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

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:

```
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:

```
/*
 * 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:

```
30  size_t encode_varint(uint32_t value, uint8_t* buf)
31  {
32      assert(buf != NULL);
33      uint8_t* cur = buf;
34      while (value >= 0x80) {
35          const uint8_t byte = (value & 0x7f) | 0x80;
36          *cur = byte;
37          value >>= 7;
38          ++cur;
39      }
40      *cur = value;
41      ++cur;
42      return cur - buf;
43  }
```

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:

```
45  uint32_t decode_varint(const uint8_t** bufp)
46  {
47      const uint8_t* cur = *bufp;
48      uint8_t byte = *cur++;
49      uint32_t value = byte & 0x7f;
50      size_t shift = 7;
51      while (byte >= 0x80) {
52          byte = *cur++;
53          value += (byte & 0x7f) << shift;
54          shift += 7;
55      }
56      *bufp = cur;
57      return value;
58  }
```

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:

```
61  int main()
62  {
63      FILE *uncompWrite = fopen("uncompressed.dat", "wb");
64      FILE *compWrite = fopen("compressed.dat", "wb");
65
66      for (int i = 0; i < 1000000; i++)
67      {
68          uint32_t value = generate_number();
69          fwrite(&value, sizeof(value), 1, uncompWrite);
70          uint8_t buf[4] = {};
71          size_t size = encode_varint(value, buf);
72          fwrite(buf, sizeof(*buf), size, compWrite);
73      }
74      fclose(uncompWrite);
75      fclose(compWrite);
76
```

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.

```
78      FILE *uncompRead = fopen("uncompressed.dat", "rb");
79      FILE *compRead = fopen("compressed.dat", "rb");
80
81      fseek(compRead, 0, SEEK_END);
82      size_t endfile = ftell(compRead);
83      fseek(compRead, 0, SEEK_SET);
84
85      fseek(uncompRead, 0, SEEK_END);
86      size_t uncomp_size = ftell(uncompRead);
87      fseek(uncompRead, 0, SEEK_SET);
88
89      uint8_t compressed[endfile];
90      fread(compressed, sizeof(*compressed), endfile, compRead);
91      uint32_t uncompressed[1000000];
92      fread(uncompressed, sizeof(*uncompressed), 1000000, uncompRead);
93
```

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.

```
94      const uint8_t *cur_comp = compressed;
95      int i = 0;
96      while (cur_comp < compressed + endfile)
97      {
98          if (uncompressed[i] != decode_varint(&cur_comp))
99          {
100             printf("Numbers with index %d not equal\n", i);
101             i = 0;
102             break;
103         }
104         i++;
105     }
106
107     if (i)
108     {
109         printf("All numbers from two files are equal\n");
110     }
111
112     fclose(uncompRead);
113     fclose(compRead);
114
115     double k = (double)uncomp_size / endfile;
116     printf("Compression ratio = %f\n", k);
117
118     return 0;
119 }
```

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:

```
root@DESKTOP-BK46MSC:/mnt/d/Files/prog/prog_lab3# gcc -Wall -o app main.c
root@DESKTOP-BK46MSC:/mnt/d/Files/prog/prog_lab3# ./app
All numbers from two files are equal
Compression ratio = 3.450635
```

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.

```c
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;
}
```

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.

```c
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;
}
```

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:

```c
int write_code_unit(FILE *out, const CodeUnits *code_unit)
{
    return fwrite(code_unit->code, 1, code_unit->length, out);
}
```

Функция принимает на вход указатель на выходной поток, куда мы хотим записать закодированное число и указатель на структуру Code_Units с закодированным числом.

Функция возвращает количество записанных байт в файл.

