

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

ОТЧЕТ
к лабораторной работе №3
на тему

**ОСНОВЫ ПРОГРАММИРОВАНИЯ НА С ПОД UNIX.
ИНСТРУМЕНТАРИЙ ПРОГРАММИСТА В UNIX**

Студент
Преподаватель

О. Л. Дайнович
Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

| | |
|--|---|
| 1 Цель работы | 3 |
| 2 Теоретические сведения | 4 |
| 3 Полученные результаты | 5 |
| Заключение | 7 |
| Список использованных источников | 8 |
| Приложение А (обязательное) Листинг кода | 9 |

1 ЦЕЛЬ РАБОТЫ

Изучение среды программирования и основных инструментов: компилятор/сборщик («коллекция компиляторов») gcc, управление обработкой проекта make (и язык makefile), библиотеки и т.д. Практическое использование основных библиотек и системных вызовов: ввод-вывод и работа с файлами, обработка текста, распределение памяти, управление выполнением и т.п.

Написать программу в соответствии с вариантом задания, создать makefile для управления обработкой проекта и проверить выполнение описанных в нем целей, собрать и протестировать исполняемый файл.

Проект желательно строить многомодульным (например, головной модуль и 1-2 подключаемых к нему модулей с «рабочими» функциями).

Для программ-фильтров надо реализовать возможность явного указания выходного файла в командной строке при вызове, а также опций (если они предусмотрены).

Среди целей makefile должны быть сборка и «очистка» (удаление промежуточных файлов) проекта, а также по возможности тестирование исполняемого файла с заранее заготовленными входными данными.

Условие задания: инверсия порядка символов в каждой строке потока, порядок самих строк не изменяется. Длину строк можно считать ограниченной некоторой достаточно большой константой.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Gcc — это компилятор языка программирования C и C++, разработанный Free Software Foundation. Он один из самых популярных и широко используется в индустрии разработки программного обеспечения. Компилятор Gcc позволяет преобразовать исходный код программы, написанной на C или C++, в машинный код, который может быть исполнен на целевой системе.

Одной из главных особенностей Gcc является его переносимость. Он доступен для большинства современных операционных систем, таких как Linux, macOS и Windows. Кроме того, Gcc поддерживает большое количество архитектур процессоров, таких как x86, ARM, PowerPC и других.

Gcc позволяет разработчикам писать эффективные и портативные программы. Он обладает множеством опций и флагов, которые позволяют оптимизировать код, управлять генерацией отладочной информации, контролировать процесс компиляции и многое другое. Компилятор Gcc также поддерживает стандарты языка C и C++, что позволяет использовать современные возможности этих языков программирования.

Компилятор превращает набор кода в объектные файлы, или модули. С помощью линковщика они соединяет все воедино с учетом зависимостей и связей между исходниками. Результат — готовый исполняемый файл.

Утилита make автоматизирует процесс — разработчику достаточно набрать одну команду в консоли. Она преобразует исходный код в модули, а их — в исполняемые файлы, которые можно запустить. Так создаются библиотеки и разнообразные программные продукты. Makefile показывает программе правила, по которым нужно выполнять преобразования.

Make работает на Linux и системах на базе Unix — для этих ОС утилита считается основным средством сборки программ. В Windows тоже есть концепция Makefile, но управляет этими файлами утилита nmake. У make две версии: для платформы BSD и для GNU. Первая используется в операционных системах FreeBSD, OpenBSD и NetBSD, вторая — в MacOS и Linux. Утилита обычно есть в ОС по умолчанию.

3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ

В результате лабораторной работы был написан скрипт, реализующий инверсию порядка символов в каждой строке потока.

Основной код написан на языке C. Bash скрипт собирает программу, состоящую из файлов .cpp и .h, с помощью файла Makefile и выполняет код.

Программа считывает исходный текст из файла «test» расширения .txt и редактирует его, инвертируя порядок символов в строке (рисунок 1).



Рисунок 1 – Исходный текст

После инвертирования, программа выводит в консоль отредактированный текст, а также информацию о сборке программы (рисунок 2).

