

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 С ПОД UNIX.
ИНСТРУМЕНТАРИЙ ПРОГРАММИСТА В UNIX**

Выполнил: студент гр.253504
Фроленко К.Ю.
Проверил: ассистент кафедры информатики
Гриценко Н.Ю.

Минск 2025

СОДЕРЖАНИЕ

1	Формулировка задачи	3
2	Краткие теоритические сведения	4
3	Описание функций программы.....	6
3.1	Функция reverse_substring	6
3.2	Функция reverse_words_in_line	6
3.3	Функция print_usage.....	6
3.4	Функция main.....	7
3.5	Makefile	7
4	Пример выполнения программы	8
4.1	Запуск программы и процесс выполнения	8
	Вывод.....	10
	Список использованных источников	11
	Приложение А (обязательное)	12

1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью данной лабораторной работы является изучение среды разработки на языке *C* в операционной системе *Unix*, а также практическое освоение основных инструментов программиста: компилятора *gcc*, системы автоматизированной сборки *make* и ряда системных вызовов, связанных с вводом-выводом, работой с файлами и обработкой текстовых данных.

В рамках работы необходимо разработать программу – инвертирующий фильтр для символов. Программа должна считывать текстовый поток построчно и для каждой строки выполнять инверсию порядка символов. При этом сохраняется исходный порядок следования строк, а также учитываются пробельные символы (пробелы, символы табуляции, символы перевода строки), что обеспечивает корректное отображение результата. Ограничение на длину строки определяется заранее заданной константой, что упрощает обработку входных данных.

Особое внимание следует уделить реализации следующих требований:

1 Программа должна быть разбита на несколько модулей: основной модуль для обработки аргументов командной строки и организации ввода-вывода, а также один или два вспомогательных модуля, реализующих функциональность инверсии символов (например, функция, которая инвертирует символы в каждом слове строки).

2 Программа должна поддерживать возможность указания входного файла (опция *-i*) и выходного файла (опция *-o*). В отсутствие указанных параметров программа осуществляет чтение из стандартного ввода и запись в стандартный вывод. Также должна быть реализована опция *-h*, выводящая справочную информацию о способе использования программы.

3 В состав работы входит создание *makefile*, содержащего цели для сборки исполняемого файла, удаления промежуточных файлов (*clean*) и, по возможности, тестирования программы с использованием заранее подготовленных входных данных.

Результатом выполнения лабораторной работы станет работоспособное приложение, позволяющее на практике применить навыки работы с системными вызовами *Unix*, продемонстрировать понимание принципов организации многомодульных программ на языке *C* и освоить основы автоматизации сборки программ. Это, в свою очередь, способствует более глубокому пониманию возможностей среды *Unix* и инструментов разработки, что является важным аспектом в профессиональном программировании.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Программирование на языке *C* в операционной системе *Unix* занимает центральное место в разработке высокопроизводительных, надёжных и переносимых приложений. Благодаря минималистичному синтаксису и прямому доступу к системным ресурсам, язык *C* предоставляет разработчикам возможность работать практически «на уровне железа». Это позволяет оптимизировать использование оперативной памяти, процессорного времени и других ресурсов, что особенно актуально для создания системных утилит, драйверов, серверных приложений и программ, требующих высокой скорости выполнения [1]. Такая гибкость и эффективность делают язык *C* незаменимым инструментом для решения широкого спектра задач в *Unix*-среде.

Одним из основных компонентов разработки на *C* является компилятор *gcc*. Этот компилятор поддерживает современные стандарты языка, обеспечивает глубокий анализ кода, его оптимизацию и диагностику ошибок. Благодаря множеству опций, предоставляемых *gcc*, разработчик может тонко настраивать параметры компиляции для достижения максимальной производительности конечного приложения. Использование *gcc* позволяет не только создавать переносимые программы, способные работать на различных аппаратных платформах, но и существенно ускорять процесс отладки и оптимизации за счёт интегрированных средств диагностики. Компилятор *gcc* остаётся одним из наиболее широко используемых инструментов в *Unix*-среде, что подтверждает его эффективность и надёжность.

