Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №3

на тему

**ОСНОВЫ ПРОГРАММИРОВАНИЯ НА C ПОД UNIX. ИНТСРУМЕНТАРИЙ ПРОРАММИСТА В UNIX**

Выполнил: студент гр.253504 Лавренова А.С.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2025

СОДЕРЖАНИЕ

Введение3

[1 [Ц](#_agswffz6g6ei)ель работы](#_Toc178067889) 4

[2 Теоретические сведения 5](#_Toc178067890)

[3 Описание и пример выполнения программы 6](#_Toc178067891)

Заключение [9](#_Toc178067899)

[Список использованных источников 10](#_Toc178067900)

[Приложение А (обязательное) 11](#_Toc178067901)

# ВВЕДЕНИЕ

Программирование на языке C в среде операционных систем семейства Unix занимает центральное место в подготовке специалистов в области системного и прикладного программного обеспечения. Эта лабораторная работа направлена на глубокое освоение ключевых аспектов программной инженерии, начиная от базовых конструкций языка и заканчивая практическим применением инструментов, необходимых для реальной разработки. Особое внимание уделяется использованию компилятора gcc и системы управления сборкой make, которые являются стандартом де-факто в среде Unix. Через эти инструменты студенты получают возможность не только компилировать и собирать программы, но и формировать структуру проекта, автоматизировать этапы тестирования, а также управлять зависимостями и конфигурациями.

Основным предметом исследования в рамках этой работы является разработка программ-фильтров, выполняющих преобразование текста в соответствии с азбукой Морзе и обратную операцию декодирования. Такие преобразования требуют не только точного знания соответствий между символами и кодами, но и умения строить надёжные алгоритмы, устойчивые к ошибкам и некорректным входным данным. Реализация фильтров должна учитывать особенности обработки текста, включая ситуацию, когда отдельные символы не имеют представления в таблице Морзе, а также случаи, когда на вход подаются некорректные или неизвестные комбинации кодов. Для пользователя такие ситуации должны сопровождаться понятными и подробными предупреждениями, отражающими суть проблемы, что обеспечивает прозрачность и удобство взаимодействия.

Отдельный интерес представляет организация кода в виде многомодульной архитектуры. Такой подход способствует лучшему структурированию программы, облегчает повторное использование компонентов и повышает читаемость. При этом makefile выступает в роли управляющего механизма, который не только автоматизирует процесс сборки, но и позволяет производить очистку, повторную компиляцию и запуск тестов в упрощённой форме. Весь проект ориентирован на соблюдение принципов модульности, масштабируемости и надёжности.

Таким образом, данная лабораторная работа охватывает не только синтаксические и компиляторные аспекты языка C, но и формирует фундаментальные практические навыки построения, отладки и тестирования программных решений в условиях, приближённых к реальной разработке.

1 ЦЕЛЬ РАБОТЫ

Цель данной лабораторной работы заключается в глубоком изучении среды программирования на языке C в операционных системах Unix, а также освоении ключевых инструментов, таких как компилятор gcc и система управления сборкой make. В процессе работы студенты получат практический опыт в использовании основных библиотек и системных вызовов, что включает в себя операции ввода-вывода, работу с файлами, текстовую обработку, распределение памяти и управление выполнением программ.

Задача состоит в разработке программы, соответствующей заданному варианту — реализации текстового фильтра, выполняющего преобразование текста в азбуку Морзе (прямой фильтр) и обратно (обратный фильтр). Программа должна предоставлять пользователю возможность выбора режима работы (шифрование или дешифрование), а также источника входного текста — ручной ввод через терминал или чтение из заранее подготовленного текстового файла. После завершения преобразования должна быть реализована дополнительная возможность автоматического восстановления исходного текста через обратное преобразование.

Особое внимание уделяется точной реализации требований, характерных для фильтров: в прямом фильтре непреобразуемые символы должны отбрасываться с выводом соответствующих предупреждений в терминал, тогда как в обратном фильтре нераспознанные комбинации азбуки Морзе должны заменяться специальной пометкой в выходном потоке и сопровождаться информативными сообщениями о возникших проблемах. Таким образом, обеспечивается прозрачная диагностика и контроль обработки текста.

