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**

Выполнил             В. Д. Ключинский

Проверил                          Н. Ю. Гриценко

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc158376840)

[2 Краткие теоретические сведения 4](#_Toc158376841)

[3 Полученные результаты 6](#_Toc158376842)

[Выводы 9](#_Toc158376843)

[Список использованных источников 10](#_Toc158376844)

[Приложение А (обязательное) Листинг исходного кода 11](#_Toc158376845)

**1 ЦЕЛЬ РАБОТЫ**

Целью выполнения лабораторной работы является изучение среды программирования и основных инструментов: компилятор/сборщик («коллекция компиляторов») *gcc*, управление обработкой проекта *make* (и язык *makefile*), библиотеки. Также необходимо освоить практическое использование основных библиотек и системных вызовов: ввод-вывод и работа с файлами, обработка текста, распределение памяти, управление выполнением.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

*Makefile* – это файл, содержащий набор инструкций, используемых утилитой *make* в инструментарии автоматизации сборки.

Чаще всего *makefile* содержит указания для утилиты *make* о том, как компилировать и компоновать программу. Например, для языков программирования *С/C++* исходный файл *C* или *C++* должен быть перекомпилирован при каждом своем изменении. Если меняется заголовочный файл, каждый *C/C++* исходный файл, включающий его, должен быть перекомпилирован, чтобы быть безопасным. Каждая компиляция создает объектный файл, соответствующий исходному файлу. Наконец, если исходный файл был скомпилирован, то все объектные файлы, будь то вновь созданные или сохраненные с предыдущих компиляций, должны быть скомпонованы вместе, чтобы собрать новый исполняемый файл программы. Эти инструкции вместе со своими зависимостями указаны в *makefile*. Если с последней компиляции ни один из файлов, необходимых для сборки, не менялся, то при компиляции ничего не происходит. Для больших проектов, использование *make*-файлов может значительно сократить время сборки приложений при изменении всего лишь нескольких исходных файлов.[1]

*Make*-файлы возникли на *Unix*-подобных системах и остаются основным средством сборки программ в данных окружениях. В свою очередь, *Windows* поддерживает различные виды *make*-файлов при помощи утилиты *nmake*. Стандартные *make*-файлы из *Unix*-подобных ОС могут быть выполнены в операционной системе *Windows* в среде *Cygwin*.[2]

Make-файлы состоят из явных правил, неявных правил, определений переменных, директив и комментариев.[3]

Для выполнения поставленного задания были использованы следующие инструменты:

1 Команда make для сборки проекта с помощью *makefile*.

2 Команда make clean для удаления промежуточных файлов проекта, чтобы добиться принудительной перекомпиляции всех модулей, даже тех, где не было изменений.

Таким образом, можно понять, что для реализации поставленного задания достаточно изучения основ написания make-файлов.

Среда программирования – это набор инструментов и условий, которые обеспечивают разработку, отладку и выполнение программного обеспечения.

Компилятор/сборщик (*gcc*): *GCC* (*GNU Compiler Collection*) – это свободный компилятор, который поддерживает множество языков, включая C и *C++.* Он преобразует исходный код на языке программирования в машинный код, который может быть выполнен компьютером.

Библиотеки: Библиотеки – это наборы функций и процедур, которые можно использовать в программе. Они обеспечивают повторное использование кода и упрощают процесс разработки.

Ввод-вывод и работа с файлами: Ввод-вывод (I/O) - это процесс передачи данных между компьютером и внешним миром или между различными компонентами компьютера. Работа с файлами включает в себя создание, открытие, чтение, запись и закрытие файлов.

Обработка текста: Обработка текста включает в себя множество операций, таких как поиск и замена текста, сортировка и фильтрация строк, анализ и трансформация текста.

Распределение памяти: Распределение памяти – это процесс выделения и освобождения памяти во время выполнения программы. Это важно для эффективного использования ресурсов и предотвращения утечек памяти.

