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

**Факультет безопасности информационных технологий**

**Дисциплина:**

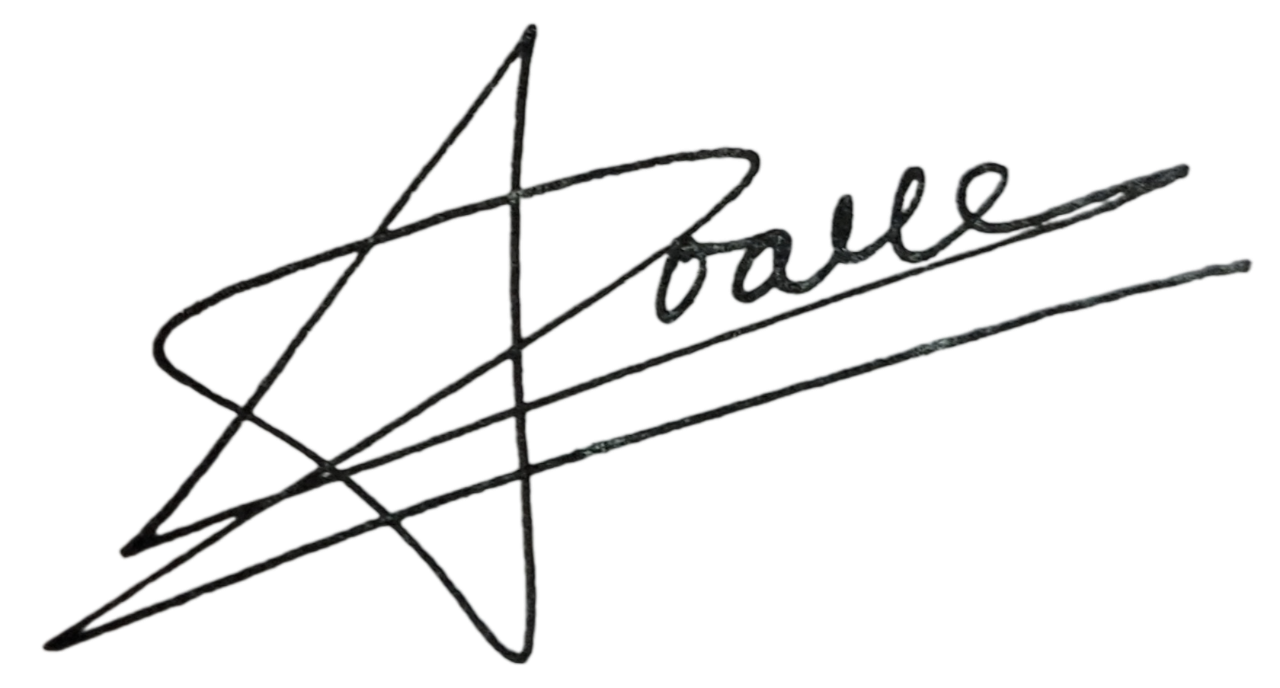
«Основы системного программирования»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1.1**

«Работа с файлами и каталогами»

**Выполнил:**

Чу Ван Доан, студент группы N3247

**

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

(подпись)

**Проверила:**

Грозов В.А., преподаватель ФБИТ

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

(отметка о выполнении)

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

(подпись)

**Содержание**

[**1**. Задание 3](#_heading=h.m7zgjkm3dxis)

[2. Ход работы 7](#_heading=h.81hq1duwme14)

[3. Make-файл 9](#_heading=h.h2hivfbqltq)

[4. Отчёт valgrind cо строчки «HEAP SUMMARY:» 10](#_heading=h.o7zop0ijaydx)

[5. Примеры работы программ (скриншоты) 11](#_heading=h.81un5s7kfkwi)

[6. Исходные тексты программ](#_heading=h.7fwje3slemeq)12

# 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 в файл README.txt в формате plain text и разместить его в каталоге проекта.