С программной точки зрения, проект должен быть реализован в виде многомодульной структуры (головной модуль и 1–2 вспомогательных), с поддержкой управления через makefile. Последний обязан включать цели по сборке, очистке проекта от промежуточных файлов, а также автоматизированному тестированию с использованием подготовленных входных данных и сравнением результатов. Такая постановка задачи обеспечивает практическое закрепление навыков по построению, сборке, отладке и тестированию программных проектов в Unix-среде, а также углубляет понимание принципов обработки текстовой информации и взаимодействия с пользователем через консоль.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Linux — это свободная операционная система (точнее, семейство систем) с открытым исходным кодом. Само название Linux относится к ядру системы, которое является ее ключевым компонентом, управляющим центральным процессором (ЦП), памятью и периферийными устройствами компьютера. [1]

Разработка на C под Linux — это мощный способ создания высокопроизводительных и эффективных приложений. Язык C является основой для многих операционных систем, включая Linux, и предоставляет разработчикам низкоуровневый доступ к аппаратуре и системным ресурсам.

Для разработки на C под Linux понадобится компилятор. GCC является стандартным компилятором для большинства дистрибутивов Linux и поддерживает множество языков программирования, включая C и C++. [2]

Система make + Makefiles используется для того, чтобы помочь с принятием решения, какие части большой программы следует перекомпилировать, а какие пока не нужно. Если тупо перекомпилировать все заново, то это может занять много времени. В подавляющем большинстве случаев это используется для компиляции файлов кода на языках C или C++. Другие языки имеют собственные инструменты, которые делают примерно то же самое, что и make. Make может также использоваться и по завершению компиляции, когда требуется выполнить ряд инструкций в зависимости от того, какие файлы поменялись в процессе компиляции.

Удивительно, но make (1976) имеет более долгую историю, чем язык C++ (1978). Он был разработан как инструмент для создания программного обеспечения, не привязанный к какому-либо конкретному языку. Несмотря на свой возраст, он до сих пор широко используется, так как в нем внедрены эффективные технологии, влияющие на другие современные инструменты автоматизации сборки.

CMake технически не является инструментом сборки (который должен использоваться как make), а скорее генератором системы сборки (он же инструмент мета-сборки). Он используется в сочетании с нативными средами сборки, такими как Make, Ninja, MSBuild и т. д., и разработан как кроссплатформенный (Linux, Windows, macOS и т. д.) и независимый от компилятора (GCC, Clang, MSVC и т. д.). [3]

В заключение, Linux представляет собой мощную платформу для разработки благодаря открытости и возможностям низкоуровневого программирования на языке C. Использование компилятора GCC и инструментов управления сборкой, таких как make и CMake, значительно упрощает процесс создания и оптимизации программного обеспечения. Эти инструменты не только ускоряют компиляцию, но и обеспечивают кроссплатформенность, что делает их незаменимыми в современном программировании. Таким образом, изучение этих технологий является важным шагом для разработчиков, стремящихся создавать эффективные и производительные приложения.

**3 ОПИСАНИЕ И ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ**

Разработанная программа представляет собой универсальный текстовый фильтр, реализующий как прямое, так и обратное преобразование текста в соответствии с кодом Морзе. Её выполнение осуществляется в среде операционной системы Linux с использованием компилятора gcc и системы управления сборкой make. Программа позволяет пользователю выбрать режим работы: преобразование печатного текста в код Морзе либо обратную операцию декодирования Морзе-кода в читаемый текст. В дополнение к этому предусмотрен выбор источника данных: текст может быть введён вручную через терминал или считан из заранее подготовленного текстового файла.

После запуска программы пользователь видит интерактивное меню, которое предлагает выбрать режим работы. Если пользователь выбирает шифрование, программа переходит к следующему шагу — выбору способа ввода текста. Здесь доступен выбор между ручным вводом и чтением из файла testinput.txt. При чтении из файла программа автоматически открывает файл, считывает его содержимое и отображает промежуточную информацию в терминале. После этого текст преобразуется в код Морзе, при этом все непреобразуемые символы (например, неподдерживаемые знаки препинания или спецсимволы) отбрасываются. Каждое такое действие сопровождается подробным предупреждением в терминале, где указывается символ, его позиция и причина отбрасывания. Первая часть работы программы отображена на рисунке 3.1.