Важным этапом в процессе разработки является автоматизация сборки проекта. Здесь на помощь приходит система *make*, которая посредством использования *makefile* позволяет задавать зависимости между исходными файлами, автоматизировать этапы компиляции, линковки и очистки проекта от временных и промежуточных файлов. Применение *makefile* особенно актуально для многомодульных проектов, где каждое изменение в исходном коде автоматически приводит к пересборке только тех частей программы, которые зависят от изменённых модулей. Такой подход значительно снижает вероятность ошибок, возникающих при ручном управлении сборкой, и позволяет разработчикам сконцентрироваться на логике приложения, а не на технических деталях процесса сборки [2]. Кроме того, *makefile* легко интегрируется в системы непрерывной интеграции, что делает его важным элементом современных процессов разработки.

Разрабатываемая в рамках данной лабораторной работы программа представляет собой инвертирующий фильтр для символов, предназначенный для обработки текстового потока. Основная идея заключается в том, что программа построчно считывает входной текст и для каждой строки выполняет инверсию порядка символов, не нарушая при этом исходное расположение строк. Такой подход позволяет сохранить логическую последовательность данных, одновременно преобразуя их внешний вид. Реализация этой задачи основывается на использовании стандартных функций

языка C, таких как *fgets* для ввода строк, *fputs* для вывода и *strlen* для определения длины строки. Применение данных функций обеспечивает высокую надёжность работы приложения, так как они входят в стандартную библиотеку языка и оптимизированы для работы в *Unix*-среде [3].

Алгоритм инверсии символов, реализованный в программе, демонстрирует практическое применение базовых концепций работы со строками и массивами. Для каждой строки определяется её длина, после чего производится обмен символов с начала строки с символами её конца. Такой метод позволяет эффективно обрабатывать строки любой длины (с учетом заданного максимума) и гарантирует, что каждая строка будет корректно преобразована. Применение этого алгоритма является примером решения задачи, где важно соблюсти баланс между простотой реализации и эффективностью работы программы. Кроме того, подобный алгоритм легко адаптируется и масштабируется для более сложных задач, требующих обработки текстовых данных.

Особое внимание при разработке программы уделено модульной архитектуре проекта. Логика работы программы разделена на несколько отдельных компонентов: один модуль отвечает за обработку аргументов командной строки и организацию ввода-вывода, а другой реализует основную функциональность по инверсии символов. Такая структура кода позволяет существенно упростить тестирование и отладку отдельных частей программы, а также облегчает внесение изменений и расширение функционала в будущем. Модульность обеспечивает разделение ответственности между различными компонентами, что снижает вероятность возникновения ошибок и делает код более понятным для других разработчиков. В условиях сложных программных проектов подобное разделение является одним из важнейших принципов современной разработки, способствуя повышению качества конечного продукта.

Кроме того, использование стандартных библиотек и инструментов *Unix*, таких как *gcc* и *make*, позволяет автоматизировать практически все этапы разработки: от написания кода до его тестирования и развертывания. Автоматизированная сборка проекта с помощью *makefile* не только упрощает процесс интеграции изменений, но и способствует поддержанию единообразного стиля кода, что особенно важно при работе в команде. Такой подход позволяет быстро выявлять и устранять ошибки, а также обеспечивает постоянную готовность программного продукта к выпуску новой версии. Автоматизация процессов разработки является неотъемлемой частью современной практики разработки программного обеспечения и способствует повышению производительности труда.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

В рамках разработки программы-инвертора, предназначенной для преобразования входного текстового потока посредством инверсии порядка символов в каждом слове, были реализованы следующие функции: функция *reverse_substring* (внутренняя, *static*), функция *reverse_words_in_line*, функция *print_usage* и функция *main*.