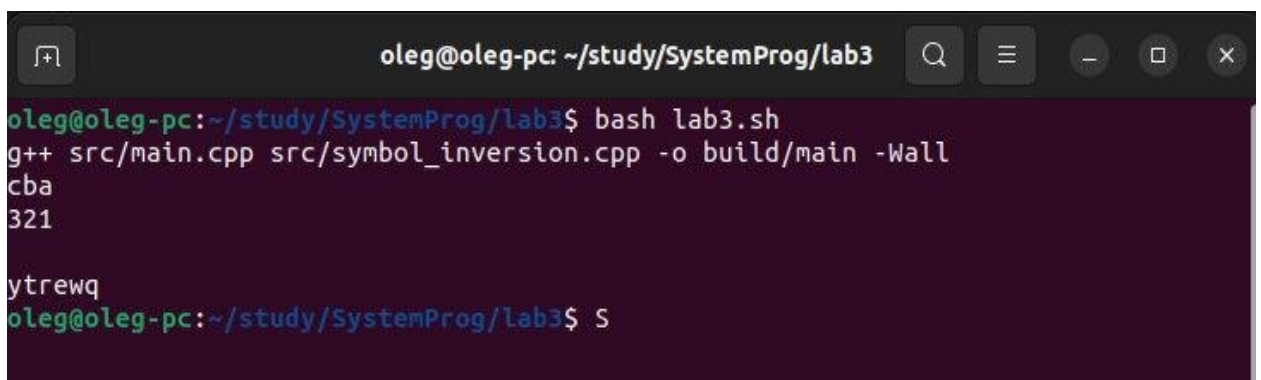
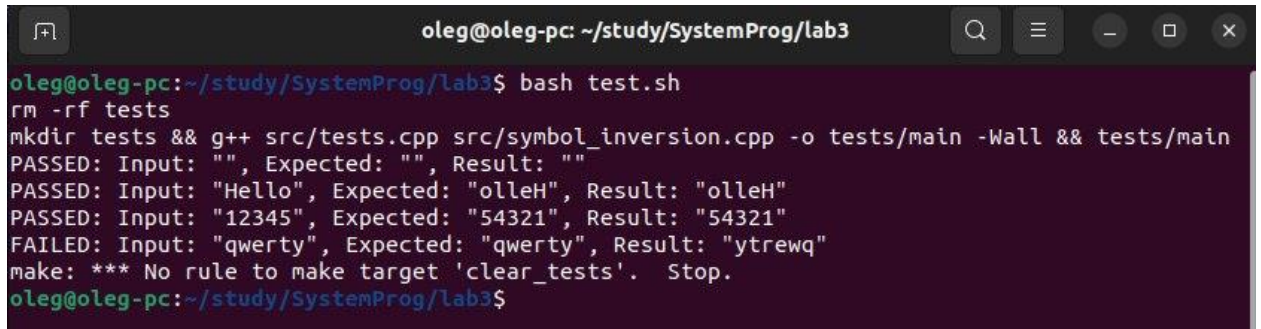


Рисунок 3 – Вывод итога программы в консоль

Помимо выполнения основного алгоритма, также реализовано тестирование исполняемого файла с заранее заготовленными входными данными. Результаты всех тестов выводятся в консоль при запуске файла «test» расширения .sh. В выведенных данных показаны результаты каждого отдельного теста: введенные данные, ожидаемый результат и итоговый результат теста (рисунок 3).

