на каждом шаге вплоть до последнего), а сложение первого байта будет происходить соответственно с 1110xxxx и 11110xxx.



uint32\_t decode(const CodeUnits\* code\_unit)

В функции decode происходит декодирование исходя из длины массива:

1. Длина 1

Число переносится без изменений (нет служебных битов).

1. Длина 2

Первый элемент переносится в массив с умножением на 00011111 (три служебных бита), а каждый последующий байт переносится со сдвигом влево на 6 и умножением на 0x3f (00111111) (потеря служебной 10 в начале).



1. Длина 3
2. Длина 4

При длинах 3 и 4 все шаги такие же, только умножение первого байта происходит на 00001111 и на 00000111 соответственно, потому что больше служебных бит.

int write\_code\_unit(FILE\* out, const CodeUnits\* code\_unit)

Функция производит запись закодированных по предложенному шаблону данных в бинарный файл.

В функции write\_code\_unit производится проверка, что ни один из указателей не NULL и длина массива с байтами не равна нуля. Если хотя бы одно из условий не выполняется, функция вернёт -1.

Если проверки прошли успешно, то с помощью fwrite в выходной файл записывается содержимое изначального массива в соответствии с его длиной. В случае, если fwrite запишет меньше, чем переданное ей code\_unit->length, то функция также вернёт -1.

int read\_next\_code\_unit(FILE\* in, CodeUnits\* code\_unit)

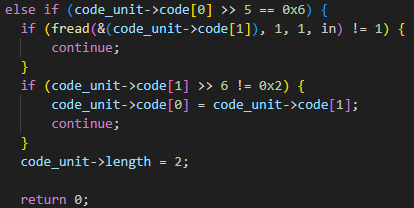
Функция производит чтение закодированных по предложенному шаблону данных из бинарного файла.

Функция read\_next\_code\_unit сначала проверяет, не является ли указатель на файл нулевым. Также проверяется, чтобы fread возвращала указанное при вызове количество объектов (один), и если это не так, достигнут ли конец файла – тогда возвращается -1. Вместе с этим производится занесение кодированного числа из файла в массив.

Далее идёт бесконечный цикл while. Первый if проверяет, является ли первый элемент и последним (т.е. отсутствуют служебные единицы) с помощью умножения на 0x80 (10000000). Если да, то длина массива обозначается 1, и работа функции завершается.

После блоками из else if производится определение длины массива (количества байт в кодированном числе) посредством битового сдвига нулевого элемента и сравнения с соответствующими шестнадцатеричными числами:

1. При двух байтах сдвиг на пять битов и сравнение с 0x6 (0110)
2. При трёх байтах сдвиг на четыре бита и сравнение с 0xe (1110)
3. При четырёх байтах сдвиг на три бита и сравнение с 0x1e (11110)



Если в процессе чтения происходит ошибка (если следующий байт битый и не соответствует формату), функция пропускает его и продолжает чтение следующего байта, пока не будет найден корректный элемент, переходя к следующему шагу цикла с помощью оператора continue. Тогда длина массива устанавливается в соответствии с else if, в который вошла функция, и функция возвращает 0.

Когда программа дойдёт до конца любого из else if, работа функции закончится.

int encode\_file(const char\* in\_file\_name, const char\* out\_file\_name)

“Консольная” функция, которая используется для выбора команды при запуске программы в качестве передаваемого параметра. Отвечает за кодирование текстового файла в бинарный, делая это с помощью последовательного вызова вышеописанных функций.

Пока не будет достигнут конец файла, шестнадцатеричные числа из текстового файла будут по одному считываться и передаваться в функцию encode для кодирования, после – заноситься в результирующий бинарный файл с помощью write\_code\_unit. В случае, если значение внутри входного файла не соответствует формату uint32\_t, программа выдаст ошибку и завершится.

Также до начала непосредственной работы производится проверка на корректное открытие входного и выходного файлов, полученных как аргументы.

int decode\_file(const char\* in\_file\_name, const char\* out\_file\_name)

“Консольная” функция, которая используется для выбора команды при запуске программы в качестве передаваемого параметра. Отвечает за декодирование бинарного файла в бинарный, делая это с помощью последовательного вызова вышеописанных функций.

Пока не будет достигнут конец файла, закодированные по шаблону числа из бинарного файла будут по одному передаваться в функцию decode для декодирования и записываться в выходной текстовый файл с помощью read\_next\_code\_unit.

До начала непосредственной работы производится проверка на корректное открытие входного и выходного файлов, полученных как аргументы.