Порядок выполнения и сдачи лабораторной работы:

1. Подготовить тестовый набор файлов (отвечающих и неотвечающих критериям поиска), разместить их в тестовом дереве каталогов.
2. Выполнить задание, подготовить все файлы проекта, скомпилировать программу и библиотеку с флагами -Wall -Wextra -Werror и устранить все предупреждения и ошибки.
3. Протестировать программу на различных каталогах, убедиться, что ошибок нет, в противном случае вернуться к пункту 2.
4. Выполнить запуск программы (поиск файлов в тестовом дереве каталогов) под управлением valgrind, убедиться, что утечки памяти отсутствуют. Если утечки есть, то сначала устранить их и вернуться к пункту 2. Если утечек нет, то сохранить отчет в файл valgrind.txt и добавить его в каталог проекта. Скомпилировать программу с флагом -O3, повторно протестировать программу поиском в различных каталогах. В случае обнаружения ошибок или предупреждений вернуться к пункту 2.
5. Удалить все исполняемые и промежуточные файлы из папки проекта (make clean). В архиве должны остаться только файлы \*.c, \*.h, Makefile, README.txt, valgrind.txt.
6. Заархивировать папку проекта, используя формат tar.gz. (tar -czvf lab11abcNXXXXX.tar.gz lab11abcNXXXXX/).
7. Отправить полученный архив на почту преподавателя, который ведет лабораторные (Гирику А.В. на [itmo.osp@gmail.com](mailto:itmo.osp.2021@gmail.com), Горлиной А.В. на [gorlina.a.v@mail.ru](mailto:gorlina.a.v@mail.ru), Грозову В.А. на [va\_groz@mail.ru](mailto:va_groz@mail.ru)), письмом с темой «*ОСП ЛР11 Фамилия Имя Отчество NXXXXX*».
8. Дождаться ответа по почте или на лабораторном занятии, устранить возможные замечания (повторить с пункта 1).
9. Получить подтверждение от преподавателя, что лабораторная работа выполнена успешно, после чего подготовить отчет в электронной форме (состав отчета см. ниже).
10. Отправить архив с окончательным вариантом проекта и отчетом в формате pdf (не забыть про подпись на первой странице!) на почту преподавателя письмом с темой

«*ОСП Отчет по ЛР11 Фамилия Имя Отчество NXXXXX*». Файл отчета должен иметь название NXXXXX\_ФамилияИО\_ЛР11.pdf.

1. Получить некоторое количество вопросов по отчету от преподавателя и дать на них ответы (а может и не получить, если лабораторная выполнена на хорошем уровне и сомнений в знаниях студента у преподавателя не возникает). Получить от преподавателя подтверждение, что отчет принят.
2. Немного расслабиться и приступить к следующей лабораторной работе :-)

Отчет должен быть подготовлен в формате pdf и содержать:

* + правильно оформленную титульную страницу (с подписью студента);
  + задание;
  + Make-файл;
  + отчет valgrind cо строчки «HEAP SUMMARY:»
  + примеры работы программ (скриншоты);
  + исходные тексты программ с комментариями.

**Замечание 1**. При выполнении лабораторной работы следует использовать функции стандартной библиотеки С и системные вызовы операционной системы. Использовать С++, а следовательно, ввод-вывод в стиле С++ (классы ifstream/ofstream/…), контейнеры и алгоритмы STL (<string>, <vector>, <map>, ...) запрещено.

**Замечание 2**. В программах должна присутствовать обработка ошибок: в случаях, если пользователь задал неверную комбинацию опций, указал файлы, которые невозможно открыть, и т.д. программа должна выдавать диагностическое сообщение на консоль (в стандартный поток ошибок и/или лог-файл), прежде чем завершиться.

**Замечание 3**. При обходе дерева каталогов нужно учитывать, что доступ к некоторым файлам и каталогам может завершиться с ошибкой (например, по причине отсутствия прав доступа и т.д.). В таком случае следует вывести сообщение об ошибке в стандартный поток ошибок и продолжить обход. Для упрощения реализации обхода символические ссылки (и аналогичные средства — жесткие ссылки, bind mount'ы и т.д.) можно игнорировать.

**Замечание 4**. Категорически запрещается использовать статические массивы (с размерами, заданными на этапе компиляции) для любых данных, размер которых зависит от входных данных или условий запуска. Для хранения таких данных необходимо использовать динамическую память и определять объем необходимой памяти в зависимости от ситуации. Статические массивы можно использовать в тех ситуациях, когда известен максимальный размер обрабатываемых данных (и он не превышает размеров стека или максимального размера статических массивов, допускаемого компилятором).

**Замечание 5**. Информационные сообщения выводятся программой в стандартный поток вывода, сообщения об ошибках — в стандартный поток ошибок. С помощью определения переменной окружения LAB11DEBUG можно включить вывод отладочных сообщений программой в стандартный поток ошибок.

**Замечание 6**. Для обработки опций командной строки рекомендуется использовать функцию getopt\_long(). При необходимости список опций можно расширить.

**Замечание 7**. Несмотря на то, что для компиляции программ необходимо использовать компилятор gcc, использования расширений GNU C желательно по возможности избегать и ориентироваться на использование стандарта C11 или более позднего.

