**Министерство науки и высшего образования Российской Федерации**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Факультет безопасности информационных технологий**

**Дисциплина:**

«Основы системного программирования»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1**

«Лабораторная 1»

**Выполнили:**

Ахраров Али, студент группы N3250

**Проверил:**

Грозов Владимир Андреевич

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(отметка о выполнении)

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

# Вариант 2

# Задание:

Разработать на языке C для ОС Linux программу, позволяющую выполнять

рекурсивный поиск файлов, начиная с указанного каталога, в соответствии с условием из

Табл. 3 и вариантом из Табл. 4.

Программа должна представлять собой консольную утилиту, настройка работы

которой осуществляется путем передачи аргументов в строке запуска и/или с помощью

переменных окружения (опции необязательны, аргументы каталог и цель\_поиска —

обязательны):

lab11abcNXXXXX [опции] каталог цель\_поиска

Программа должна выполнять рекурсивный поиск файлов, отвечающих критерию,

который задается аргументом цель\_поиска в соответствии с условием из Табл. 3. При

обнаружении файла, отвечающего заданным критериям поиска, программа должна вывести в

стандартный поток вывода полный путь к этому файлу.

При указании опций -h или -v (или их "длинных" аналогов --help или --version)

выполняется вывод информации, заданной опцией, и работа программы завершается. Опции,

которые должны поддерживаться программой, приведены в Табл. 1.

При определении переменной окружения LAB11DEBUG в стандартный поток ошибок

должна выводиться информация о том, что и в каком месте файла нашлось (чтобы было

легче понять, почему файл отвечает критериям поиска), а также может выводиться любая

дополнительная отладочная информация. Переменные окружения, которые должны

поддерживаться программой, приведены в Табл. 2.

Имя программы должно начинаться на lab11, далее должен следовать уникальный для

варианта суффикс. Уникальный суффикс составляется из первых букв имени, отчества (если

есть) и фамилии студента, выполняющего лабораторную работу. Далее следует номер

группы студента. Используются строчные латинские буквы и арабские (в традиционном

понимании, т. е. 0..9) цифры. Например, если студента, выполняющего лабораторную, зовут

Петр Сергеевич Иванов, его группа — N32451, то имя программы должно быть

lab11psiN32451.

Проект (исходные коды, заголовочные файлы, Makefile и прочие файлы, необходимые

для сборки) должен содержаться в отдельном каталоге с именем, совпадающим с названием

программы (lab11abcNXXXXX) и собираться с помощью стандартной утилиты make.

Исходные файлы программы на языке C должны компилироваться с помощью gcc. Makefile

должен поддерживать как минимум цели all и clean. Если для сборки проекта требуется что-

то большее, чем make all, или для запуска и проверки проекта требуются какие-либо

нетривиальные или неочевидные действия, то инструкции по сборке и запуску проекта

следует добавить

# Makefile

all: main.c

gcc -c main.c -o lab11araN3250.o -Wall -Wextra -Werror -O3 -g

gcc lab11araN3250.o -o lab11araN3250

clean:

rm -f lab11araN3250.o

rm -f lab11araN3250

# Отчет valgrind:

==7137== HEAP SUMMARY:

==7137== in use at exit: 0 bytes in 0 blocks

==7137== total heap usage: 12 allocs, 12 frees, 37,161 bytes allocated

==7137==

==7137== All heap blocks were freed -- no leaks are possible

==7137==

==7137== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

# Скриншоты работы:

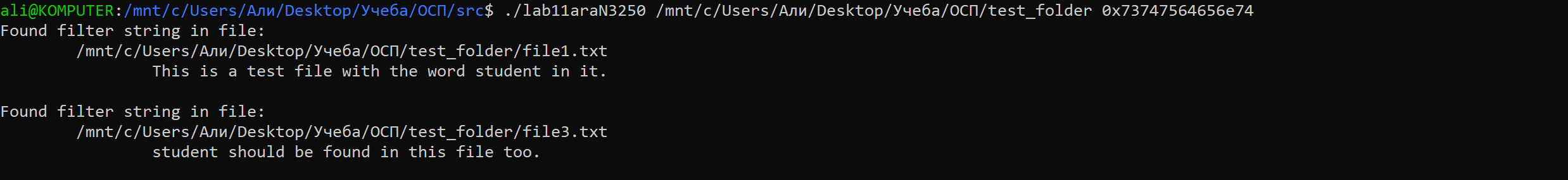


Рис.1 – скриншот работы программы

# Исходные тексты программ с комментариями.

#define \_XOPEN\_SOURCE 500

#define \_DEFAULT\_SOURCE

#include <stdlib.h>

#include <stdio.h>

#include <getopt.h>

#include <unistd.h>

#include <stdint.h>

#include <limits.h>

#include <dirent.h>

#include <string.h>

// Переменная для хранения режима отладки

char\* debug;

// Переменная для хранения искомой последовательности байтов

char\* desired\_sequence;

// Определение долгих опций для getopt\_long

struct option long\_opts[] = {

{"help", no\_argument, NULL, 'h'},

{"version", no\_argument, NULL, 'v'}

};

// Определение коротких опций для getopt\_long

const char\* opts = "hv";