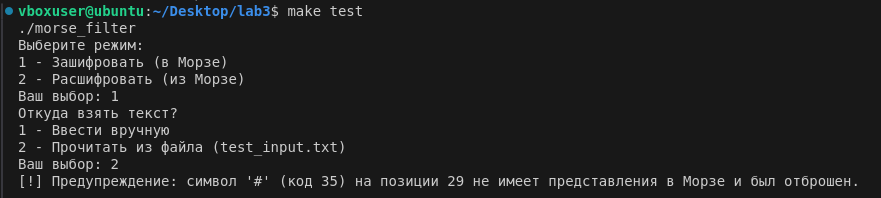


Рисунок 3.1 – Предупреждение об отброшенных символах

После завершения шифрования пользователь получает уведомление о сохранении результата в файл test\_output.txt. Этот файл может быть просмотрен с помощью утилиты cat, либо пользователь может выбрать в программе режим обратного преобразования. В случае выбора расшифровки программа вновь предложит указать способ ввода данных, где в типовом сценарии выбирается прочтение из файла, содержащего код Морзе. При расшифровке осуществляется последовательный разбор комбинаций символов и попытка сопоставления их с таблицей кодов. Если программа встречает нераспознанную комбинацию, она выводит в терминал предупреждение с описанием проблемы, а в выходной поток помещает специальную пометку [???], чтобы визуально отметить ошибку (рисунок 3.2).

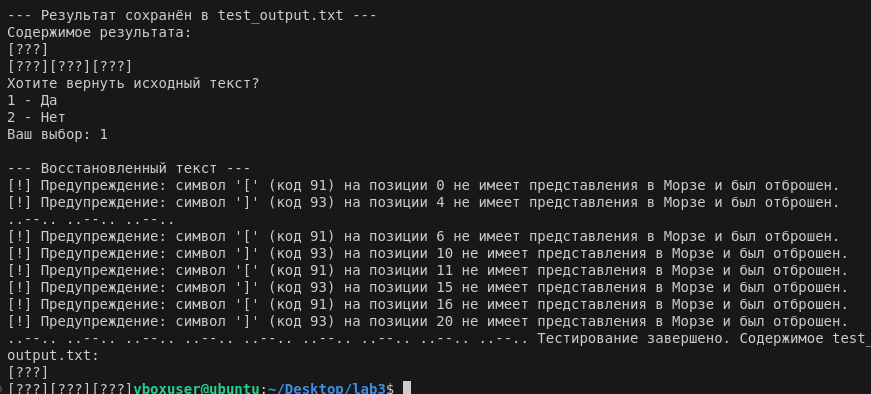


Рисунок 3.2 – Вывод предупреждений об ошибочных комбинациях

Одной из особенностей реализованной логики является автоматизированная проверка корректности шифрования и обратного преобразования. После завершения основной операции пользователю предлагается восстановить исходный текст, выполнив обратную операцию. Это позволяет убедиться в симметричности фильтра и проверить, не произошло ли искажение данных. Результат восстанавливается не путём копирования оригинального текста, а исключительно через соответствующее преобразование, что подтверждает корректность алгоритма.

Одним из ключевых требований к программам-фильтрам в рамках данной лабораторной работы является реализация возможности явного указания файлов ввода и вывода, а также режима работы (шифрование или дешифровка) через параметры командной строки. Это позволяет использовать программу не только в интерактивном режиме, но и в составе автоматических скриптов или при тестировании, где необходимо минимизировать участие пользователя.

В разработанной программе поддерживается гибкий механизм обработки аргументов командной строки, что делает её более универсальной. Если программа запускается без параметров, она переходит в диалоговый (интерактивный) режим, запрашивая у пользователя режим работы и способ ввода. Однако при указании параметров -e или -d поведение программы меняется — она выполняет соответствующее преобразование без необходимости взаимодействия через терминал.

Передача файлов осуществляется с помощью параметров -i и -o, где -i указывает путь к входному файлу, а -o — путь к выходному. Программа проверяет, были ли эти аргументы действительно переданы, и в случае их отсутствия сообщает об ошибке, выводит краткую инструкцию по использованию и завершает выполнение. Такой подход обеспечивает надёжность и предсказуемость поведения, особенно при автоматическом запуске.

На рисунке 3.3 будет продемонстрирован запуск программы с параметрами командной строки, отсутствие интерактивного ввода и финальное сообщение об успешном сохранении результата.