**Замечание 8**. Программа должна успешно компилироваться и выполняться в ОС Linux с ядром версии >= 5.0. По возможности следует избегать специфичных для Linux и glibc функций и системных вызовов и стремиться к соответствию стандарту POSIX. При желании и возможности рекомендуется проверить работу программы не только на платформе x86\_64 GNU/Linux, но и на других POSIX платформах: x86/ARM/... + \*BSD/macOS/...

**Таблица 1. Опции командной строки, поддерживаемые программой**

| **Опция** | **Назначение** |
| --- | --- |
| -v, --version | Вывод версии программы и информации о программе (ФИО исполнителя, номер группы, номер варианта лабораторной). |
| -h, --help | Вывод справки по опциям. |

**Таблица 2. Переменные окружения, поддерживаемые программой**

| **Переменная** | **Назначение** |
| --- | --- |
| LAB11DEBUG | Включение вывода отладочной информации. |

**Таблица 3. Формат аргумента цель\_поиска**

| **Вариант (условие)** | **Назначение** |
| --- | --- |
| Количество букв в **фамилии** студента, выполняющего работу, **нечетное**. | Выполняется поиск заданной последовательности байтов. Аргумент цель\_поиска имеет формат 0xhh[hh\*], где hh — две шестнадцатеричных цифры (число должно начинаться с префикса 0x).  *Пример запуска программы*:  ./lab11psiN32451 /home 0xc0ffee  (Начиная с каталога /home, рекурсивно выполняется поиск файлов, содержащих последовательность байтов 0xc0 (192), 0xff (255) и 0xee (238).) |
| Количество букв в **фамилии** студента, выполняющего работу, **четное**. | Выполняется поиск заданной последовательности байтов. Аргумент цель\_поиска имеет формат строки в кодировке UTF-8.  *Пример запуска программы*:  ./lab11psiN32451 /home "лавандовый раф"  (Начиная с каталога /home, рекурсивно выполняется поиск файлов, содержащих последовательность байтов 0xd0 0xbb ('л'), 0xd0 0xb0 ('а'), ..., 0xd1 0x84('ф').) |

**Таблица 4. Функции, которые необходимо использовать для реализации поиска файлов и каталогов**

| **Вариант (номер)** | **Функции** |
| --- | --- |
| 1 | opendir()/readdir() без рекурсии |
| 2 | opendir()/readdir() с рекурсией |
| 3 | ftw() |
| 4 | nftw() |
| 5 | fts\_open()/fts\_read()/... |

# Ход работы

1. Вариант 1: opendir()/readdir() без рекурсии - Выполняется поиск заданной последовательности байтов. Аргумент цель\_поиска имеет формат строки в кодировке UTF-8. Пример запуска программы: ./lab11psiN32451 /home "лавандовый раф" (Начиная с каталога /home, рекурсивно выполняется поиск файлов, содержащих последовательность байтов 0xd0 0xbb ('л'), 0xd0 0xb0 ('а'), ..., 0xd1 0x84('ф').)
2. Подготовка среды:

Убедитесь, что у вас установлены компилятор GCC и Valgrind. Это можно сделать следующими командами

gcc --version

valgrind --version

1. Компиляция программы:

* Перейдите в директорию, содержащую исходные файлы программы. Предположим, что вы находитесь в папке, где находятся файлы main.c и Makefile.
* Скомпилируйте программу, используя следующий Makefile. В терминале выполните: make
* Makefile содержит правила компиляции и сборки исполняемого файла lab11cvdN3247.

1. Запуск программы:

После успешной компиляции программа может быть запущена с различными параметрами. Примеры запуска:

./lab11cvdN3247 /home/chu/Desktop/ 3

./lab11cvdN3247 /lib/ ab

1. Отладка программы:

* Для включения режима отладки вы можете использовать препроцессорную директиву LAB11DEBUG. Это можно сделать следующим образом:

gcc -DLAB11DEBUG -o lab11cvdN3247 main.