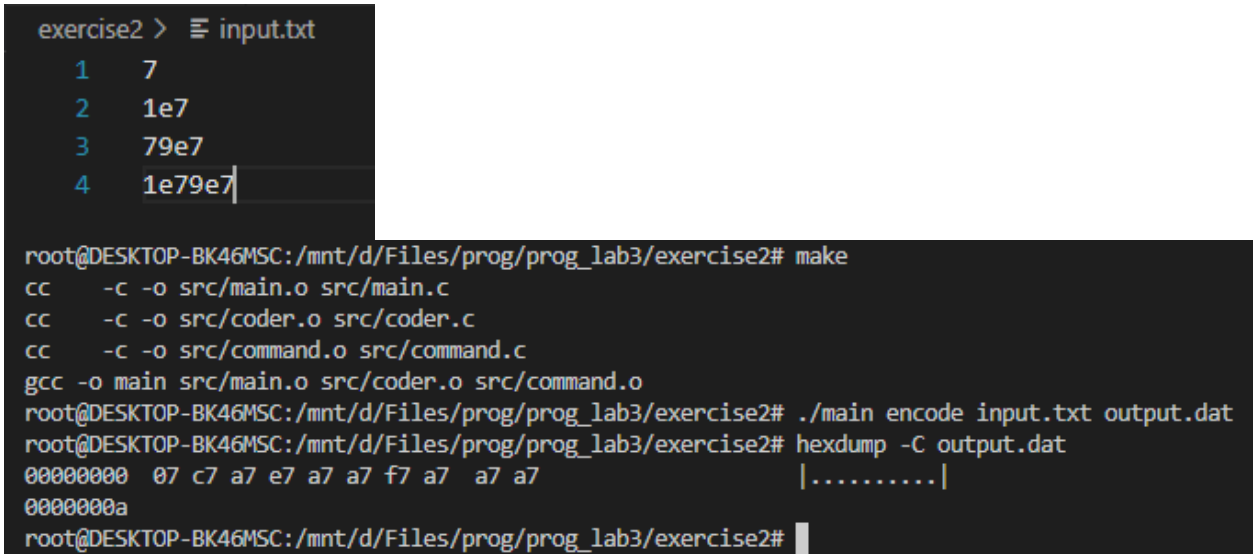
Функция encode_file:

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

```
 9  int encode_file(const char *in_file_name, const char *out_file_name)
10  {
11      FILE *input = fopen(in_file_name, "r");
12      FILE *output = fopen(out_file_name, "wb");
13
14      if (!input)
15      {
16          printf("ERROR: Wrong source to input file\n");
17          return -1;
18      }
19      if (!output)
20      {
21          printf("ERROR: Wrong source to output file\n");
22          fclose(input);
23          return -1;
24      }
25
26      fseek(input, 0, SEEK_END);
27      size_t end_of_input_file_stream = ftell(input);
28      fseek(input, 0, SEEK_SET);
29
30      while(ftell(input) < end_of_input_file_stream)
31      {
32          uint32_t code_point;
33          fscanf(input, "%" SCNx32, &code_point);
34
35          CodeUnits code_unit;
36          if (encode(code_point, &code_unit))
37          {
38              printf("ERROR: Failed to encode number\n");
39              return -1;
40          }
41          write_code_unit(output, &code_unit);
42      }
43
44      fclose(input);
45      fclose(output);
46      return 0;
47  }
```

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.

```c
int read_next_code_unit(FILE *in, CodeUnits *code_units)
{
    code_units->length = 0;
    while (code_units->length == 0 && !feof(in))
    {
        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;
}
```

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.

```
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;
    }
}
```

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.

```c

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)
    {
        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);
}
```

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.

```
root@DESKTOP-BK46MSC:/mnt/d/Files/prog/prog_lab3/exercise2# hexdump -C output.dat
00000000  07 c7 a7 e7 a7 a7 f7 a7  a7 a7                    |..........|
0000000a
root@DESKTOP-BK46MSC:/mnt/d/Files/prog/prog_lab3/exercise2# ./main decode output.dat hexnum.txt
root@DESKTOP-BK46MSC:/mnt/d/Files/prog/prog_lab3/exercise2#
```

exercise2 > hexnum.txt

```
1   7
2   1e7
3   79e7
4   1e79e7
5
```

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  #include <assert.h>
 2  #include <stddef.h>
 3  #include <stdint.h>
 4  #include <stdio.h>
 5  #include <stdlib.h>
 6  /*
 7   * Диапазон              Вероятность
 8   * -------------------- -----------
 9   * [0; 128)              90%
10   * [128; 16384)          5%
11   * [16384; 2097152)      4%
12   * [2097152; 268435455) 1%
13   */
14  uint32_t generate_number()
15  {
16      const int r = rand();
17      const int p = r % 100;
18      if (p < 90) {
19          return r % 128;
20      }
21      if (p < 95) {
22          return r % 16384;
23      }
24      if (p < 99) {
25          return r % 2097152;
26      }
27      return r % 268435455;
28  }
29
30  size_t encode_varint(uint32_t value, uint8_t* buf)
31  {
32      assert(buf != NULL);
33      uint8_t* cur = buf;
34      while (value >= 0x80) {
35          const uint8_t byte = (value & 0x7f) | 0x80;
36          *cur = byte;
37          value >>= 7;
38          ++cur;
39      }
40      *cur = value;
41      ++cur;
42      return cur - buf;
43  }
44
45  uint32_t decode_varint(const uint8_t** bufp)
46  {
47      const uint8_t* cur = *bufp;
48      uint8_t byte = *cur++;
49      uint32_t value = byte & 0x7f;
```

