ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»

Факультет безопасности информационных технологий

Дисциплина:

«Системное программирование»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

Вариант 13

Выполнил:

Студент гр. N3352

Шарипов Ф.Р.

Проверил:

Гирик А.В.

Санкт-Петербург

2020г.

Задание: В бинарном файле найти все текстовые строки (последовательности печатных символов).

Опции:

- -l длина – минимальное количество символов, которые считаются строкой;
- -a – сортировка найденных строк по алфавиту;
- -n – сортировка найденных строк по длине;
- -r – сортировка в обратном порядке (опция может встречаться только в сочетании с -a или -n);

Код программы:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#define constSize 10
typedef struct array
{
   char *start; //указатель на начало строки
   char *end; // указатель на конец строки
   int len1; // выделенный размер
   int len; // уже использованный размер
} array;
// переменные для парсинга
struct globalArgs_t
{
   char *input;
   char *output;
   int l;
```

```
    int a;
    int n;
    int r;
} globalArgs;
char *program_name;
FILE *in;
FILE *out;
// начальное выделение памяти под строку
void Create_array(array *);
// добавление данных в конец строки
void Add_to_array(array *, char *, long);
// освобождение памяти
void Free_array(array *);
// отображение мануала и ошибок
void Show_err( char* );
// парсинг входного и выходного файла + опция -h
void Parsing( int, char ** );
// чтение входного файла
int ReadFile(FILE *, array *);
// удалить список
void Drop(array *);
// напечатать список строк
void print(FILE *, array *);
// сравниватели по алфавиту и длине
int alp_len(array **a, array **b)
{
    int res = len_comp(a,b);
    if (!res) res = strcmp((*a)->start,(*b)->start);
    return (globalArgs.r ? -1: 1)*res;
}
```

```
int len_comp(array **a, array **b)
{
   return (globalArgs.r ? -1: 1)*((*a)->len - (*b)->len);
}
int main( int argc, char **argv )
{
   program_name = argv[0];
   Parsing(argc, argv);
   array dict;
   Create_array(&dict);
   int count = ReadFile(in, &dict);
   if (globalArgs.n)
   {
      qsort(dict.start, count, sizeof(array**),len_comp);
   }
   else if (globalArgs.a)
   {
      qsort(dict.start, count, sizeof(array**),alp_len);
   }
   printf("\n");
   print(out, &dict);
   Drop(&dict);
   Free_array(&dict);
   return EXIT_SUCCESS;
}
int ReadFile(FILE *stream, array *buf)
{
   /* копируем в buf пока не EOF */
   int len = 0;
   int c = EOF;
```

```
array *tmp = NULL;
do {
  c = fgetc(stream);
  if (c != EOF)
  {
    if (isprint(c))
    {
      if (!tmp)
      {
        tmp = calloc(1, sizeof(array));
        Create_array(tmp);
      }
      if (tmp)
      {
        add_to_array(tmp, &c, 1);
      }
    }
    else
    {
      if (tmp)
      {
        LAST_TRY:
        if (tmp->len >= globalArgs.l)
        {
          add_to_array(buf, &tmp, sizeof(tmp));
          len++;
        }
        else
        {
          Free_array(tmp);
```

```
            free(tmp);
          }
          tmp = NULL;
        }
      }
    }
  } while (c != EOF);
  if (tmp) goto LAST_TRY;
  return len;
}
void Drop(array * buf)
{
  for (array **cur_string = buf->start; cur_string < buf->end; cur_string++)
  {
    Free_array(*cur_string);
    free(*cur_string);
  }
}
void print(FILE *stream, array *buf)
{
  for (array **cur_string = buf->start; cur_string < buf->end; cur_string++)
  {
    fwrite((*cur_string)->start, sizeof(char), (*cur_string)->len,stream);
    fwrite("\n", sizeof(char), 1, stream);
  }
}
void Show_err( char* error )
{
  /*
   отображает использование в stdin или в stderr
```

```
    */
    fprintf( error ? stderr : stdout, "%sUsage: %s [options][input_file
[output_file]]\n"
    "\tOptions:\n"
    "\t\t-l - minimal array length, if less, ignore\n"
    "\t\t-a - sort by alphabet\n"
    "\t\t-n - sort by length\n"
    "\t\t-r - descending sort (only with -a or -n)\n"
    , (error ? error : ""), program_name);
    exit( error ? EXIT_FAILURE : EXIT_SUCCESS );
}
void Parsing( int argc, char **argv )
{
  /*
  Parse the options and parameters
  */
  const char *optString = "l:anrp?"; // это значит опции f и какое-то его
значение, r и значение, i без значения(флаг), h флаг, неизвестно
  int opt = 0;
  globalArgs.input = NULL;
  globalArgs.output = NULL;
  globalArgs.l = 0;
  globalArgs.a = 0;
  globalArgs.n = 0;
  globalArgs.r = 0;
  // Parse options
  do
  {
    switch( opt )
    {
```

```
    case 'l':
    if (!sscanf(optarg, "%d", &globalArgs.l) || globalArgs.l < 0)
    Show_err("Error: length must be positive integer\n");
    break;
    case 'a':
    globalArgs.a = 1;
    break;
    case 'n':
    globalArgs.n = 1;
    break;
    case 'r':
    globalArgs.r = 1;
    break;
    case 'h':
    Show_err(0);
    break;
    case '?':
    Show_err("Error: cannot parse the options\n");
    break;
  }
  opt = getopt( argc, argv, optString );
} while( opt != -1 );
// Parse positional parameters
// если указали файлы
if (argc >= optind)
globalArgs.input = argv[optind];
if (argc > optind)
globalArgs.output = argv[optind + 1];
// назначение потоков ввода и вывода
if (globalArgs.input)
```

```
    in = fopen(globalArgs.input, "rb");
  else
    in = stdin;
  if (globalArgs.output)
    out = fopen(globalArgs.output, "wb");
  else
    out = stdout;
  if (globalArgs.r && !globalArgs.n && !globalArgs.a)
      Show_err("Error: option -r can be used only with -n or - a\n");
  if (!in || !out)
      Show_err("Error: cannot open the file\n");
}
void add_to_array(array *dest, char *src, long len)
{
  if (dest->len + len > dest->len1)
  { // проверка "хватит ли памяти?" и перевыделение, если нет
    dest->start = (char*)realloc(dest->start, sizeof(char) * (dest->len + len +
constSize));
    if (!dest->start)
      exit( EXIT_FAILURE );
    dest->end = dest->start + dest->len; // изменение указателей, т.к. после
перевыделения может измениться адрес строки
    dest->len1 = dest->len + len + constSize;
  }
    memcpy(dest->end, src, len); // добавление данных в конец
    dest->len += len;
    dest->end += len;
}
void Create_array(array *s)
{
```

```
    s->start = (char*)malloc(constSize);
    if (!s->start)
      exit( EXIT_FAILURE );
    s->end = s->start;
    s->len1 = constSize;
    s->len = 0;
}
void Free_array(array *s)
{
    free(s->start)
;}
```