Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и
информатики»
(СибГУТИ)

Кафедра вычислительных систем

ОТЧЕТ
по практической работе 3

по дисциплине «**Программирование**»

Выполнил:
студент гр. ИВ-122
«03» февраля 2022 г. _____ Клепче Г.В..

Проверил:
старший преподаватель
Кафедры ВС
«__» февраля 2022 г. _____ Фульман В.О.

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

Алгоритм кодирования:

Числа будем представлять в виде “0xxxxxxx” , «как есть»:

```
00000000 – 0
00000001 – 1
...
00000101 – 5
...
01111111 – 127
```

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.

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

ВЫПОЛНЕНИЕ РАБОТЫ

ЗАДАНИЕ 1

В функции “main” вызываем функцию “write_and_read()”

```
1  int main()
2  {
3     write_and_read();
4     return 0;
5  }
```

которая открывает два файла для записи в них данных:

```
1  FILE* uncom;
2  FILE* com;
3  if ((uncom = fopen("uncompressed.dat", "wb"))  == NULL){
4        return -1;
5  }
6     if ((com = fopen("compressed.dat", "wb")) == NULL){
7        return -1;
8  }
```

После записываем 1000000 значений в два файла.

После чего закрываем файлы и открываем их заново для проверки. Декодируем числа из двух файлов и сравниваем их.

```
1     for (int i = 0; i < 1000000; i++) {
2           uint32_t valueUncom = 0;
3           uint32_t valueCom = 0;
4           fread(&valueUncom, sizeof(uint8_t),  sizeof(uint32_t), uncom);
5           fread(&valueCom,  sizeof(uint8_t), sizeBuf[i], com);
6           const uint8_t* value  = (uint8_t*)&valueCom;
7           valueCom = decode_varint(&value);
8           if (valueUncom != valueCom) {
9                 flag = 1;
10          };
11       }
```

ЗАДАНИЕ 2

Считываем аргументы командной строки:

Если вторым аргументом передан “encode” вызываем функцию encode_file.

Открываем файл, имя которого передали первый параметром для чтения, и второй файл для записи.

Пока файл не кончится, считываем целые числа из входящего файла, кодируем их и записываем в исходящий файл, потом закрываем файлы:

Кодирование происходит по следующему алгоритму: если число меньше 128, то оно хранится в одном байте без кодировки.

Если число меньше 2048, то оно хранится в двух байтах, где байты записаны в виде: 110xxxxx|10xxxxxx.

Если число меньше 65536, то оно хранится в трех байтах, где байты записаны в виде: 1110xxxx|10xxxxxx|10xxxxxx.

Если число меньше 4194304, то оно хранится в четырех байтах, где байты записаны в виде: 1111xxxx| 10xxxxxx|10xxxxxx|10xxxxxx.

Если вторым аргументом передано «decode», вызываем функцию decode_file с теми же параметрами.

Открываем файлы по переданным именам.

Пока файл не закончится, читаем из входящего файла и записываем декодированные числа в исходящий, после закрываем файлы.

Функция декодирования декодирует переданный ей код в целое число.

ПРИЛОЖЕНИЕ

ЗАДАНИЕ 1

main:

```c
#include <assert.h>
#include <inttypes.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

size_t encode_varint(uint32_t, uint8_t*);
uint32_t decode_varint(const uint8_t**);
uint32_t generate_number();
int write_and_read();

size_t encode_varint(uint32_t value, uint8_t* buf)
{
    assert(buf != NULL);
    uint8_t varint[16];
    uint32_t pos = sizeof(varint) - 1;
    varint[pos] = value & 127;

    while (value >>= 7)
        varint[--pos] = 128 | (--value & 127);
    if (buf)
        memcpy(buf, varint + pos, sizeof(varint) - pos);
    return sizeof(varint) - pos;
}

uint32_t decode_varint(const uint8_t** bufp)
{
    const uint8_t* cur = *bufp;
    uint8_t byte = *cur++;
    uint32_t value = byte & 0x7f;
    while (byte >= 0x80) {
        value += 1;
        byte = *cur++;
        value = (value << 7) + (byte & 127);
    }
    *bufp = cur;
    return value;
}

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 % 268435455;
    return r % 268435455;
}
```

```
55  int write_and_read()
56  {
57      srand(time(NULL));
58
59      uint8_t buf[4];
60      size_t sizeBuf[1000000];
61
62      FILE* uncom;
63      FILE* com;
64      if ((uncom = fopen("uncompressed.dat", "wb")) == NULL) {
65          return -1;
66      }
67      if ((com = fopen("compressed.dat", "wb")) == NULL) {
68          return -1;
69      }
70
71      for (int i = 0; i < 1000000; i++) {
72          uint32_t value = generate_number();
73          fwrite(&value, sizeof(uint32_t), 1, uncom);
74          size_t size = encode_varint(value, buf);
75          fwrite(buf, sizeof(uint8_t), size, com);
76          if (i < 1000000)
77              sizeBuf[i] = size;
78      }
79
80      uint8_t flag = 0;
81
82      fclose(uncom);
83      fclose(com);
84      if ((uncom = fopen("uncompressed.dat", "rb")) == NULL) {
85          return -1;
86      }
87      if ((com = fopen("compressed.dat", "rb")) == NULL){
88          return -1;
89      }
90
91      for (int i = 0; i < 1000000; i++) {
92          uint32_t valueUncom = 0;
93          uint32_t valueCom = 0;
94
95          fread(&valueUncom, sizeof(uint8_t), sizeof(uint32_t), uncom);
96          fread(&valueCom, sizeof(uint8_t), sizeBuf[i], com);
97
98          const uint8_t* value = (uint8_t*)&valueCom;
99          valueCom = decode_varint(&value);
100         if (valueUncom != valueCom) {
101             flag = 1;
102         };
103     }
104     if (flag == 1) {
105         printf("Not compared\n");
106     } else {
107         printf("Compared\n");
108     };
```

```
109     return 0;
110 }
111 int main()
112 {
113     write_and_read();
114     return 0;
115 }
```