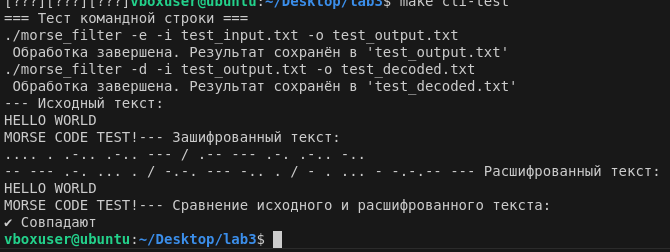


Рисунок 3.3 – Вывод предупреждений об ошибочных комбинациях

В первой команде параметр -e говорит программе, что необходимо выполнить шифрование текста, содержащегося в файле testinput.txt, и сохранить результат в файл testoutput.txt. Весь процесс проходит без дополнительных вопросов пользователю: программа открывает входной файл, читает данные, преобразует их в азбуку Морзе, выводит предупреждения о непреобразуемых символах, если они есть, и сохраняет результат.

Реализация этой функциональности требует использования массива argv и подсчёта количества аргументов через argc. Основные проверки построены с использованием стандартной функции strcmp, которая сравнивает переданные параметры с ожидаемыми значениями, такими как -e, -d, -i, -o. При корректном указании параметров открываются соответствующие файлы, считываются или записываются данные, и вызываются функции шифрования или дешифровки, определённые в модуле morse.c.

Такой подход обеспечивает соответствие требованиям методических указаний, делает программу гибкой, пригодной для повторного использования в различных сценариях, и позволяет без труда подключать её к внешним системам автоматизации. Кроме того, наличие как командного, так и интерактивного режима делает её одинаково удобной как для ручного тестирования, так и для массовой обработки данных.

Таким образом, программа охватывает оба режима работы, включает подробную диагностику и выполняет шифровку и расшифровку текста с соблюдением структуры строк, обработкой ошибок и возможностью проверки корректности восстановления. Это делает её функциональным инструментом для изучения преобразования текстовых данных и принципов построения консольных программ на языке C в Unix-среде.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы удалось не только реализовать функциональную консольную программу-фильтр на языке C, но и глубоко ознакомиться с важнейшими аспектами разработки в среде операционных систем Unix. Работа охватывала как проектирование алгоритмов преобразования текста в код Морзе и обратно, так и применение инструментов низкоуровневого программирования, включая файловый ввод-вывод, работу со строками, системные вызовы и управление памятью. Особое внимание было уделено модульной структуре программы, что позволило четко разделить логику взаимодействия с пользователем, обработки текста и преобразования данных.

Реализация как прямого, так и обратного фильтра с сохранением логики обработки ошибок и предупреждений в терминале стала важным шагом к формированию устойчивых навыков написания надёжных программ. В процессе выполнения задания были решены задачи организации пользовательского интерфейса через терминал, обеспечения удобной навигации между режимами работы и реализации как интерактивного, так и автоматического управления через параметры командной строки. Поддержка опций для указания входного и выходного файла напрямую через аргументы при запуске делает программу пригодной для использования в различных сценариях, включая пакетную обработку текстовых данных.

Дополнительно, благодаря использованию системы сборки make и созданию Makefile, удалось освоить механизмы автоматизации компиляции, очистки проекта и тестирования. Это обеспечило не только удобство при разработке, но и повысило повторяемость и надёжность всех этапов работы. Возможность легко переключаться между ручным тестированием и тестами на основе заранее подготовленных файловых сценариев позволила убедиться в корректности и устойчивости алгоритма.

Лабораторная работа в целом способствовала формированию комплексного понимания разработки консольных утилит на языке C в среде Linux. Она затронула не только техническую сторону программирования, но и принципы проектирования пользовательского взаимодействия, контроля ошибок и организации кода. Полученные знания и навыки будут полезны как при дальнейшем изучении системного программирования, так и при разработке утилит и сервисов, ориентированных на работу с текстовыми данными и консольным интерфейсом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Linux и UNIX: программирование в shell. Руководство разработчика. Дэвид Тейнсли – [Электронный ресурс]. – Режим доступа: https://coollib. com/b/275969/read?ysclid=m6ujdy2lag999562018.