# **ПРИЛОЖЕНИЕ**

## Приложение 1

random.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 | *#include <assert.h>*  *#include <stddef.h>*  *#include <stdint.h>*  *#include <stdio.h>*  *#include <stdlib.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**;**  **}**  */\**  *\* 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;**  **}**  **int** main**()**  **{**  **FILE\*** uncompressed **=** fopen**(**"uncompressed.dat"**,** "wb"**);**  **FILE\*** compressed **=** fopen**(**"compressed.dat"**,** "wb"**);**  **uint32\_t** number**;**  **uint32\_t** varint\_size**;**  **uint8\_t\*** var\_arr **=** malloc**(sizeof(uint32\_t));**    **for** **(int** i **=** **0;** i **<** **1000000;** i**++)** **{**  number **=** generate\_number**();**  fwrite**(&**number**,** **sizeof(**number**),** **1,** uncompressed**);**  varint\_size **=** encode\_varint**(**number**,** var\_arr**);**  fwrite**(**var\_arr**,** varint\_size**,** **1,** compressed**);**  **}**  fclose**(**uncompressed**);**  fclose**(**compressed**);**  uncompressed **=** fopen**(**"uncompressed.dat"**,** "rb"**);**  compressed **=** fopen**(**"compressed.dat"**,** "rb"**);**  fseek**(**uncompressed**,** **0,** SEEK\_END**);**  **long** uncomp\_size **=** ftell**(**uncompressed**);**  fseek**(**compressed**,** **0,** SEEK\_END**);**  **long** comp\_size **=** ftell**(**compressed**);**  printf**(**"%ld\n"**,** uncomp\_size**);**  printf**(**"%ld\n"**,** comp\_size**);**  printf**(**"Коэффициент сжатия равен %ld\n"**,** **(**uncomp\_size **/** comp\_size**));**  fseek**(**uncompressed**,** **0,** SEEK\_SET**);**  fseek**(**compressed**,** **0,** SEEK\_SET**);**  **uint32\_t** decode**;**  var\_arr **=** realloc**(**var\_arr**,** comp\_size**);**  **const** **uint8\_t\*** buf **=** var\_arr**;**  fread**(**var\_arr**,** **sizeof(uint8\_t),** comp\_size**,** compressed**);**  **for** **(int** i **=** **0;** i **<** **1000000;** i**++)** **{**  fread**(&**number**,** **sizeof(uint32\_t),** **1,** uncompressed**);**  decode **=** decode\_varint**(&**buf**);**  **if** **(**number **!=** decode**)** **{**  printf**(**"Не удалось произвести кодирование или декодирование (%d)\n"**,** i**);**  **return** **1;**  **}**  **}**  free**(**var\_arr**);**  fclose**(**uncompressed**);**  fclose**(**compressed**);**  **return** **0;**  **}** |

## Приложение 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  37  38 | *#include <stdint.h>*  *#include <stdlib.h>*  *#include <string.h>*  *#include "coder.h"*  *#include "command.h"*  **void** print\_usage**(char\*** argv**[]);**  **int** main**(int** argc**,** **char\*** argv**[])**  **{**  **if** **(**argc **!=** **4)**  print\_usage**(**argv**);**  **if** **((**strcmp**(**argv**[1],** "encode"**)** **==** **0))** **{**  **if** **(**encode\_file**(**argv**[2],** argv**[3])** **==** **-1)** **{**  printf**(**"Ошибка: невозможно кодировать %s в %s\n"**,** argv**[2],** argv**[3]);**  exit**(**EXIT\_FAILURE**);**  **}**  **}** **else** **if** **(**strcmp**(**argv**[1],** "decode"**)** **==** **0)** **{**  **if** **(**decode\_file**(**argv**[2],** argv**[3])** **==** **-1)** **{**  printf**(**"Ошибка: невозможно декодировать %s в %s\n"**,** argv**[2],** argv**[3]);**  exit**(**EXIT\_FAILURE**);**  **}**  **}** **else** **{**  print\_usage**(**argv**);**  **}**  **return** **0;**  **}**  **void** print\_usage**(char\*** argv**[])**  **{**  printf**(**"Usage:\n"**);**  printf**(**"%s encode <in-file-name> <out-file-name>\n"**,** argv**[0]);**  printf**(**"%s decode <in-file-name> <out-file-name>\n"**,** argv**[0]);**  exit**(**EXIT\_FAILURE**);**  **}** |