* После компиляции с отладочной информацией, запустите программу с переменной окружения LAB11DEBUG, чтобы включить вывод отладочной информации:

LAB11DEBUG=1 ./lab11cvdN3247 /home

1. Проверка ошибок памяти с помощью Valgrind:

Для проверки программы на наличие ошибок памяти используйте Valgrind. Запустите программу через Valgrind следующей командой:

valgrind ./lab11cvdN3247 /home/chu/Downloads/ anhngoc

# Make-файл

CC = gcc

CFLAGS = -Wall -Wextra -Werror

TARGET = lab11cvdN3247

.PHONY: all clean

all: $(TARGET)

$(TARGET): main.o

$(CC) $(CFLAGS) -o $@ $^

main.o: main.c

$(CC) $(CFLAGS) -c -o $@ $<

clean:

rm -f $(TARGET) \*.o

# Отчёт valgrind cо строчки «HEAP SUMMARY:»

==33900== Memcheck, a memory error detector

==33900== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==33900== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info

==33900== Command: ./lab11cvdN3247 /home/chu/Downloads/ anhngoc

==33900==

==33900== Warning: set address range perms: large range [0x59c87040, 0x18e2d5841) (undefined)

==33900== Warning: set address range perms: large range [0x59c87040, 0xd9c86040) (defined)

==33900== Warning: set address range perms: large range [0xd9c86040, 0x159c85040) (defined)

==33900== Warning: set address range perms: large range [0x159c85040, 0x18e2d5040) (defined)

==33900== Warning: set address range perms: large range [0x59c87028, 0x18e2d5859) (noaccess)

==33900== Warning: set address range perms: large range [0x59c87040, 0x15649f041) (undefined)

==33900== Warning: set address range perms: large range [0x59c87040, 0xd9c86040) (defined)

==33900== Warning: set address range perms: large range [0xd9c86040, 0x15649f040) (defined)

==33900== Warning: set address range perms: large range [0x59c87028, 0x15649f059) (noaccess)

==33900== Warning: set address range perms: large range [0x59c87040, 0xe1a0e3f9) (undefined)

==33900== Warning: set address range perms: large range [0x59c87040, 0xd9c86040) (defined)

==33900== Warning: set address range perms: large range [0x59c87028, 0xe1a0e411) (noaccess)

==33900== Warning: set address range perms: large range [0x6ecb040, 0x255671b6) (undefined)

==33900== Warning: set address range perms: large range [0x6ecb040, 0x25567040) (defined)

==33900== Warning: set address range perms: large range [0x6ecb028, 0x255671ce) (noaccess)

==33900== Warning: set address range perms: large range [0x6ecb040, 0x19d3ca2e) (undefined)

==33900== Warning: set address range perms: large range [0x6ecb040, 0x19d3c040) (defined)

==33900== Warning: set address range perms: large range [0x6ecb028, 0x19d3ca46) (noaccess)

==33900== Warning: set address range perms: large range [0x6ecb040, 0x26b49d96) (undefined)

==33900== Warning: set address range perms: large range [0x6ecb040, 0x26b49040) (defined)

==33900== Warning: set address range perms: large range [0x6ecb028, 0x26b49dae) (noaccess)

==33900==

==33900== HEAP SUMMARY:

==33900== in use at exit: 0 bytes in 0 blocks

==33900== total heap usage: 2,486 allocs, 2,486 frees, 13,937,580,585 bytes allocated

==33900==

==33900== All heap blocks were freed -- no leaks are possible

==33900==

==33900== For lists of detected and suppressed errors, rerun with: -s

==33900== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

# Примеры работы программ (скриншоты)

1. Каталог home



1. Каталог version



1. Каталог help

# Исходные тексты программ

#include <stdio.h> // Библиотека стандартного ввода/вывода, используется для функций как printf, fopen, fclose и т.д.

#include <stdlib.h> // Стандартная библиотека для функций как malloc, free, exit и т.д.

#include <string.h> // Библиотека для работы со строками, например, strcpy, strlen, strstr и т.д.

#include <dirent.h> // Библиотека для работы с директориями, например, opendir, readdir, closedir и т.д.

#include <sys/stat.h> // Библиотека для работы с файловой системой, например, struct stat, stat и т.д.

#include <unistd.h> // Библиотека POSIX, например, для getopt, getopt\_long и т.д.

#include <getopt.h> // Библиотека для функции getopt\_long, используемой для разбора опций командной строки

#define MAX\_PATH\_LENGTH 1024 // Определение максимальной длины пути

// Функция для проверки, содержит ли файл определенную последовательность байтов

int file\_contains(const char \*filename, const char \*search\_target) {

FILE \*file = fopen(filename, "rb"); // Открыть файл в режиме чтения в бинарном формате

if (!file) { // Проверка, удалось ли открыть файл

perror("Error opening file"); // Вывод сообщения об ошибке

return 0; // Возвращает 0 в случае ошибки

}

// Получение размера файла

fseek(file, 0, SEEK\_END); // Перемещение указателя файла в конец

long file\_size = ftell(file); // Определение текущей позиции указателя, т.е. размера файла