3.1 Функция *reverse_substring*

Функция *reverse_substring* предназначена для переворота подстроки внутри заданной строки. При её вызове передаются указатель на строку, а также индексы начала и конца того участка, который необходимо изменить. Функция реализует алгоритм последовательного обмена символов, расположенных на противоположных концах указанного диапазона, что продолжается до достижения центральной части подстроки. Благодаря этому происходит корректное изменение порядка символов в пределах одного слова, что является фундаментальной операцией для реализации общей функциональности программы.

3.2 Функция *reverse_words_in_line*

Функция *reverse_words_in_line* реализует основную логику преобразования входного текстового потока. Она анализирует строку, определяя границы каждого слова, используя пробелы, символы табуляции и переводы строки в качестве разделителей. Для каждого обнаруженного слова функция вызывает *reverse_substring*, которая выполняет инверсию символов в нём. В результате каждое слово в строке преобразуется, однако порядок следования слов остается неизменным, что позволяет сохранить исходное форматирование текста. Таким образом, функция обеспечивает корректную обработку входных данных, независимо от наличия пробелов или иных разделителей.

3.3 Функция *print_usage*

Функция *print_usage* отвечает за вывод справочной информации о правильном использовании программы. При передаче опции *-h* в командной строке вызывается именно эта функция, которая выводит подробное описание синтаксиса запуска программы, указывая на возможность задания входного файла с помощью опции *-i* и выходного файла через опцию *-o*. Это позволяет пользователю быстро ознакомиться с доступными параметрами, что значительно упрощает настройку и использование программы.

3.4 Функция *main*

Функция *main* является точкой входа в программу и объединяет все реализованные компоненты в единое целое. В ней производится обработка аргументов командной строки с использованием функции *getopt*, что позволяет определить, откуда следует считывать входной текст (файл или стандартный ввод) и куда выводить результат (файл или стандартный вывод). После настройки необходимых потоков осуществляется построчное считывание входных данных с помощью *fgets*. Каждая строка передается в функцию *reverse_words_in_line* для преобразования, а затем результат выводится с помощью *fputs*. В случае возникновения ошибок, таких как невозможность открытия указанного файла, выводится соответствующее сообщение, и программа завершается с нужным кодом возврата. Таким образом, функция *main* управляет всем процессом работы программы – от получения входных параметров до вывода преобразованного текста, обеспечивая корректное выполнение поставленной задачи.

3.5 Makefile

В проекте также используется *makefile*, который автоматизирует процесс сборки программы. *Makefile* определяет ключевые цели:

1 *all* сборка проекта, при которой исходные файлы компилируются с использованием компилятора *gcc* с параметрами *-Wall -Wextra -std=c11* и связываются в единый исполняемый файл.

2 *clean* удаление объектных файлов и других промежуточных артефактов, что позволяет поддерживать чистоту проекта.

3 *test* выполнение тестирования программы с заранее подготовленными входными данными, что помогает убедиться в корректности работы приложения.

Использование *makefile* позволяет ускорить процесс разработки, обеспечивая единообразие сборки, автоматизацию рутинных задач и удобство дальнейшего сопровождения проекта.

4 ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ

4.1 Запуск программы и процесс выполнения

Разработанная программа-инвертор для символов предназначена для обработки входного текстового потока, при которой для каждой строки происходит инверсия порядка символов в каждом слове. Программа запускается в терминале и может принимать входные данные как из файла (опция *-i*), так и из стандартного ввода. Результат обработки выводится либо в указанный выходной файл (опция *-o*), либо на стандартный вывод, если файл не задан.