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  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151 | *#include <assert.h>*  *#include "coder.h"*  *#include "command.h"*  **int** encode**(uint32\_t** code\_point**,** CodeUnits**\*** code\_unit**)**  **{**  assert**(**code\_unit **!=** NULL**);**  **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** **{**  printf**(**"Ошибка: %d превышает максимальное значение для кодирования\n"**,** code\_point**);**  code\_unit**->**length **=** **0;**  **return** **-1;**  **}**  **return** **0;**  **}**  **uint32\_t** decode**(const** CodeUnits**\*** code\_unit**)**  **{**  **uint32\_t** code\_point**;**  **if** **(**code\_unit**->**length **==** **1)** **{**  code\_point **=** code\_unit**->**code**[0];**  **}** **else** **if** **(**code\_unit**->**length **==** **2)** **{**  code\_point **=** code\_unit**->**code**[0]** **&** **0x1f;**  code\_point **=** **(**code\_point **<<** **6)** **|** **(**code\_unit**->**code**[1]** **&** **0x3f);**  **}** **else** **if** **(**code\_unit**->**length **==** **3)** **{**  code\_point **=** code\_unit**->**code**[0]** **&** **0xf;**  code\_point **=** **(**code\_point **<<** **6)** **|** **(**code\_unit**->**code**[1]** **&** **0x3f);**  code\_point **=** **(**code\_point **<<** **6)** **|** **(**code\_unit**->**code**[2]** **&** **0x3f);**  **}** **else** **if** **(**code\_unit**->**length **==** **4)** **{**  code\_point **=** code\_unit**->**code**[0]** **&** **0x7;**  code\_point **=** **(**code\_point **<<** **6)** **|** **(**code\_unit**->**code**[1]** **&** **0x3f);**  code\_point **=** **(**code\_point **<<** **6)** **|** **(**code\_unit**->**code**[2]** **&** **0x3f);**  code\_point **=** **(**code\_point **<<** **6)** **|** **(**code\_unit**->**code**[3]** **&** **0x3f);**  code\_unit **=** NULL**;**  **}**  **return** code\_point**;**  **}**  **int** write\_code\_unit**(FILE\*** out**,** **const** CodeUnits**\*** code\_unit**)**  **{**  **if** **((**out **==** NULL**)** **||** **(**code\_unit **==** NULL**)** **||** **(**code\_unit**->**length **==** **0))** **{**  **return** **-1;**  **}**  **if** **(**fwrite**(**code\_unit**->**code**,** **1,** code\_unit**->**length**,** out**)** **<** code\_unit**->**length**)** **{**  **return** **-1;**  **}**  **return** **0;**  **}**  **int** read\_next\_code\_unit**(FILE\*** in**,** CodeUnits**\*** code\_unit**)**  **{**  **if** **(**in **==** NULL**)** **{**  **return** **-1;**  **}**  **if** **(**fread**(**code\_unit**->**code**,** **1,** **1,** in**)** **!=** **1)** **{**  **if** **(**feof**(**in**))** **{**  **return** **-1;**  **}**  **}**  **while** **(1)** **{**  **if** **(**feof**(**in**))** **{**  code\_unit **=** NULL**;**  **return** **-1;**  **}**  **if** **((**code\_unit**->**code**[0]** **&** **0x80)** **==** **0)** **{**  code\_unit**->**length **=** **1;**  **return** **0;**  **}** **else** **if** **(**code\_unit**->**code**[0]** **>>** **5** **==** **0x6)** **{**  **if** **(**fread**(&(**code\_unit**->**code**[1]),** **1,** **1,** in**)** **!=** **1)** **{**  **continue;**  **}**  **if** **(**code\_unit**->**code**[1]** **>>** **6** **!=** **0x2)** **{**  code\_unit**->**code**[0]** **=** code\_unit**->**code**[1];**  **continue;**  **}**  code\_unit**->**length **=** **2;**  **return** **0;**  **}** **else** **if** **(**code\_unit**->**code**[0]** **>>** **4** **==** **0xe)** **{**  **if** **(**fread**(&(**code\_unit**->**code**[1]),** **1,** **1,** in**)** **!=** **1)** **{**  **continue;**  **}**  **if** **(**code\_unit**->**code**[1]** **>>** **6** **!=** **0x2)** **{**  code\_unit**->**code**[0]** **=** code\_unit**->**code**[1];**  **continue;**  **}**  **if** **(**fread**(&(**code\_unit**->**code**[2]),** **1,** **1,** in**)** **!=** **1)** **{**  **continue;**  **}**  **if** **(**code\_unit**->**code**[2]** **>>** **6** **!=** **0x2)** **{**  code\_unit**->**code**[0]** **=** code\_unit**->**code**[2];**  **continue;**  **}**  code\_unit**->**length **=** **3;**  **return** **0;**  **}** **else** **if** **(**code\_unit**->**code**[0]** **>>** **3** **==** **0x1e)** **{**  **if** **(**fread**(&(**code\_unit**->**code**[1]),** **1,** **1,** in**)** **!=** **1)** **{**  **continue;**  **}**  **if** **(**code\_unit**->**code**[1]** **>>** **6** **!=** **0x2)** **{**  code\_unit**->**code**[0]** **=** code\_unit**->**code**[1];**  **continue;**  **}**  **if** **(**fread**(&(**code\_unit**->**code**[2]),** **1,** **1,** in**)** **!=** **1)** **{**  **continue;**  **}**  **if** **(**code\_unit**->**code**[2]** **>>** **6** **!=** **0x2)** **{**  code\_unit**->**code**[0]** **=** code\_unit**->**code**[2];**  **continue;**  **}**  **if** **(**fread**(&(**code\_unit**->**code**[3]),** **1,** **1,** in**)** **!=** **1)** **{**  **continue;**  **}**  **if** **(**code\_unit**->**code**[3]** **>>** **6** **!=** **0x2)** **{**  code\_unit**->**code**[0]** **=** code\_unit**->**code**[3];**  **continue;**  **}**  code\_unit**->**length **=** **4;**  **return** **0;**  **}**  fread**(**code\_unit**->**code**,** **1,** **1,** in**);**  **}**  **}** |