A screenshot of a terminal window with a dark background. The window title is 'oleg@oleg-pc: ~/study/SystemProg/lab3'. The terminal shows the execution of a script 'test.sh'. The script first removes the 'tests' directory and then creates it. It then compiles two C++ files, 'tests.cpp' and 'symbol_inversion.cpp', into an executable 'main' in the 'tests' directory. The script runs 'main' with several test cases. The output shows 'PASSED' for empty input, 'Hello' (resulting in 'olleH'), and '12345' (resulting in '54321'). It shows 'FAILED' for 'qwerty' (resulting in 'ytrewq'). Finally, it shows a 'make' error: '*** No rule to make target 'clear_tests'. Stop.'.

```
oleg@oleg-pc:~/study/SystemProg/lab3$ bash test.sh
rm -rf tests
mkdir tests && g++ src/tests.cpp src/symbol_inversion.cpp -o tests/main -Wall && tests/main
PASSED: Input: "", Expected: "", Result: ""
PASSED: Input: "Hello", Expected: "olleH", Result: "olleH"
PASSED: Input: "12345", Expected: "54321", Result: "54321"
FAILED: Input: "qwerty", Expected: "qwerty", Result: "ytrewq"
make: *** No rule to make target 'clear_tests'. Stop.
oleg@oleg-pc:~/study/SystemProg/lab3$
```

Рисунок 3 – Вывод итогов тестирования в консоль

ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы была изучена среда программирования и основные инструменты: компилятор/сборщик («коллекция компиляторов») gcc, управление обработкой проекта make (и язык makefile), библиотеки и т.д. Практическое использование основных библиотек и системных вызовов: ввод-вывод и работа с файлами, обработка текста, распределение памяти, управление выполнением и т.п.. В ходе работы была написана программа на языке программирования C для обработки входных данных, реализующая инверсию порядка символов в каждой строке потока.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Компилятор GCC [Электронный ресурс]. – Режим доступа: <https://parallel.uran.ru/book/export/html/25> – Дата доступа: 03.03.2024.

[2] Makefile [Электронный ресурс]. – Режим доступа: <https://blog.skillfactory.ru/glossary/makefile/> – Дата доступа: 04.03.2024.

[3] Что такое Makefile и как начать его использовать [Электронный ресурс]. – Режим доступа: <https://guides.hexlet.io/ru/makefile-as-task-runner/> – Дата доступа: 04.03.2024.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг кода

Листинг 1 – Файл lab3.sh

```
#!/usr/bin/bash
make
./build/main
```

Листинг 2 – Файл test.sh

```
#!/usr/bin/bash
make test
make clear_tests
```

Листинг 3 – Файл makefile

```
BUILD_DIR=build
TESTS_DIR=tests
SRC_DIR=src

all: build
    g++ ${SRC_DIR}/main.cpp ${SRC_DIR}/symbol_inversion.cpp -o
    ${BUILD_DIR}/main -Wall

test: clean_tests
    mkdir ${TESTS_DIR} && g++ ${SRC_DIR}/tests.cpp
    ${SRC_DIR}/symbol_inversion.cpp -o ${TESTS_DIR}/main -Wall &&
    ${TESTS_DIR}/main

clean_tests:
    rm -rf ${TESTS_DIR}

build:
    mkdir build

clean: clean_tests
    rm -rf build
```

Листинг 4 – Файл main.cpp

```
#include <stdio.h>
#include <string.h>
#include "symbol_inversion.h"

#define MAX_LENGTH 1000
```

```

int main() {
    char line[MAX_LENGTH];
    FILE* file = fopen("test.txt", "r");

    if (file == NULL) {
        printf("Error. Failed to open file.\n");
        return 1;
    }

    while (fgets(line, sizeof(line), file)) {
        if (line[strlen(line) - 1] == '\n') {
            line[strlen(line) - 1] = '\0';

            symbolInversion(line);
            printf("%s\n", line);
        }

        fclose(file);
        return 0;
    }
}

```

Листинг 5 – Файл symbol_inversion.cpp

```

#include "symbol_inversion.h"
#include <string.h>

void symbolInversion(char* str) {
    int length = strlen(str);
    int i = 0;
    int j = length - 1;

    while (i < j) {
        char temp = str[i];
        str[i] = str[j];
        str[j] = temp;
        i++;
        j--;
    }
}

```

Листинг 6 – Файл symbol_inversion.h

```

#ifndef SYMBOL_INVERSION_H
#define SYMBOL_INVERSION_H

void symbolInversion(char* str);

#endif

```