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: Гердележов Д.Д.
студент гр. ИВ-122
«10» мая 2022 г.

Проверил: Фульман В.О.
Старший преподаватель
кафедры ВС.
«11» мая 2022 г.

Оценка «_____»

Новосибирск 2022

Оглавление

Задание:

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.

Задание 2: разработать приложение для кодирования и декодирования чисел по описанному ниже алгоритму.

1. Числа [0;28−1][0;28−1] будем представлять в виде `0xxxxxxx`, «как есть»:
   00000000 — 0
   00000001 — 1
   ...
   00000101 — 5
   ...
   01111111 — 127
2. Для бóльших значений в старшем байте будем хранить столько единиц, сколько байт требуется для представления закодированного числа. `110xxxxx` — два, `1110xxxx` — три, и т. д. Все последующие байты имеют вид `10xxxxxx`.

Биты, обозначенные символами `x`, заполняются битами кодируемого числа.

Выполнение работы:

Задание 1:

Структура проекта:

```
.
|-- Makefile
|-- main.c
|-- compressed.dat
 `- uncompressed.dat
```

Открываем два файла для записи в них исходных и закодированных чисел:

```
70      FILE* uncom;
71      FILE* com;
72      uncom = fopen("uncompressed.dat", "wb");
73      com = fopen("compressed.dat", "wb");
```

Генерируем число, после чего записываем его в файл uncompressed.dat. С помощью функции encode_varint кодируем число и записываем результат в файл compressed.dat (запись в файлы в двоичном виде).

```
75      for (int i = 0; i < 1000000; i++) {
76          uint32_t value = generate_number();
77          fwrite(&value, sizeof(uint32_t), 1, uncom);
78          size_t size = encode_varint(value, buf);
79          fwrite(buf, sizeof(uint8_t), size, com);
80      }
```

С помощью функции fseek перемещаем каретку в конец файла, в переменную size записываем количество бит в файле после чего динамически выделяю память для массива и считываю в него весь файл.

```
94      fseek(com, 0, SEEK_END);
95      size = ftell(com);
96      fseek(com, 0, SEEK_SET);
97      comp_numbers = malloc(sizeof(uint8_t) * size);
98      p = comp_numbers;
99
100     for (int i = 0; i < size; i++) {
101         fread(&comp_numbers[i], sizeof(uint8_t), 1, com);
102     }
103
```

В цикле считываю число из файла uncompress.dat и передав в функцию decode_varint адрес массива с данными из файла compress.dat получаю декодированное число. Сравниваю числа, что бы понять правильно ли прошло кодирование (декодирование) информации.

```
for (int i = 0; i < 1000000; i++) {
    uint32_t valueUncom = 0;
    uint32_t valueCom = 0;

    fread(&valueUncom, sizeof(uint8_t), sizeof(uint32_t), uncom);
    valueCom = decode_varint(&p);

    if (i % 200000 == 0) {
        printf("%-12d %d\n", valueUncom, valueCom);
    }
    if (valueUncom != valueCom) {
        status = 1;
    }
}
if (status == 1) {
    printf("NOT OK\n");
} else {
    printf("\n-------\nOK\n");
}
```

Задание 2:
В файле main.c считываю аргументы (названия файлов из консоли) и проверяю их на корректность и в зависимости от заданного параметра decode или encode вызываю соответствующую функцию.

```c
int main(int argc, char* argv[])
{
    if (argc != 4) {
        printf("Для запуска используйте:\n"
               "coder encode <in-file-name> <out-file-name>, или\n"
               "coder decode <in-file-name> <out-file-name>\n");
        return -1;
    }
    if (!strcmp(argv[1], "decode")) {
        if (decode_file(argv[2], argv[3])) {
            printf("Указан несуществующий файл\n");
            return -1;
        }
        printf("Ok\n");
    } else if (!strcmp(argv[1], "encode")) {
        if (encode_file(argv[2], argv[3])) {
            printf("Указан несуществующий файл\n");
            return -1;
        }
        printf("Ok\n");
    } else {
        printf("Для запуска используйте:\n"
               "coder encode <in-file-name> <out-file-name>, или\n"
               "coder decode <in-file-name> <out-file-name>\n");
        return -1;
    }
    return 0;
```

Если информацию необходимо закодировать вызывается функция encode_file (содержится в файле command.c).

```c
int encode_file(const char* in_file_name, const char* out_file_name)
{
    FILE* input;
    FILE* output;

    if ((input = fopen(in_file_name, "r")) == NULL) {
        return -1;
    }
    if ((output = fopen(out_file_name, "wb")) == NULL) {
        return -1;
    }

    CodeUnits code_unit;
    uint32_t code_point;

    while (fscanf(input, "%" SCNx32, &code_point)
           == 1) { // считываем данные из потока input
        encode(code_point, &code_unit);
        write_code_unit(output, &code_unit);
    }
    fclose(input);
    fclose(output);
    return 0;
}
```