```
50      size_t shift = 7;
51      while (byte >= 0x80) {
52          byte = *cur++;
53          value += (byte & 0x7f) << shift;
54          shift += 7;
55      }
56      *bufp = cur;
57      return value;
58  }
59
60
61  int main()
62  {
63          FILE *uncompWrite = fopen("uncompressed.dat", "wb");
64          FILE *compWrite = fopen("compressed.dat", "wb");
65
66          for (int i = 0; i < 1000000; i++)
67          {
68                  uint32_t value = generate_number();
69                  fwrite(&value, sizeof(value), 1, uncompWrite);
70                  uint8_t buf[4] = {};
71                  size_t size = encode_varint(value, buf);
72                  fwrite(buf, sizeof(*buf), size, compWrite);
73          }
74          fclose(uncompWrite);
75          fclose(compWrite);
76
77
78      FILE *uncompRead = fopen("uncompressed.dat", "rb");
79          FILE *compRead = fopen("compressed.dat", "rb");
80
81          fseek(compRead, 0, SEEK_END);
82          size_t endfile = ftell(compRead);
83          fseek(compRead, 0, SEEK_SET);
84
85      fseek(uncompRead, 0, SEEK_END);
86      size_t uncomp_size = ftell(uncompRead);
87      fseek(uncompRead, 0, SEEK_SET);
88
89          uint8_t compressed[endfile];
90          fread(compressed, sizeof(*compressed), endfile, compRead);
91      uint32_t uncompressed[1000000];
92      fread(uncompressed, sizeof(*uncompressed), 1000000, uncompRead);
93
94          const uint8_t *cur_comp = compressed;
95      int i = 0;
96          while (cur_comp < compressed + endfile)
97          {
98                  if (uncompressed[i] != decode_varint(&cur_comp))
99          {
100             printf("Numbers with index %d not equal\n", i);
101             i = 0;
102             break;
103         }
104         i++;
105         }
106
107     if (i)
```

```
108     {
109         printf("All numbers from two files are equal\n");
110     }
111 
112         fclose(uncompRead);
113         fclose(compRead);
114 
115     double k = (double)uncomp_size / endfile;
116     printf("Compression ratio = %f\n", k);
117 
118         return 0;
119 }
```

Задание 2:

main.c

```
 1  #include <stdio.h>
 2  #include <stdint.h>
 3  #include <stdlib.h>
 4  #include <string.h>
 5  #include <inttypes.h>
 6  
 7  #include "coder.h"
 8  #include "command.h"
 9  
10  int main(int argc, char *argv[])
11  {
12      if (argc != 4)
13      {
14          printf("EROR: Wrong number of arguments\n");
15          return 1;
16      }
17  
18      const char *command = argv[1];
19      const char *in_file_name = argv[2];
20      const char *out_file_name = argv[3];
21  
22      if (strcmp(command, "encode") == 0)
23      {
24          encode_file(in_file_name, out_file_name);
25      }
26      else if (strcmp(command, "decode") == 0)
27      {
28          decode_file(in_file_name, out_file_name);
29      }
30      else
31      {
32          printf("EROR: Wrong command\n");
33          return 1;
34      }
35      return 0;
36  }
```

command.c

```c
#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;
```

```
58        }
59        if (!output)
60        {
61            printf("ERROR: Wrong source to output file\n");
62            fclose(input);
63            return -1;
64        }
65
66        fseek(input, 0, SEEK_END);
67        size_t end_of_input_file_stream = ftell(input);
68        fseek(input, 0, SEEK_SET);
69
70        while(ftell(input) < end_of_input_file_stream)
71        {
72            CodeUnits code_unit;
73            if (read_next_code_unit(input, &code_unit))
74            {
75                printf("ERROR: Failed reading next code unit. End of file
76    reached\n");
77            }
78            uint32_t code_point = decode(&code_unit);
79            fprintf(output, "%" PRIx32 "\n", code_point);
80        }
81
82        fclose(input);
83        fclose(output);
84    }
```

coder.c