fseek(file, 0, SEEK\_SET); // Перемещение указателя файла в начало

// Выделение памяти для хранения содержимого файла

char \*file\_content = (char \*)malloc((size\_t)file\_size + 1); // Добавление 1 байта для завершающего нулевого символа

if (!file\_content) { // Проверка, удалось ли выделить память

fclose(file); // Закрытие файла

perror("Error allocating memory"); // Вывод сообщения об ошибке

return 0; // Возвращает 0 в случае ошибки

}

// Чтение содержимого файла

if (fread(file\_content, 1, (size\_t)file\_size, file) != (size\_t)file\_size) { // Чтение содержимого файла в память

fclose(file); // Закрытие файла

free(file\_content); // Освобождение выделенной памяти

perror("Error reading file"); // Вывод сообщения об ошибке

return 0; // Возвращает 0 в случае ошибки

}

file\_content[file\_size] = '\0'; // Добавление завершающего нулевого символа

fclose(file); // Закрытие файла

// Поиск целевой последовательности в содержимом файла

char \*found = strstr(file\_content, search\_target); // Поиск строки search\_target в file\_content

free(file\_content); // Освобождение выделенной памяти

return (found != NULL); // Возвращает 1, если найдено, иначе возвращает 0

}

// Функция для рекурсивного поиска файлов в директории

void search\_directory(const char \*path, const char \*search\_target) {

DIR \*dir;

struct dirent \*entry; // Структура для хранения информации о файле/папке в директории

char full\_path[MAX\_PATH\_LENGTH]; // Строка для хранения полного пути

dir = opendir(path); // Открытие директории

if (!dir) { // Проверка, удалось ли открыть директорию

perror("Error opening directory"); // Вывод сообщения об ошибке

return; // Выход из функции в случае ошибки

}

while ((entry = readdir(dir)) != NULL) { // Чтение каждого элемента в директории

if (strcmp(entry->d\_name, ".") == 0 || strcmp(entry->d\_name, "..") == 0) // Пропуск специальных директорий "." и ".."

continue;

// Формирование полного пути

snprintf(full\_path, MAX\_PATH\_LENGTH, "%s/%s", path, entry->d\_name);

struct stat statbuf; // Структура для хранения информации о файле/директории

if (stat(full\_path, &statbuf) == -1) { // Получение информации о файле/директории

perror("Error stating file"); // Вывод сообщения об ошибке

continue; // Пропуск этого элемента в случае ошибки

}

if (S\_ISDIR(statbuf.st\_mode)) { // Проверка, является ли это директорией

search\_directory(full\_path, search\_target); // Рекурсивный поиск в поддиректории

} else { // Если это файл

if (file\_contains(full\_path, search\_target)) { // Проверка, содержит ли файл целевую строку

printf("%s\n", full\_path); // Вывод полного пути к файлу

}

}

}

closedir(dir); // Закрытие директории

}

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

int opt;

int help\_flag = 0, version\_flag = 0;

// Разбор опций командной строки

static struct option long\_options[] = {

{"help", no\_argument, 0, 'h'}, // Опция помощи

{"version", no\_argument, 0, 'v'}, // Опция версии

{0, 0, 0, 0} // Завершение списка опций

};

while ((opt = getopt\_long(argc, argv, "hv", long\_options, NULL)) != -1) { // Получение опций командной строки

switch (opt) {

case 'h':

help\_flag = 1; // Установка флага помощи

break;

case 'v':

version\_flag = 1; // Установка флага версии

break;

default:

fprintf(stderr, "Usage: %s [OPTION]... DIRECTORY SEARCH\_TARGET\n", argv[0]); // Вывод использования

exit(EXIT\_FAILURE); // Завершение программы с ошибкой

}

}

if (help\_flag) { // Проверка флага помощи

printf("Help information...\n"); // Вывод информации о помощи

exit(EXIT\_SUCCESS); // Завершение программы успешно

}

if (version\_flag) { // Проверка флага версии

printf("Version information: Lab11cvdN3247\n"); // Вывод информации о версии

exit(EXIT\_SUCCESS); // Завершение программы успешно

}

if (optind >= argc - 1) { // Проверка, есть ли достаточно аргументов

fprintf(stderr, "Usage: %s [OPTION]... DIRECTORY SEARCH\_TARGET\n", argv[0]); // Вывод использования

exit(EXIT\_FAILURE); // Завершение программы с ошибкой

}

const char \*directory = argv[optind]; // Получение аргумента директории

const char \*search\_target = argv[optind + 1]; // Получение аргумента целевой строки

// Выполнение рекурсивного поиска в указанной директории

search\_directory(directory, search\_target);

return 0; // Завершение программы успешно

}