В данноой функции открываем файлы и считываем значения, кодируем их с помощью функции encode:

```
int encode(uint32_t code_point, CodeUnits* code_units)
{
    uint8_t count = 0;
    for (uint32_t i = code_point; i > 0; i >>= 1) {
        count++; // подсчет кол-ва битов
    }
    if (count <= 7) // для 1-го байта
    {
        code_units->code[0] = code_point;
        code_units->length = 1;
        return 0;
    } else if (count <= 11) { // для 2-х байтов
        code_units->code[0] = (code_point >> 6) | (3 << 6); // 3 = 11
        code_units->code[1] = (code_point & ~(1 << 6)) | (1 << 7);
        code_units->length = 2;
        return 0;

    } else if (count <= 16) { // для 3-х байтов
        code_units->code[0] = (code_point >> 12) | (7 << 5); // 7 == 111
        code_units->code[1] = ((code_point >> 6) & ~(1 << 6)) | (1 << 7);
        code_units->code[2] = (code_point & ~(1 << 6)) | (1 << 7);
        code_units->length = 3;
        return 0;

    } else if (count <= 21) { //  для 4-х байтов
        code_units->code[0] = (code_point >> 18) | (15 << 4); // 15 ==1111
        code_units->code[1] = ((code_point >> 12) & ~(1 << 6)) | (1 << 7);
        code_units->code[2] = ((code_point >> 6) & ~(1 << 6)) | (1 << 7);
        code_units->code[3] = (code_point & ~(1 << 6)) | (1 << 7);
        code_units->length = 4;
        return 0;
    }
    return -1;
}
```

И записываем результат в файл с помощью функции write_code_unit:

```
int write_code_unit(FILE* out, const CodeUnits* code_unit)
{
    for (int i = 0; i < code_unit->length; i++) {
        fwrite(&code_unit->code[i], sizeof(uint8_t), 1, out);
    }
    return 0;
}
```

Если информацию необходимо декодировать вызывается функция deencode_file (содержится в файле command.c).

```c
int decode_file(const char* in_file_name, const char* out_file_name)
{
    FILE* input;
    FILE* output;
    if ((input = fopen(in_file_name, "rb")) == NULL) {
        return -1;
    }
    if ((output = fopen(out_file_name, "w+")) == NULL) {
        return -1;
    }
    CodeUnits code_unit;
    while (!read_next_code_unit(input, &code_unit)) {
        if (code_unit.code[0] != 0) {
            if (!fprintf(output, "%" PRIx32 "\n", decode(&code_unit))) {
                return -1;
            }
        }
    }
    fclose(input);
    fclose(output);
    return 0;
}
```

Открываем файлы и читаем значения read_next_code_uint:

```c
int read_next_code_unit(
        FILE* in,
        CodeUnits* code_unit) //считывание последовательности из потока in
{
    uint8_t byte;
    fread(&byte, sizeof(uint8_t), 1, in);
    if ((byte & 0xF0) == 0xF0) {
        code_unit->length = 4;
    } else if ((byte & 0xE0) == 0xE0) {
        code_unit->length = 3;
    } else if ((byte & 0xC0) == 0xC0) {
        code_unit->length = 2;
    } else if ((byte >> 7) == 0)
        code_unit->length = 1;
    else {
        code_unit->length = 0;
        return 0;
    }

    code_unit->code[0] = byte;
    if (code_unit->length != 1) {
        for (int i = 1; i < code_unit->length; i++) {
            if (!fread(code_unit->code + i, sizeof(uint8_t), 1, in))
                return -1;
        }
    }
    return 0;
}
```

Декодирую данные с помощью функции decode:

```
uint32_t decode(const CodeUnits* code_unit)
{
    uint32_t code_point;
    if (code_unit->length == 1) {
        return (code_point = code_unit->code[0]);

    } else if (code_unit->length == 2) {
        code_point = (code_unit->code[0] & 31) << 6;          // 31 == 00011111
        code_point = code_point | (code_unit->code[1] & 63); // 63 == 00111111
        return code_point;
    } else if (code_unit->length == 3) {
        code_point = (code_unit->code[0] & 15) << 6;
        code_point = (code_point | (code_unit->code[1] & 63)) << 6;
        code_point = code_point | (code_unit->code[2] & 63);
        return code_point;
    } else if (code_unit->length == 4) {
        code_point = (code_unit->code[0] & 7) << 6; //  7 = 00000111
        code_point = (code_point | (code_unit->code[1] & 63)) << 6;
        code_point = (code_point | (code_unit->code[2] & 63)) << 6;
        code_point = code_point | (code_unit->code[3] & 63);
        return code_point;
    }
    return 0;
}
```

