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 _____ Денисов М. А.

Проверил:
Старший преподаватель
Кафедры ВС
«17» мая 2023 г. _____ Фульман В.О.

Оценка «_____»

Новосибирск 2023

ОГЛАВЛЕНИЕ

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:

```
#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;
}

```

```c
#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

```
int main()
{
    size_t size = 0;
    FILE *fp;
    FILE *unfp;
    fp = fopen("compressed.dat", "wb");
    unfp = fopen("uncompressed.dat", "wb");
    for (int i = 0; i < 1000000; i++)
    {
        uint32_t number = generate_number();
        fwrite(&number, sizeof(uint32_t), 1, unfp);
        uint8_t buf[4];
        size = encode_varint(number, buf);
        fwrite(buf, sizeof(uint8_t), size, fp);
        const uint8_t *cur_uncomp = buf;
        uint32_t value = decode_varint(&cur_uncomp);
        if (value != number)
        {
            printf("ERROR");
        }
    }
    fclose(fp);
    fclose(unfp);
    FILE *fpcomp = fopen("compressed.dat", "rb");
    fseek(fpcomp, 0, SEEK_END);
    long int count = ftell(fpcomp);
    fseek(fpcomp, 0, SEEK_SET);
    while ((ftell(fpcomp)) != count)
    {
        uint8_t compressed[4];
        fread(compressed, sizeof(uint8_t), 1, fpcomp);
        const uint8_t *curcomp = compressed;
        uint32_t value = decode_varint(&curcomp);
    }
    fclose(fpcomp);
    return 0;
}
```

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

```
root@DESKTOP-RCKFD8V:~/proga/laba3/N1# stat compressed.dat
  File: compressed.dat
  Size: 1159207         Blocks: 2304       IO Block: 4096   regular file
Device: 2h/2d   Inode: 5629499534362328  Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-05-17 10:40:16.583529600 +0700
Modify: 2023-05-17 10:40:16.501536400 +0700
Change: 2023-05-17 10:40:16.501536400 +0700
 Birth: -
```

Размер файла uncompressed.dat

```
root@DESKTOP-RCKFD8V:~/proga/laba3/N1# stat uncompressed.dat
  File: uncompressed.dat
  Size: 4000000         Blocks: 7936       IO Block: 4096   regular file
Device: 2h/2d   Inode: 2533274790556738  Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-05-17 10:40:16.502536400 +0700
Modify: 2023-05-17 10:40:16.502536400 +0700
Change: 2023-05-17 10:40:16.502536400 +0700
 Birth: -
```

Задание 2

```
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;
}
```

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 c0, 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.

```
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)
        return number;
    }
    return 0;
}
```

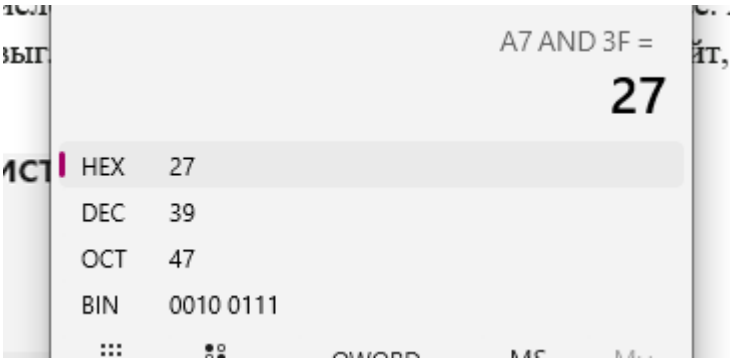
Функция decode.

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.

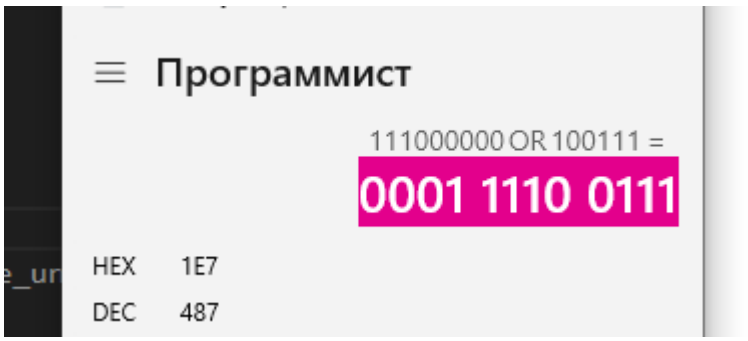
Сделаем пример на одном числе, том же, которым и пользовались в encode. 2 байта закодированного числа 1e7 выглядит так – C7 A7. Декодируем первый байт, сначала проводим операцию “и”.

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.



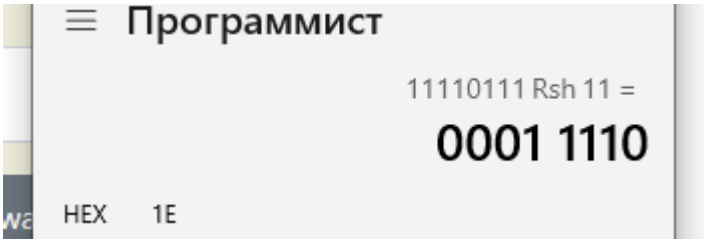