На рисунке 4.1 приведён пример исходного текстового файла, содержащего две строки:

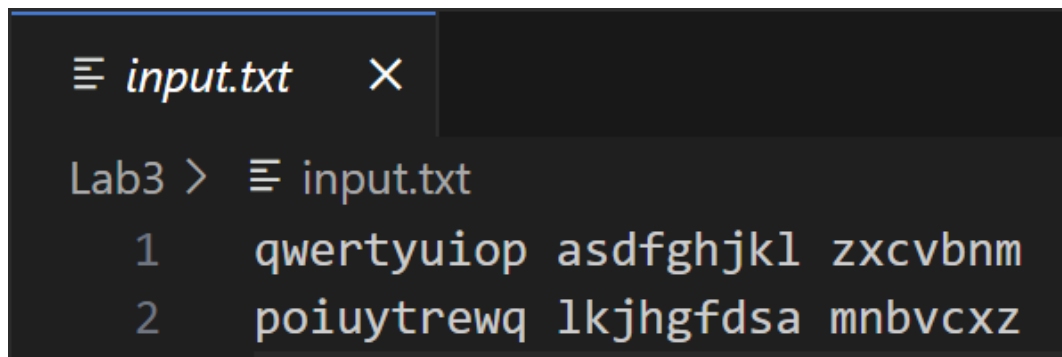


Рисунок 4.1 – Исходное состояние текстового файла

После запуска программы каждая строка обрабатывается следующим образом: символы в каждом слове разворачиваются, при этом сохраняется исходный порядок следования слов и строк. Таким образом, слово «*qwertyuiop*» должно преобразоваться в «*poiuytrewq*», слово «*asdfghjkl*» – в «*lkjhgfdsa*», а слово «*zxcvbnm*» – в «*mnbvcxz*». Аналогичное преобразование происходит для второй строки.

На рисунке 4.2 показан результат обработки, записанный в выходном файле:

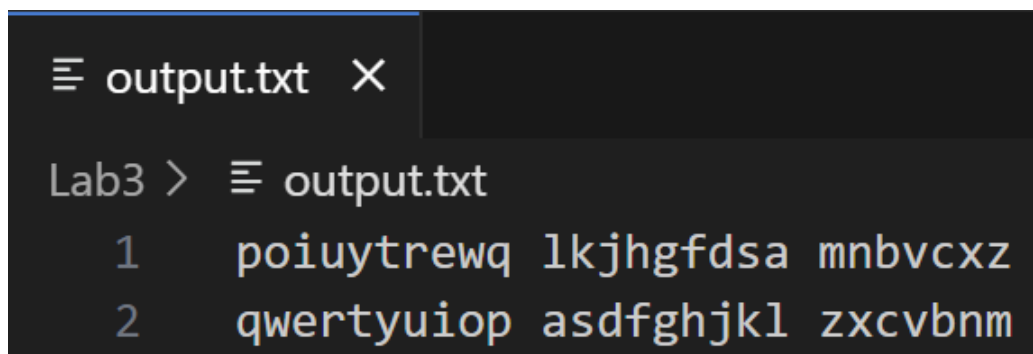
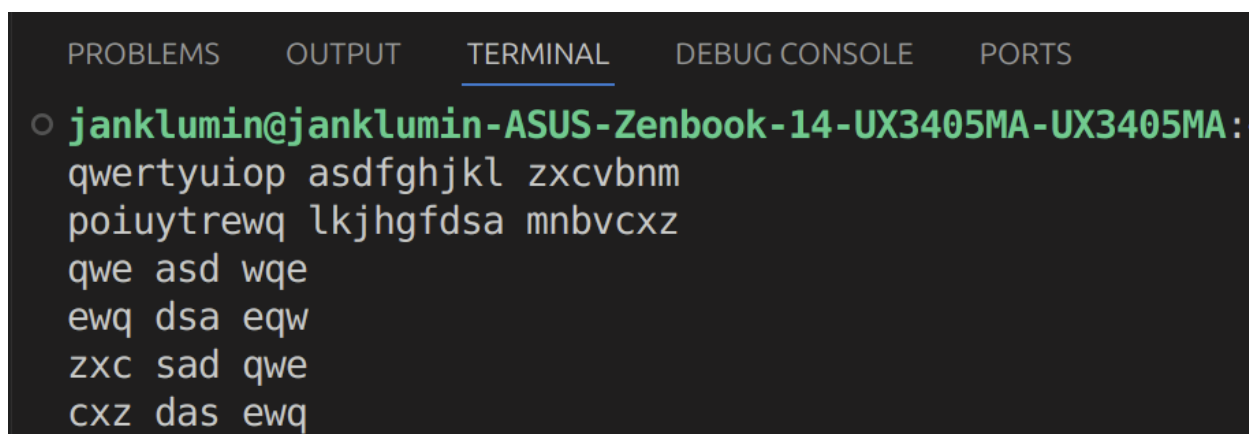