ЗАДАНИЕ 2

main:

```
1   #include "coder.h"
2   #include "command.h"
3   #include <string.h>
4
5   int main(int argc, char* argv[])
6   {
7      if (argc != 4) {
8         printf("Usage:\n"
9               "1.coder encode <in-file-name> <out-file-name>\n"
10              "coder decode <in-file-name> <out-file-name>\n");
11        return -1;
12     }
13
14     if (!strcmp(argv[1], "decode")) {
15        if (decode_file(argv[2], argv[3])) {
16           printf("Error decode file\n");
17           return -1;
18        }
19     } else if (!strcmp(argv[1], "encode")) {
20
21        if (encode_file(argv[2], argv[3])) {
22           printf("Error encode file\n");
23           return -1;
24        }
25     } else {
26        printf("Usage:\n"
27              "2.coder encode <in-file-name> <out-file-name>\n"
28              "coder decode <in-file-name> <out-file-name>\n");
29     }
30     return 0;
31  }
```

coder.c:

```c
#include "coder.h"

int encode(uint32_t code_point, CodeUnits* code_units)
{
   uint8_t count = 0;
   for (uint32_t i = code_point; i > 0; i >>= 1) {
      count++;
   }
   if (count <= 7) {
      code_units->code[0] = code_point;
      code_units->length = 1;
      return 0;
   } else if (count <= 11) {
      code_units->code[0] = (code_point >> 6) | (3 << 6);
      code_units->code[1] = (code_point & ~(1 << 6)) | (1 << 7);
      code_units->length = 2;
      return 0;

   } else if (count <= 16) {
      code_units->code[0] = (code_point >> 12) | (7 << 5);
      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) {
      code_units->code[0] = (code_point >> 18) | (15 << 4);
      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;
}
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;
      code_point = code_point | (code_unit->code[1] & 63);
      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;
      code_point = (code_point | (code_unit->code[1] & 63)) << 6;
      code_point = (code_point | (code_unit->code[2] & 63)) << 6;
```

```
55          code_point = code_point | (code_unit->code[3] & 63);
56          return code_point;
57      }
58      return 0;
59  }
60  int read_next_code_unit(FILE* in, CodeUnits* code_unit)
61  {
62      uint8_t byte;
63      fread(&byte, sizeof(uint8_t), 1, in);
64      if ((byte & 0xF0) == 0xF0) {
65          code_unit->length = 4;
66      } else if ((byte & 0xE0) == 0xE0) {
67          code_unit->length = 3;
68      } else if ((byte & 0xC0) == 0xC0) {
69          code_unit->length = 2;
70      } else if ((byte >> 7) == 0)
71          code_unit->length = 1;
72      else {
73          code_unit->length = 0;
74          return 0;
75      }
76
77      code_unit->code[0] = byte;
78      if (code_unit->length != 1) {
79          for (int i = 1; i < code_unit->length; i++) {
80              if (!fread(code_unit->code + i, sizeof(uint8_t), 1, in))
81                  return -1;
82          }
83      }
84      return 0;
85  }
86
87  int write_code_unit(FILE* out, const CodeUnits* code_unit)
88  {
89      for (int i = 0; i < code_unit->length; i++) {
90          fwrite(&code_unit->code[i], sizeof(uint8_t), 1, out);
91      }
92      return 0;
93  }
```

coder.h

```c
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.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 read_next_code_unit(FILE* in, CodeUnits* code_units);
int write_code_unit(FILE* out, const CodeUnits* code_unit);
```

command.c:

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

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, "w+")) == NULL) {
        return -1;
    }

    CodeUnits code_unit;
    uint32_t code_point;

    while (fscanf(input, "%" SCNx32, &code_point) == 1) {
        encode(code_point, &code_unit);
        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;
    FILE* output;
    if ((input = fopen(in_file_name, "r")) == 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;
}
```

command.h:

```
1  #ifndef COMMAND_H
2  #define COMMAND_H
3  #include <inttypes.h>
4  #include <stdio.h>
5
6  int encode_file(const char* in_file_name, const char* out_file_name);
7  int decode_file(const char* in_file_name, const char* out_file_name);
8
9  #endif
```