// Функция для проверки файла на наличие искомой последовательности байтов

void check\_file(char\* file){

FILE\* fp;

size\_t len = 0;

char\* line = NULL;

ssize\_t read;

// Открываем файл для чтения

fp = fopen(file, "r");

if(!fp){

// Выводим сообщение об ошибке, если файл не удалось открыть

fprintf(stderr, "Bad file: %s\n", file);

return;

}

int found = 0;

// Читаем файл построчно

while((read = getline(&line, &len, fp)) != -1){

if(debug){

// Выводим отладочную информацию, если включен режим отладки

fprintf(stderr, "Read line:\n\t%s\n", line);

}

// Проверяем, содержит ли строка искомую последовательность байтов

found = strstr(line, desired\_sequence) != NULL ? 1 : 0;

if(found){

break;

}

}

if(found){

// Если последовательность найдена, выводим информацию об этом

printf("Found filter string in file:\n\t%s\n\t\t%s\n", file, line);

}

// Освобождаем память, выделенную для строки

if(line){

free(line);

}

// Закрываем файл

fclose(fp);

}

// Функция для рекурсивного обхода директорий

void explore(char\* current\_dir){

struct dirent\* dir;

DIR\* d;

// Открываем директорию

d = opendir(current\_dir);

if(!d){

// Выводим сообщение об ошибке, если директорию не удалось открыть

fprintf(stderr, "Bad dir: %s\n", current\_dir);

return;

}

char buf[PATH\_MAX];

// Читаем содержимое директории

while((dir = readdir(d)) != NULL){

// Получаем полный путь к файлу

for(int i = 0; i < PATH\_MAX; i++){

buf[i] = '\0';

}

strcat(buf, current\_dir);

if(buf[strlen(buf) - 1] != '/'){

strcat(buf, "/");

}

strcat(buf, dir->d\_name);

if(debug){

// Выводим отладочную информацию, если включен режим отладки

fprintf(stderr, "Current dir:\n\t%s\n", buf);

}

// Если это файл, проверяем его на наличие искомой последовательности

if(dir->d\_type == DT\_REG){

check\_file(buf);

}

// Если это директория, рекурсивно обходим ее

else if(dir->d\_type == DT\_DIR && dir->d\_name[0] != '.'){

explore(buf);

}

}

// Закрываем директорию

closedir(d);

}

int main(int argc, char\* argv[]){

// Получаем значение переменной окружения LAB11DEBUG

debug = getenv("LAB11DEBUG");

int opt;

int option\_ind = -1;

// Обрабатываем аргументы командной строки

while((opt = getopt\_long(argc, argv, opts, long\_opts, &option\_ind)) != -1){

switch (opt)

{

case 'v':

// Выводим версию программы и информацию о разработчике

printf("lab11araN3250 version 1.\nAhrarov Ali Rustamovich N3250\n2-nd variant\n");

return 0;

case 'h':

// Выводим справочную информацию

printf("Usage: ./lab11araN3250 [options...] <dir path> <bytes> \nOptions:\n%s: %s\n%s: %s\n",

"-h, --help",

"Get help for commands",

"-v, --version",

"Show info"

);

return 0;

}

option\_ind = -1;

}

// Проверяем количество аргументов

if(argc < 3){

fprintf(stderr, "%s\n%s\n", "Few arguments", "Usage: ./lab11araN3250 [options...] <dir\_path> <bytes>");

exit(EXIT\_FAILURE);

}

char\* bytes = argv[2];

int len = strlen(bytes);

// Проверяем правильность формата аргумента "bytes"

if(len < 4 || bytes[0] != '0' || bytes[1] != 'x'){

fprintf(stderr, "Mimimal byte sequency looks like 0xhh\n");

exit(EXIT\_FAILURE);

}

// Количество символов в последовательности должно быть четным

if(len % 2 != 0){

fprintf(stderr, "Byte sequency has to have even count of bytes\n");

exit(EXIT\_FAILURE);

}

// Проверяем, что последовательность состоит из шестнадцатеричных цифр

for(int i = 2; i < len; i++){

if(!(('a' <= bytes[i] && bytes[i] <= 'f') || ('A' <= bytes[i] && bytes[i] <= 'F') ||

('0' <= bytes[i] && bytes[i] <= '9'))){

fprintf(stderr, "Byte sequency has to consist of hex numbers\n");

exit(EXIT\_FAILURE);

}

}

// Преобразуем последовательность байтов в строку

char buf[3];

desired\_sequence = malloc(sizeof(char) \* (len / 2 + 1));

for(int i = 0; i < (int)sizeof(char) \* (len / 2 + 1); i++){

desired\_sequence[i] = '\0';

}

for(int i = 2; i < len; i += 2){

for(int j = 0; j < 2; j++){

buf[j] = bytes[i + j];

}

buf[2] = '\0';

desired\_sequence[(i - 2) / 2] = (char)((int) strtol(buf, NULL, 16));

}

desired\_sequence[len / 2] = '\0';

if(debug){

// Выводим отладочную информацию, если включен режим отладки

fprintf(stderr, "Got string:\n\t%s\n", desired\_sequence);

}

char\* current\_dir = argv[1];

// Рекурсивно обходим директории, начиная с указанной

explore(current\_dir);

// Освобождаем память

free(desired\_sequence);

return 0;

}