```
 1  #include <stdint.h>
 2  #include <stdio.h>
 3  #include <inttypes.h>
 4
 5  #include "coder.h"
 6  #include "command.h"
 7
 8  int encode(uint32_t code_point, CodeUnits *code_units)
 9  {
10      if (code_point < 0x80)
11      {
12          code_units->length = 1;
13          code_units->code[0] = code_point;
14      }
15      else if (code_point < 0x800)
16      {
17          code_units->length = 2;
18          code_units->code[0] = 0xc0 | (code_point >> 6);
19          code_units->code[1] = 0x80 | (code_point & 0x3f);
20      }
21      else if (code_point < 0x10000)
22      {
23          code_units->length = 3;
24          code_units->code[0] = 0xe0 | (code_point >> 12);
25          code_units->code[1] = 0x80 | ((code_point >> 6) & 0x3f);
26          code_units->code[2] = 0x80 | (code_point & 0x3f);
27      }
28      else if (code_point < 0x200000)
```

```
29      {
30          code_units->length = 4;
31          code_units->code[0] = 0xf0 | (code_point >> 18);
32          code_units->code[1] = 0x80 | ((code_point >> 12) & 0x3f);
33          code_units->code[2] = 0x80 | ((code_point >> 6) & 0x3f);
34          code_units->code[3] = 0x80 | (code_point & 0x3f);
35      }
36      else
37      {
38          return -1;
39      }
40
41      return 0;
42  }
43
44  uint32_t decode(const CodeUnits *code_unit)
45  {
46      uint32_t *code_point;
47      uint8_t buf[4];
48      if (code_unit->length == 1)
49      {
50          return (uint32_t)code_unit->code[0];
51      }
52      else if (code_unit->length == 2)
53      {
54          buf[0] = (code_unit->code[0] << 6) | ((code_unit->code[1] << 2)
55  >> 2);
56          buf[1] = ((code_unit->code[0] & 0x1f) >> 2);
57          code_point = (uint32_t *)buf;
58          return *code_point;
59      }
60      else if (code_unit->length == 3)
61      {
62          buf[0] = (code_unit->code[2] & 0x3f) | (code_unit->code[1] <<
63  6);
64          buf[1] = ((code_unit->code[1] >> 2) & 0x0f) | (code_unit-
65  >code[0] << 4);
66          code_point = (uint32_t *)buf;
67          return *code_point;
68      }
69      else if (code_unit->length == 4)
70      {
71          buf[0] = (code_unit->code[3] & 0x3f) | (code_unit->code[2] <<
72  6);
73          buf[1] = ((code_unit->code[2] >> 2) & 0x0f) | (code_unit-
74  >code[1] << 4);
75          buf[2] = ((code_unit->code[1] >> 4) & 0x03) | ((code_unit-
76  >code[0] & 0x07) << 2);
77          code_point = (uint32_t *)buf;
78          return *code_point;
79      }
80  }
81
82  int read_next_code_unit(FILE *in, CodeUnits *code_units)
83  {
84      code_units->length = 0;
85      while (code_units->length == 0)
86      {
```

```
 87          uint8_t *buf = code_units->code;
 88          fread(buf, sizeof(uint8_t), 1, in);
 89          if ((*buf & 0x80) == 0x00)
 90          {
 91              code_units->length = 1;
 92          }
 93          else if ((*buf >> 5) == 0x06)
 94          {
 95              code_units->length = 2;
 96              buf++;
 97              fread(buf, sizeof(uint8_t), 1, in);
 98          }
 99          else if ((*buf >> 4) == 0x0e)
100          {
101              code_units->length = 3;
102              buf++;
103              fread(buf, sizeof(uint8_t), 2, in);
104          }
105          else if ((*buf >> 3) == 0x1e)
106          {
107              code_units->length = 4;
108              buf++;
109              fread(buf, sizeof(uint8_t), 3, in);
110          }
111      }
112      return 0;
113  }
114
115  int write_code_unit(FILE *out, const CodeUnits *code_unit)
116  {
117      return fwrite(code_unit->code, 1, code_unit->length, out);
118  }
```

command.h

```
 1  int encode_file(const char *in_file_name, const char *out_file_name);
 2  int decode_file(const char *in_file_name, const char *out_file_name);
```

coder.h

```
 1  #include <stdlib.h>
 2
 3  enum
 4  {
 5      MaxCodeLength = 4
 6  };
 7
 8  typedef struct {
 9      uint8_t code[MaxCodeLength];
10      size_t length;
11  } CodeUnits;
12
13  int encode(uint32_t code_point, CodeUnits *code_units);
14  uint32_t decode(const CodeUnits *code_unit);
15  int read_next_code_unit(FILE *in, CodeUnits *code_units);
16  int write_code_unit(FILE *out, const CodeUnits *code_unit);
```