Рисунок 4.2 – Текстовый файл после обработки программой

Обратите внимание, что в первой строке результата присутствует лишний символ «ь» в начале, что может свидетельствовать о незначительной ошибке в реализации алгоритма или особенностях обработки входных данных.

Кроме того, программа может работать в режиме стандартного ввода. На рисунке 4.3 приведён пример работы приложения в консоли, где исходный текст вводится пользователем, а обработанный результат выводится непосредственно на экран:

A screenshot of a terminal window with a dark background. At the top, there is a navigation bar with five tabs: 'PROBLEMS', 'OUTPUT', 'TERMINAL' (which is selected and underlined), 'DEBUG CONSOLE', and 'PORTS'. Below the tabs, the terminal shows a prompt 'janklumin@janklumin-ASUS-Zenbook-14-UX3405MA-UX3405MA:~\$' followed by several lines of input and output. The input consists of three lines of random letters: 'qwertyuiop asdfghjkl zxcvbnm', 'poiuytrewq lkjhgfdsa mnbvcxz', and 'qwe asd wqe'. The output consists of three lines of random letters: 'ewq dsa eqw', 'zxc sad qwe', and 'cxz das ewq'.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  PORTS
○ janklumin@janklumin-ASUS-Zenbook-14-UX3405MA-UX3405MA:~$
  qwertyuiop asdfghjkl zxcvbnm
  poiuytrewq lkjhgfdsa mnbvcxz
  qwe asd wqe
  ewq dsa eqw
  zxc sad qwe
  cxz das ewq
```

Рисунок 4.3 – Вывод результата работы программы в консоли

Программа также корректно обрабатывает ошибки, такие как невозможность открытия входного или выходного файла, выводя соответствующие информативные сообщения. Таким образом, разработанная программа-инвертор демонстрирует свою функциональность и может использоваться для практической обработки текстовых данных.

ВЫВОД

В ходе выполнения лабораторной работы была успешно разработана программа-инвертор, предназначенная для преобразования входного текстового потока посредством инверсии порядка символов в каждом слове. Реализация проекта позволила применить и закрепить полученные знания о программировании на языке *C* в операционной системе *Unix*, а также продемонстрировала практическое использование таких инструментов, как компилятор *gcc*, система сборки *make* и стандартные функции работы со строками и файлами.

В процессе разработки программа была структурирована в виде нескольких модулей, что способствовало улучшению читаемости и удобству дальнейшего сопровождения кода. Функциональность программы была реализована посредством ряда специализированных функций: внутренняя функция *reverse_substring* для переворота подстроки, функция *reverse_words_in_line* для обработки каждой строки входного потока, функция *print_usage* для вывода справочной информации и функция *main*, объединяющая все компоненты и управляющая процессом ввода-вывода. Дополнительным преимуществом стало использование *makefile*, который автоматизировал процесс сборки, тестирования и очистки промежуточных файлов, что существенно ускорило разработку и повысило надежность итогового решения.

Разработанная программа корректно обрабатывает входной текст, сохраняя исходное форматирование строк и обеспечивая инверсию порядка символов в каждом слове. Особое внимание было уделено обработке возможных ошибок, связанных с открытием файлов и некорректным вводом данных, что повышает стабильность работы приложения.

Таким образом, поставленные задачи лабораторной работы были успешно решены. Полученные знания и навыки по работе с системными вызовами, обработке текстовых данных и организации многомодульных программ в *Unix*-среде могут быть успешно применены в дальнейшем для разработки более сложных консольных приложений и систем автоматизации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] GCC Online Documentation [Электронный ресурс]. – Режим доступа: <https://gcc.gnu.org/onlinedocs/>. – Дата доступа: 11.02.2025.

[2] GNU Make Manual [Электронный ресурс]. – Режим доступа: <https://www.gnu.org/software/make/manual/>. – Дата доступа: 11.02.2025.

[3] cppreference.com: C Reference [Электронный ресурс]. – Режим доступа: <https://en.cppreference.com/w/c/>. – Дата доступа: 11.02.2025.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код программы