[2] [Системное программирование на C++ для Unix](https://djvu.online/file/yXjAm5jtD3YwQ?ysclid=mdha3a59dl812349636). С. Тимачева – [Электронный ресурс]. – Режим доступа: https://djvu.online/file/yXjAm5jtD3Y wQ?ysclid=mdha3a59dl812349636.

[3] Система построения проектов СMake. Д.В. Дубров – [Электронный ресурс]. – Режим доступа: https://ftp.zhirov.kz/books/IT/Other/Система%20пост роения%20проектов%20CMake%20%28Д.В.%20Дубров%29.pdf.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "morse.h"

void print\_usage(const char\* progname) {

printf("Использование:\n");

printf(" %s -e|-d -i <входной\_файл> -o <выходной\_файл>\n", progname);

printf("Опции:\n");

printf(" -e шифровать (текст -> Морзе)\n");

printf(" -d расшифровать (Морзе -> текст)\n");

printf(" -i FILE указать входной файл\n");

printf(" -o FILE указать выходной файл\n");

printf("Если опции не заданы — запускается интерактивный режим.\n");

}

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

if (argc == 6 && (strcmp(argv[1], "-e") == 0 strcmp(argv[1], "-d") == 0)) {

const char\* infile = NULL;

const char\* outfile = NULL;

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

if (strcmp(argv[i], "-i") == 0 && i + 1 < argc)

infile = argv[++i];

else if (strcmp(argv[i], "-o") == 0 && i + 1 < argc)

outfile = argv[++i];

else {

print\_usage(argv[0]);

return 1;

}

}

if (!infile !outfile) {

fprintf(stderr, "Ошибка: не указаны входной или выходной файл.\n");

print\_usage(argv[0]);

return 1;

}

FILE\* in = fopen(infile, "r");

if (!in) {

perror("Ошибка открытия входного файла");

return 1;

}

FILE\* out = fopen(outfile, "w");

if (!out) {

perror("Ошибка открытия выходного файла");

fclose(in);

return 1;

}

char buffer[4096] = {0};

fread(buffer, 1, sizeof(buffer) - 1, in);

fclose(in);

if (strcmp(argv[1], "-e") == 0) {

encrypt\_text(buffer, out);

} else {

decrypt\_text(buffer, out);

}

fclose(out);

printf(" Обработка завершена. Результат сохранён в '%s'\n", outfile);

return 0;

}

// === ИНТЕРАКТИВНЫЙ РЕЖИМ ===

// (Остаётся прежний интерфейс, как описано выше)

int mode = 0;

int source = 0;

printf("Выберите режим:\n1 - Зашифровать (в Морзе)\n2 - Расшифровать (из Морзе)\nВаш выбор: ");

scanf("%d", &mode);

getchar(); // поглощение \n

printf("Откуда взять текст?\n1 - Ввести вручную\n2 - Прочитать из файла (test\_input.txt)\nВаш выбор: ");

scanf("%d", &source);

getchar(); // поглощение \n

char input\_text[4096] = {0};

if (source == 1) {

printf("Введите текст:\n> ");

fgets(input\_text, sizeof(input\_text), stdin);

input\_text[strcspn(input\_text, "\n")] = 0;

} else {

FILE\* in = fopen("test\_input.txt", "r");

if (!in) {

perror("Ошибка открытия файла test\_input.txt");

return 1;

}

fread(input\_text, 1, sizeof(input\_text) - 1, in);

fclose(in);

}

FILE\* out = fopen("test\_output.txt", "w");

if (!out) {

perror("Ошибка записи в файл");

return 1;

}

if (mode == 1)

encrypt\_text(input\_text, out);

else

decrypt\_text(input\_text, out);

fclose(out);

printf("\n--- Результат сохранён в test\_output.txt ---\n");

FILE\* show = fopen("test\_output.txt", "r");

if (show) {

char buffer[4096] = {0};

fread(buffer, 1, sizeof(buffer) - 1, show);

fclose(show);

printf("Содержимое результата:\n%s\n", buffer);

printf("Хотите вернуть исходный текст?\n1 - Да\n2 - Нет\nВаш выбор: ");

int rev = 0;

scanf("%d", &rev);

getchar();

if (rev == 1) {

printf("\n--- Восстановленный текст ---\n");

if (mode == 1)

decrypt\_text(buffer, stdout);

else

encrypt\_text(buffer, stdout);

}

}

return 0;

}