coder.h

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | *#pragma once*  *#include <stdint.h>*  *#include <stdio.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** write\_code\_unit**(FILE\*** out**,** **const** CodeUnits**\*** code\_unit**);**  **int** read\_next\_code\_unit**(FILE\*** in**,** CodeUnits**\*** code\_units**);** |

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 | *#include "coder.h"*  *#include "command.h"*  **int** encode\_file**(const** **char\*** in\_file\_name**,** **const** **char\*** out\_file\_name**)**  **{**  **FILE** **\***in**,** **\***out**;**  in **=** fopen**(**in\_file\_name**,** "r"**);**  **if** **(!**in**)** **{**  printf**(**"Ошибка: не удалось открыть файл %s\n"**,** in\_file\_name**);**  **return** **-1;**  **}**  out **=** fopen**(**out\_file\_name**,** "wb"**);**  **if** **(!**out**)** **{**  printf**(**"Ошибка: не удалось открыть файл %s\n"**,** out\_file\_name**);**  fclose**(**in**);**  **return** **-1;**  **}**  **uint32\_t** value**;**  CodeUnits enc\_units**;**  **while** **(!**feof**(**in**))** **{**  **if** **(**fscanf**(**in**,** "%" SCNx32**,** **&**value**)** **==** **0)** **{**  printf**(**"Ошибка: элемент '%d'в файле '%s' не является значением типа uint32\_t\n"**,** value**,** in\_file\_name**);**  **return** **-1;**  **}**  encode**(**value**,** **&**enc\_units**);**  write\_code\_unit**(**out**,** **&**enc\_units**);**  **}**  fclose**(**in**);**  fclose**(**out**);**  **return** **0;**  **}**  **int** decode\_file**(const** **char\*** in\_file\_name**,** **const** **char\*** out\_file\_name**)**  **{**  **FILE** **\***in**,** **\***out**;**  in **=** fopen**(**in\_file\_name**,** "rb"**);**  **if** **(!**in**)** **{**  printf**(**"Ошибка: не удалось открыть файл %s\n"**,** in\_file\_name**);**  **return** **-1;**  **}**  out **=** fopen**(**out\_file\_name**,** "w"**);**  **if** **(!**out**)** **{**  printf**(**"Ошибка: не удалось открыть файл %s\n"**,** out\_file\_name**);**  fclose**(**in**);**  **return** **-1;**  **}**  **uint32\_t** value**;**  CodeUnits dec\_units**;**  **while** **(!**feof**(**in**))** **{**  **if** **(**read\_next\_code\_unit**(**in**,** **&**dec\_units**)** **==** EOF**)** **{**  **break;**  **}**  value **=** decode**(&**dec\_units**);**  fprintf**(**out**,** "%" PRIx32**,** value**);**  fprintf**(**out**,** "\n"**);**  **}**  fclose**(**in**);**  fclose**(**out**);**  **return** **0;**  **}** |

command.h

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | *#pragma once*  *#include <inttypes.h>*  *#include "coder.h"*  **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**);** |