Приложение:

Задание 1:
Main.c

```
#include <assert.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.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 unsigned char** bufp)
{
    const unsigned char* cur = *bufp;
    uint8_t byte = *cur++;
    uint32_t value = byte & 0x7f;
    size_t shift = 7;
    while (byte >= 0x80) {
```

```
54          byte = *cur++;
55          value += (byte & 0x7f) << shift;
56          shift += 7;
57      }
58      *bufp = cur;
59      return value;
60  }
61
62  int main()
63  {
64      srand(time(NULL));
65
66      uint8_t buf[4];
67      uint32_t generate_number();
68      int status = 0;
69
70      FILE* uncom;
71      FILE* com;
72      uncom = fopen("uncompressed.dat", "wb");
73      com = fopen("compressed.dat", "wb");
74
75      for (int i = 0; i < 1000000; i++) {
76          uint32_t value = generate_number();
77          fwrite(&value, sizeof(uint32_t), 1, uncom);
78          size_t size = encode_varint(value, buf);
79          fwrite(buf, sizeof(uint8_t), size, com);
80      }
81
82      fclose(uncom);
83      fclose(com);
84
85      uncom = fopen("uncompressed.dat", "rb");
86      com = fopen("compressed.dat", "rb");
87
88      printf("До сжатия    После\n");
89
90      int size;
91      uint8_t* comp_numbers;
92      const uint8_t* p;
93      fseek(com, 0, SEEK_END);
94      size = ftell(com);
95      fseek(com, 0, SEEK_SET);
96      comp_numbers = malloc(sizeof(uint8_t) * size);
97      p = comp_numbers;
98
99      for (int i = 0; i < size; i++) {
100         fread(&comp_numbers[i], sizeof(uint8_t), 1, com);
101     }
102
103     for (int i = 0; i < 1000000; i++) {
104         uint32_t valueUncom = 0;
105         uint32_t valueCom = 0;
106
107         fread(&valueUncom, sizeof(uint8_t), sizeof(uint32_t), uncom);
108         valueCom = decode_varint(&p);
109
110         if (i % 200000 == 0) {
111             printf("%-12d %d\n", valueUncom, valueCom);
```

```
        }
        if (valueUncom != valueCom) {
            status = 1;
        }
    }
    if (status == 1) {
        printf("NOT OK\n");
    } else {
        printf("\n-------\nOK\n");
    }
    return 0;
}
```

Задание 2:

Main.c

```
#include "coder.h"
#include "command.h"
#include <string.h>

int main(int argc, char* argv[])
{
    if (argc != 4) {
        printf("Для запуска используйте:\n"
               "coder encode <in-file-name> <out-file-name>, или\n"
               "coder decode <in-file-name> <out-file-name>\n");
        return -1;
    }
    if (!strcmp(argv[1], "decode")) {
        if (decode_file(argv[2], argv[3])) {
            printf("Указан несуществующий файл\n");
            return -1;
        }
        printf("Ok\n");
    } else if (!strcmp(argv[1], "encode")) {
        if (encode_file(argv[2], argv[3])) {
            printf("Указан несуществующий файл\n");
            return -1;
        }
        printf("Ok\n");
    } else {
        printf("Для запуска используйте:\n"
               "coder encode <in-file-name> <out-file-name>, или\n"
               "coder decode <in-file-name> <out-file-name>\n");
        return -1;
    }
    return 0;
}
```

Command.c

```
#include "command.h"
#include "coder.h"

int encode_file(const char* in_file_name, const char* out_file_name)
{
```

```
 6     FILE* input;
 7     FILE* output;
 8
 9     if ((input = fopen(in_file_name, "r")) == NULL) {
10         return -1;
11     }
12     if ((output = fopen(out_file_name, "wb")) == NULL) {
13         return -1;
14     }
15
16     CodeUnits code_unit;
17     uint32_t code_point;
18
19     while (fscanf(input, "%" SCNx32, &code_point)
20            == 1) { // считываем данные из потока input
21         encode(code_point, &code_unit);
22         write_code_unit(output, &code_unit);
23     }
24     fclose(input);
25     fclose(output);
26     return 0;
27 }
28
29 int decode_file(const char* in_file_name, const char* out_file_name)
30 {
31     FILE* input;
32     FILE* output;
33     if ((input = fopen(in_file_name, "rb")) == NULL) {
34         return -1;
35     }
36     if ((output = fopen(out_file_name, "w+")) == NULL) {
37         return -1;
38     }
39     CodeUnits code_unit;
40     while (!read_next_code_unit(input, &code_unit)) {
41         if (code_unit.code[0] != 0) {
42             if (!fprintf(output, "%" PRIx32 "\n", decode(&code_unit))) {
43                 return -1;
44             }
45         }
46     }
47     fclose(input);
48     fclose(output);
49     return 0;
50 }
```

