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 C ПОД UNIX.  
ИНСТРУМЕНТАРИЙ ПРОГРАММИСТА В UNIX**

Студент Д. С. Кончик

Преподаватель Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 5](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

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

Написать многомодульную программу (например, головной модуль и подключаемые к нему модули с «рабочими» функциями), преобразующую символы потока в комбинации азбуки Морзе, создать *makefile* для управления обработкой проекта и проверить выполнение описанных в нем целей, собрать и протестировать исполняемый файл.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

*GСС* – это свободно доступный оптимизирующий компилятор для языков *C*, *C++*. Собственно, программа *gcc* – это некоторая надстройка над группой компиляторов, которая способна анализировать имена файлов, передаваемые ей в качестве аргументов, и определять, какие действия необходимо выполнить. Файлы с расширением *.cc* или *.c* рассматриваются, как файлы на языке *C++*, файлы с расширением *.c* – как программы на языке *C*, а файлы c расширением *.o* считаются объектными.

Можно использовать компилятор *gcc* для компиляции программ в объектные модули и для компоновки полученных модулей в единую исполняемую программу.

В процессе компоновки очень часто приходится использовать библиотеки. Библиотекой называют набор объектных файлов, сгруппированных в единый файл и проиндексированных. Когда команда компоновки обнаруживает некоторую библиотеку в списке объектных файлов для компоновки, она проверяет, содержат ли уже скомпонованные объектные файлы вызовы для функций, определенных в одном из файлов библиотек. Если такие функции найдены, соответствующие вызовы связываются с кодом объектного файла из библиотеки [1].

*Makefile* – это файл, который хранится вместе с кодом в репозитории. Его обычно помещают в корень проекта. Он выступает и как документация, и как исполняемый код. Мейкфайл скрывает за собой детали реализации и раскладывает "по полочкам" команды, а утилита *make* запускает их из того мейкфайла, который находится в текущей директории.

Изначально *make* предназначалась для автоматизации сборки исполняемых программ и библиотек из исходного кода. Она поставлялась по умолчанию в большинство \*nix дистрибутивов, что и привело к её широкому распространению и повсеместному использованию. Позже оказалось что данный инструмент удобно использовать и при разработке любых других проектов, потому что процесс в большинстве своём сводится к тем же   
задачам – автоматизация и сборка приложений.

Применение мейка в проектах стало стандартом для многих разработчиков, включая крупные проекты. Примеры мейкфайла можно найти у таких проектов, как *Kubernetes*, *Babel* и *Ansible* [2].

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате лабораторной работы была создана программа, преобразующая символы латинского алфавита в комбинации азбуки Морзе. При запуске программы без передачи аргументов появляется информация о возможных параметрах командной строки (рисунок 1).

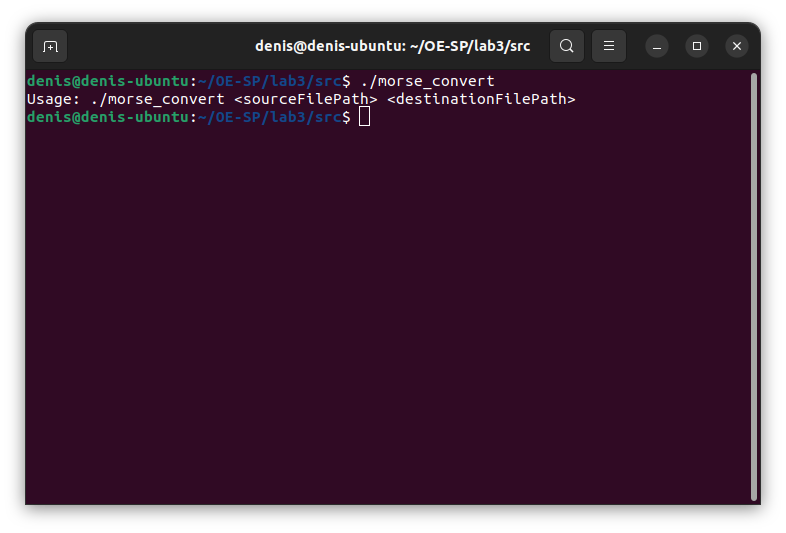


Рисунок 1 – Вывод инструкции

Программа посимвольно считывает исходный файл, конвертирует содержимое в азбуку Морзе и записывает в файл назначения (рисунок 2). Алфавит программы – символы латинского алфавита и цифры, иные символы отбрасываются.