```c
int read_next_code_unit(FILE *in, CodeUnit *code_unit)
{
    uint8_t buf = 0;
    fread(&buf, sizeof(uint8_t), 1, in);
    if ((buf >> 7) == 0)
    {
        code_unit->code[0] = buf;
        code_unit->length = 1;
    }
    else if ((buf >> 5) == 0x06)
    {
        code_unit->code[0] = buf;
        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;
}
```

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.

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

Функция write_code_unit

Записывает в файл байты закодированного числа.

```c
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;
}
```

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.

```c
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;
}
```

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

```c
int main(int argc, char *argv[])
{
    if (argc != 4)
    {
        printf("Usage:\ncoder encode <in-file-name> <out-file-name>\ncoder 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;
    }
}
```

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

```c
#include <stdio.h>
#include <string.h>

#include "coder.h"
#include "command.h"
int main(int argc, char *argv[])
{
    if (argc != 4)
    {
        printf("Usage:\ncoder encode <in-file-name> <out-file-
name>\ncoder 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

```c
#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;
```

```
 9
10      in = fopen(in_file_name, "r");
11
12      if (!in)
13      {
14          return -1;
15      }
16
17      out = fopen(out_file_name, "wb");
18
19      if (!out)
20      {
21          return -1;
22      }
23      while (!feof(in))
24      {
25          uint32_t code_point;
26          CodeUnit code_unit;
27          fscanf(in, "%" SCNx32, &code_point);
28          if (encode(code_point, &code_unit) < 0)
29          {
30              printf("Error encode function\n");
31              return -1;
32          }
33          write_code_unit(out, &code_unit);
34      }
35      fclose(in);
36      fclose(out);
37      return 0;
38  }
39
40  int decode_file(const char *in_file_name, const char *out_file_name)
41  {
42      FILE *in;
43      FILE *out;
44      in = fopen(in_file_name, "r");
45      if (!in)
46      {
47          return -1;
48      }
49      out = fopen(out_file_name, "wb");
50      if (!out)
51      {
52          return -1;
53      }
54      CodeUnit code_unit;
55      fseek(in, 0, SEEK_SET);
56      fseek(in, 0, SEEK_END);
57      long end_symbol = ftell(in);
58      fseek(in, 0, SEEK_SET);
59      while ((ftell(in)) != (end_symbol))
60      {
61          if ((read_next_code_unit(in, &code_unit)) == 0)
62          {
63              uint32_t num = decode(&code_unit);
64              if (decode(&code_unit) < 0)
65              {
```

```
66                  printf("Error decode function\n");
67                  return -1;
68              }
69              fprintf(out, "%" PRIx32 "\n", num);
70          }
71      }
72      fclose(in);
73      fclose(out);
74      return 0;
75  }
76
```