Управление выполнением: Управление выполнением включает в себя контроль над порядком выполнения инструкций в программе. Это может включать в себя такие вещи, как управление потоками и процессами, обработка исключений и сигналов.

# **3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

В результате выполнения лабораторной работы была скомпилирована программа с помощью *makefile*, написанная на языке *C*, которая заменяет определённые слова в текстовом файле на новые, записанные в словаре.

Результаты работы программы и исходные файлы представлены на рисунках ниже.

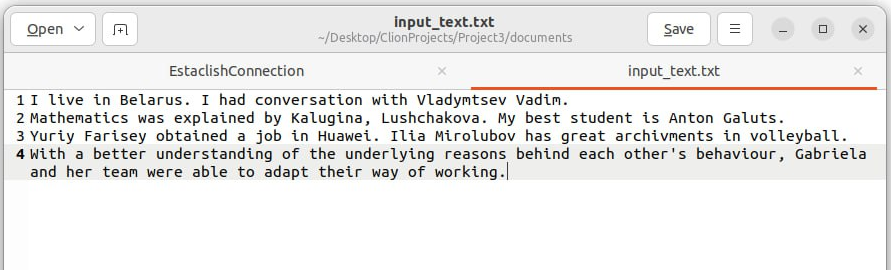


Рисунок 3.1 – Изначальный текст, который подлежит обработке

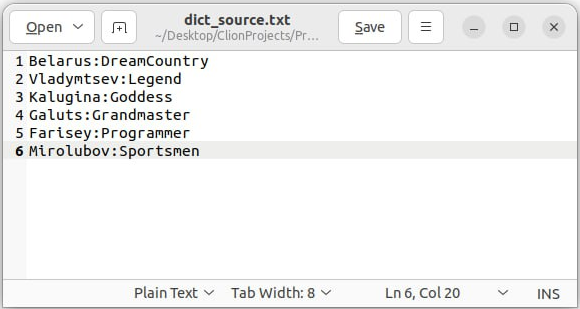


Рисунок 3.2 – Текстовый файл, содержащий слова и их новое значение

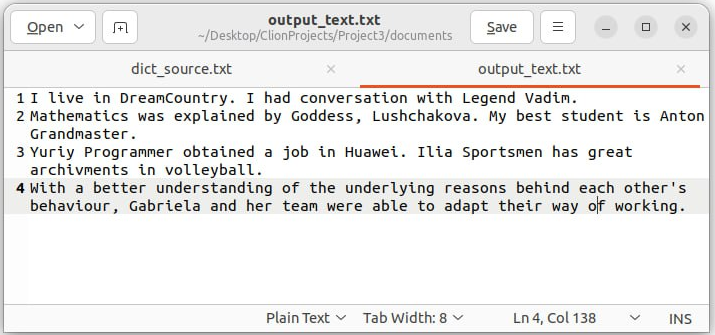


Рисунок 3.3 – Текст после преобразования программой

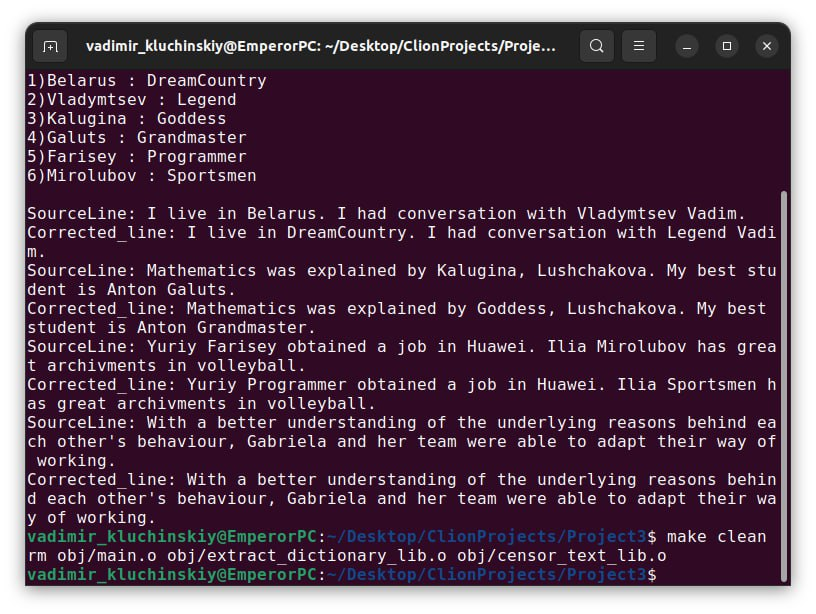


Рисунок 3.4 – Результат работы программы

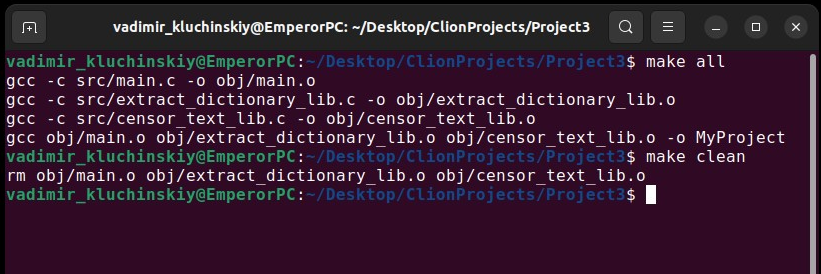


Рисунок 3.5 – Процесс создания и удаления объектных файлов

Таким образом, созданный скрипт реализует поставленную задачу: находит в текстовом файле все вхождения записанных в словаре слов и заменяет их на новые.

# **ВЫВОДЫ**

В ходе выполнения данной лабораторный работы была создана программа написанная на языке *C*, скомпилированная с помощью *makefile* компилятором *gcc*, которая находит в текстовом файле все вхождения записанных в словаре слов и заменяет их на новые с последующей записью в другой текстовый файл. Был изучен механизм выделения памяти, механизм ввода-вывода, а также функции обработки текста в языке *C*.