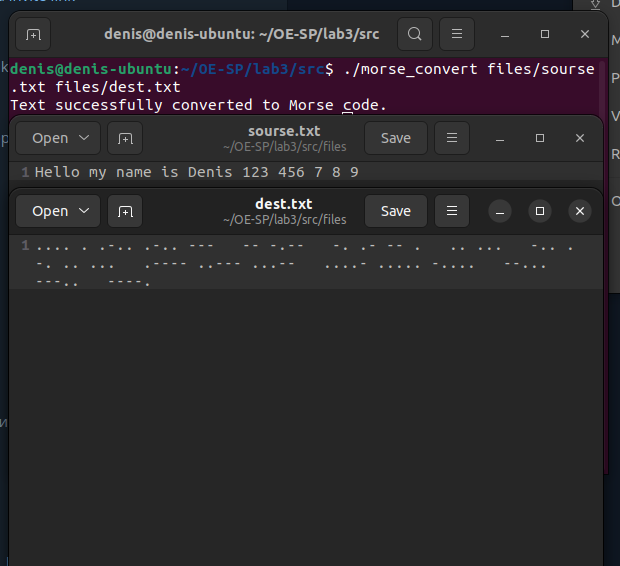


Рисунок 2 – Результат работы программы

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были изучены среда программирования и основные инструменты компиляции, такие как "коллекция компиляторов" *gcc*, а также управление процессом сборки проекта с помощью *make* (и языка *makefile*).

Результатом выполнения лабораторной работы стала созданная многомодульная программа, способная преобразовывать символы латинского алфавита и цифры в соответствующие комбинации азбуки Морзе. При запуске программы без передачи аргументов выводится информация о возможных параметрах командной строки, что улучшает удобство использования программы.

Программа работает посимвольно, считывая содержимое исходного файла, конвертируя его в азбуку Морзе и записывая результат в файл-назначение. Алфавит программы ограничен символами латинского алфавита и цифрами, а все остальные символы отбрасываются в процессе обработки.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Инструментарий программиста в Linux: Компилятор GCC [Электронный ресурс]. – Режим доступа: http://parallel.imm.uran.ru/freesoft/make/instrum.html.
2. Что такое Makefile и как начать его использовать [Электронный ресурс]. – Режим доступа: https://guides.hexlet.io/ru/makefile-as-task-runner/.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл *main.c*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "morse.h"

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

    if (argc != 3) {

        printf("Usage: %s <sourceFilePath> <destinationFilePath>\n", argv[0]);

        return 1;

    }

    char \*sourceFilePath = argv[1];

    char \*destinationFilePath = argv[2];

    FILE \*sourceFile = fopen(sourceFilePath, "r");

    FILE \*destinationFile = fopen(destinationFilePath, "w");

    if (sourceFile == NULL || destinationFile == NULL) {

        printf("Error opening files.\n");

        return 1;

    }

    char c;

    while ((c = fgetc(sourceFile)) != EOF) {

        char\* morse = charToMorse(c);

        fputs(morse, destinationFile);

        fputs(" ", destinationFile);

    }

    printf("Text successfully converted to Morse code.\n");

    fclose(sourceFile);

    fclose(destinationFile);

    return 0;

}

Листинг 2 – Файл *morse\_dictionary.c*

const char \*morseCode[] = {

    ".-",    // A

    "-...",  // B

    "-.-.",  // C

    "-..",   // D

    ".",     // E

    "..-.",  // F

    "--.",   // G

    "....",  // H

    "..",    // I

    ".---",  // J

    "-.-",   // K

    ".-..",  // L

    "--",    // M

    "-.",    // N

    "---",   // O

    ".--.",  // P

    "--.-",  // Q

    ".-.",   // R

    "...",   // S

    "-",     // T

    "..-",   // U

    "...-",  // V

    ".--",   // W

    "-..-",  // X

    "-.--",  // Y

    "--..",  // Z

    "-----", // 0

    ".----", // 1

    "..---", // 2

    "...--", // 3

    "....-", // 4

    ".....", // 5

    "-....", // 6

    "--...", // 7

    "---..", // 8

    "----."  // 9

};

Листинг 3 – Файл *morse.c*

#include "morse.h"

#include "morse\_dictionary.h"

#include <ctype.h>

#include <string.h>

char\* charToMorse(char c) {

    if (isalpha(c)) {

        c = toupper(c);

        return strdup(morseCode[c - 'A']);

    } else if (isdigit(c)) {

        return strdup(morseCode[c - '0' + 26]);

    } else if (c == ' ') {

        return strdup(" ");

    } else {

        return strdup("");

    }

}

Листинг 4 – Файл *makefile*

CC = gcc

TARGET = morse\_convert

$(TARGET): main.o morse.o morse\_dictionary.o

    $(CC) $^ -o $@

main.o: main.c

    $(CC) -c $<

morse.o: morse.c

    $(CC) -c $^

morse\_dictionary.o: morse\_dictionary.c

    $(CC) -c $^

clean:

    rm -f \*.o