Coder.c

```
 1  #include <stdio.h>
 2  #include <inttypes.h>
 3  #include "coder.h"
 4  #include "command.h"
 5  int encode(uint32_t code_point, CodeUnit *code_unit)
 6  {
 7      if (code_point < 0x80)
 8      {
 9          code_unit->length = 1;
10          code_unit->code[0] = code_point;
11      }
12      else if (code_point < 0x800)
13      {
14          code_unit->length = 2;
15          code_unit->code[0] = 0xc0 | (code_point >> 6);
16          code_unit->code[1] = 0x80 | (code_point & 0x3f);
17      }
18      else if (code_point < 0x10000)
19      {
20          code_unit->length = 3;
21          code_unit->code[0] = 0xe0 | (code_point >> 12);
22          code_unit->code[1] = 0x80 | ((code_point >> 6) & 0x3f);
23          code_unit->code[2] = 0x80 | (code_point & 0x3f);
24      }
25      else if (code_point < 0x200000)
26      {
27          code_unit->length = 4;
28          code_unit->code[0] = 0xf0 | (code_point >> 18);
29          code_unit->code[1] = 0x80 | ((code_point >> 12) & 0x3f);
30          code_unit->code[2] = 0x80 | ((code_point >> 6) & 0x3f);
31          code_unit->code[3] = 0x80 | (code_point & 0x3f);
32      }
33      else
34      {
35          return -1;
36      }
37      return 0;
38  }
39
40  uint32_t decode(const CodeUnit *code_unit)
41  {
```

```
42          uint32_t number;
43          if ((code_unit->length) == 1)
44          {
45              number = code_unit->code[0];
46              return number;
47          }
48          else if ((code_unit->length) == 2)
49          {
50              number = (((code_unit->code[0]) & 0x1f) << 6) | ((code_unit-
51      >code[1]) & 0x3f);
52              return number;
53          }
54          else if ((code_unit->length) == 3)
55          {
56              number = ((((code_unit->code[0]) & 0xf) << 12) |
57      (((code_unit->code[1]) & 0x3f) << 6) | ((code_unit->code[2]) &
58      0x3f));
59              return number;
60          }
61          else if (code_unit->length == 4)
62          {
63              number = (((((code_unit->code[0]) & 0x7) << 18) |
64      (((code_unit->code[1]) & 0x3f) << 12) | ((code_unit->code[2]) &
65      0x3f) << 6) | ((code_unit->code[3]) & 0x3f));
66              return number;
67          }
68          return 0;
69      }
70
71      int read_next_code_unit(FILE *in, CodeUnit *code_unit)
72      {
73          uint8_t buf = 0;
74          fread(&buf, sizeof(uint8_t), 1, in);
75          if ((buf >> 7) == 0)
76          {
77              code_unit->code[0] = buf;
78              code_unit->length = 1;
79          }
80          else if ((buf >> 5) == 0x06)
81          {
82              code_unit->code[0] = buf;
83              fread(&buf, sizeof(uint8_t), 1, in);
84              code_unit->code[1] = buf;
85              code_unit->length = 2;
86          }
87          else if ((buf >> 4) == 0x0e)
88          {
89              code_unit->code[0] = buf;
90              fread(&buf, sizeof(uint8_t), 1, in);
91              code_unit->code[1] = buf;
92              fread(&buf, sizeof(uint8_t), 1, in);
93              code_unit->code[2] = buf;
94              code_unit->length = 3;
95          }
96          else if ((buf >> 3) == 0x1e)
97          {
98              code_unit->code[0] = buf;
```

```
99              fread(&buf, sizeof(uint8_t), 1, in);
100             code_unit->code[1] = buf;
101             fread(&buf, sizeof(uint8_t), 1, in);
102             code_unit->code[2] = buf;
103             fread(&buf, sizeof(uint8_t), 1, in);
104             code_unit->code[3] = buf;
105             code_unit->length = 4;
106         }
107         return 0;
108     }
109
110     int write_code_unit(FILE *out, const CodeUnit *code_unit)
111     {
112         int result = fwrite(code_unit->code, 1, code_unit->length, out);
113         return result;
114     }
115
```