Coder.c

```
1  #include "coder.h"
2
3 int encode(uint32_t code_point, CodeUnits* code_units)
4 {
5     uint8_t count = 0;
6     for (uint32_t i = code_point; i > 0; i >>= 1) {
7         count++; // подсчет кол-ва битов
8     }
9     if (count <= 7) // для 1-го байта
```

```
10      {
11          code_units->code[0] = code_point;
12          code_units->length = 1;
13          return 0;
14      } else if (count <= 11) { // для 2-х байтов
15          code_units->code[0] = (code_point >> 6) | (3 << 6); // 3 = 11
16          code_units->code[1] = (code_point & ~(1 << 6)) | (1 << 7);
17          code_units->length = 2;
18          return 0;
19
20      } else if (count <= 16) { // для 3-х байтов
21          code_units->code[0] = (code_point >> 12) | (7 << 5); // 7 == 111
22          code_units->code[1] = ((code_point >> 6) & ~(1 << 6)) | (1 << 7);
23          code_units->code[2] = (code_point & ~(1 << 6)) | (1 << 7);
24          code_units->length = 3;
25          return 0;
26
27      } else if (count <= 21) { //  для 4-х байтов
28          code_units->code[0] = (code_point >> 18) | (15 << 4); // 15 ==1111
29          code_units->code[1] = ((code_point >> 12) & ~(1 << 6)) | (1 << 7);
30          code_units->code[2] = ((code_point >> 6) & ~(1 << 6)) | (1 << 7);
31          code_units->code[3] = (code_point & ~(1 << 6)) | (1 << 7);
32          code_units->length = 4;
33          return 0;
34      }
35      return -1;
36  }
37  uint32_t decode(const CodeUnits* code_unit)
38  {
39      uint32_t code_point;
40      if (code_unit->length == 1) {
41          return (code_point = code_unit->code[0]);
42
43      } else if (code_unit->length == 2) {
44          code_point = (code_unit->code[0] & 31) << 6;
45          code_point = code_point | (code_unit->code[1] & 63);
46          return code_point;
47      } else if (code_unit->length == 3) {
48          code_point = (code_unit->code[0] & 15) << 6;
49          code_point = (code_point | (code_unit->code[1] & 63)) << 6;
50          code_point = code_point | (code_unit->code[2] & 63);
51          return code_point;
52      } else if (code_unit->length == 4) {
53          code_point = (code_unit->code[0] & 7) << 6; //  7 = 00000111
54          code_point = (code_point | (code_unit->code[1] & 63)) << 6;
55          code_point = (code_point | (code_unit->code[2] & 63)) << 6;
56          code_point = code_point | (code_unit->code[3] & 63);
57          return code_point;
58      }
59      return 0;
60  }
61  int read_next_code_unit(
62          FILE* in,
63          CodeUnits* code_unit) //считывание последовательности из потока in
64  {
65      uint8_t byte;
66      fread(&byte, sizeof(uint8_t), 1, in);
67      if ((byte & 0xF0) == 0xF0) {
```

```
68          code_unit->length = 4;
69      } else if ((byte & 0xE0) == 0xE0) {
70          code_unit->length = 3;
71      } else if ((byte & 0xC0) == 0xC0) {
72          code_unit->length = 2;
73      } else if ((byte >> 7) == 0)
74          code_unit->length = 1;
75      else {
76          code_unit->length = 0;
77          return 0;
78      }
79
80      code_unit->code[0] = byte;
81      if (code_unit->length != 1) {
82          for (int i = 1; i < code_unit->length; i++) {
83              if (!fread(code_unit->code + i, sizeof(uint8_t), 1, in))
84                  return -1;
85          }
86      }
87      return 0;
88  }
89
90  int write_code_unit(FILE* out, const CodeUnits* code_unit)
91  {
92      for (int i = 0; i < code_unit->length; i++) {
93          fwrite(&code_unit->code[i], sizeof(uint8_t), 1, out);
94      }
95      return 0;
96  }
```

Coder.h

```
1  #pragma once
2  #include <inttypes.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  enum { MaxCodeLength = 4 };
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);
```

Command.h

```
1 #pragma once
2 #include <inttypes.h>
3 #include <stdio.h>
4
5 int encode_file(const char* in_file_name, const char* out_file_name);
```

```
6 int decode_file(const char* in_file_name, const char* out_file_name);
```