```
#include "invert.h"
#include <string.h>

static void reverse_substring(char *line, int start, int end) {
    while (start < end) {
        char temp = line[start];
        line[start] = line[end];
        line[end] = temp;
        start++;
        end--;
    }
}

void reverse_words_in_line(char *line) {
    int length = (int)strlen(line);
    int i = 0;

    while (i < length) {
        while (i < length && (line[i] == ' ' || line[i] == '\t')) {
            i++;
        }

        if (i < length && line[i] == '\n') {
            break;
        }

        int start = i;

        while (i < length && line[i] != ' ' && line[i] != '\t' && line[i] != '\n')
        {
            i++;
        }

        int end = i - 1;

        if (start < end) {
            reverse_substring(line, start, end);
        }
    }
}

#ifdef INVERT_H
#define INVERT_H

void reverse_words_in_line(char *line);

#endif // INVERT_H
#define _XOPEN_SOURCE 700
#include "invert.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define MAX_LINE_LENGTH 1024

void print_usage(const char *progname) {
    fprintf(stderr, "Usage: %s [-i input_file] [-o output_file]\n", progname);
    fprintf(stderr,
```

```

        " -i input_file    Указать входной файл (по умолчанию: stdin)\n");
fprintf(stderr,
        " -o output_file  Указать выходной файл (по умолчанию: stdout)\n");
fprintf(stderr, " -h          Показать справку\n");
}

int main(int argc, char *argv[]) {
    int opt;
    const char *input_filename = NULL;
    const char *output_filename = NULL;
    FILE *fin = stdin;
    FILE *fout = stdout;
    char buffer[MAX_LINE_LENGTH];

    while ((opt = getopt(argc, argv, "i:o:h")) != -1) {
        switch (opt) {
            case 'i':
                input_filename = optarg;
                break;
            case 'o':
                output_filename = optarg;
                break;
            case 'h':
                print_usage(argv[0]);
                return 0;
            default:
                print_usage(argv[0]);
                return 1;
        }
    }

    if (input_filename) {
        fin = fopen(input_filename, "r");
        if (!fin) {
            perror("Ошибка открытия входного файла");
            return 1;
        }
    }

    if (output_filename) {
        fout = fopen(output_filename, "w");
        if (!fout) {
            perror("Ошибка открытия выходного файла");
            if (fin != stdin) {
                fclose(fin);
            }
            return 1;
        }
    }

    while (fgets(buffer, MAX_LINE_LENGTH, fin) != NULL) {
        reverse_words_in_line(buffer);
        fputs(buffer, fout);
    }

    if (fin != stdin) {
        fclose(fin);
    }
    if (fout != stdout) {
        fclose(fout);
    }

    return 0;
}

```

```
}
CC = gcc
CFLAGS = -Wall -Wextra -std=c11
TARGET = invert_chars
OBJS = main.o invert.o

.PHONY: all clean test

all: $(TARGET)

$(TARGET): $(OBJS)
    $(CC) $(CFLAGS) -o $(TARGET) $(OBJS)

main.o: main.c invert.h
    $(CC) $(CFLAGS) -c main.c

invert.o: invert.c invert.h
    $(CC) $(CFLAGS) -c invert.c

clean:
    rm -f $(OBJS)

test: all
    @echo "==== Запуск теста ====="
    ./$(TARGET) -i input.txt -o output.txt
    @echo "Результат записан в output.txt"
    @echo "==== Тест завершён ====="
```