Для редактирования кода была выбрана IDE «*Clion*».

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Тейнсли, Д. Linux и другие UNIX-подобные операционные системы: программирование в Shеll. / Д. Тейнсли. – СПб.: БВХ-Петербург, 2006. – 1056 с: ил.

[2] Щупаков, Ю. Руководство по командам и shell-программированию, операционная система Linux. / Ю. Щупаков. – СПб: Питер, 2008. – 592 с.

[3] BashTutorials [Электронный ресурс]. – Режим доступа: https://prog /bash-tutorials/. – Дата доступа: 01.02.2024.

# **ПРИЛОЖЕНИЕ А**

# **(обязательное)**

# **Листинг исходного кода**

Листинг 1 – Программный код разработанного приложения:

Файл *censor\_text\_lib.h*:

#ifndef PROJECT3\_CENSOR\_TEXT\_LIB\_H

#define PROJECT3\_CENSOR\_TEXT\_LIB\_H

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "data\_structure.h"

void clean\_file\_sources(FILE\* input\_file, FILE\* output\_file);

void writeStringToFile(FILE\* file, const char\* str);

void replaceWord(char\* str, const char\* oldW, const char\* newW);

void censor\_text(struct Pair\* PairArray, int\* array\_size, const char\* input\_filename, const char\* output\_filename);

#endif //PROJECT3\_CENSOR\_TEXT\_LIB\_H

Файл *extract\_dictionary\_lib.h*:

#ifndef PROJECT3\_EXTRACT\_DICTIONARY\_LIB\_H

#define PROJECT3\_EXTRACT\_DICTIONARY\_LIB\_H

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "data\_structure.h"

#define MAX\_ARRAY\_LENGTH 1000

void clean\_sources(FILE\* file);

struct Pair\* extract\_text(const char\* filename, int\* array\_size0);

#endif //PROJECT3\_EXTRACT\_DICTIONARY\_LIB\_H

Файл *data\_structure\_lib.h*:

#ifndef PROJECT3\_EXTRACT\_DICTIONARY\_LIB\_H

#define PROJECT3\_EXTRACT\_DICTIONARY\_LIB\_H

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "data\_structure.h"

#define MAX\_ARRAY\_LENGTH 1000

void clean\_sources(FILE\* file);

struct Pair\* extract\_text(const char\* filename, int\* array\_size0);

#endif //PROJECT3\_EXTRACT\_DICTIONARY\_LIB\_H

Файл *censor\_text\_lib.c*:

#include "../headers/censor\_text\_lib.h"

void clean\_file\_sources(FILE\* input\_file, FILE\* output\_file){

//free(PairArray);

fclose(input\_file);

fclose(output\_file);

}

void writeStringToFile(FILE\* file, const char\* str) {

fprintf(file, "%s\n", str);

}

void replaceWord(char\* str, const char\* oldW, const char\* newW)

{

char\* result;

int i, cnt = 0;

int newWlen = strlen(newW);

int oldWlen = strlen(oldW);

for (i = 0; str[i] != '\0'; i++) {

if (strstr(&str[i], oldW) == &str[i]) {

cnt++;

// Jumping to index after the old word.

i += oldWlen - 1;

}

}

// Making new string of enough length

result = (char\*)malloc(i + cnt \* (newWlen - oldWlen) + 1);

i = 0;

char \*orig\_str = str;

while (\*str) {

// compare the substring with the result

if (strstr(str, oldW) == str) {

strcpy(&result[i], newW);

i += newWlen;

str += oldWlen;

}

else

result[i++] = \*str++;

}

result[i] = '\0';

str = orig\_str;

i=0;

for (; result[i] != '\0'; ++i) {

str[i] = result[i];

}

str[i] = '\0';

free(result);

}

void censor\_text(struct Pair\* PairArray, int\* array\_size, const char\* input\_filename, const char\* output\_filename) {

FILE\* input\_file = fopen(input\_filename, "r");

if (input\_file == NULL) {

printf("Failed to open input\_file\n");

return;

}

FILE\* output\_file = fopen(output\_filename, "w");

if (output\_file == NULL) {

printf("Failed to open output\_file\n");

return;

}

char line[MAX\_LINE\_LENGTH];

while (fgets(line, MAX\_LINE\_LENGTH, input\_file)) {

line[strcspn(line, "\n")] = '\0';

printf("SourceLine: %s\n", line);

for(int i=0;i<\*array\_size;++i){

replaceWord(line, PairArray[i].first, PairArray[i].second);

}

writeStringToFile(output\_file, line);

printf("Corrected\_line: %s\n", line);

//printf("%s\n", line);

}

clean\_file\_sources(input\_file, output\_file);

}

Файл *extract\_dictionary\_lib.c*:

#ifndef EXTRACT\_DICTIONARY

#define EXTRACT\_DICTIONARY

#include "../headers/extract\_dictionary\_lib.h"

void clean\_sources(FILE\* file){

//free(PairArray);

fclose(file);

}

struct Pair\* extract\_text(const char\* filename, int\* array\_size0) {

FILE\* file = fopen(filename, "r");

if (file == NULL) {

printf("Failed to open file\n");

return NULL;

}

int array\_size = 0;

struct Pair\* PairArray = malloc(sizeof(struct Pair)\*MAX\_ARRAY\_LENGTH);

char line[MAX\_LINE\_LENGTH];

while (fgets(line, MAX\_LINE\_LENGTH, file)) {

line[strcspn(line, "\n")] = '\0';

//printf("%s\n", line);

char\* token = strtok(line, ":");

//printf("FirstToken %s\n", token);

if (token) {

strncpy(PairArray[array\_size].first, token, MAX\_WORD\_LENGTH);

token = strtok(NULL, ":");

//printf("SecondToken %s\n", token);

if (token) {

strncpy(PairArray[array\_size].second, token, MAX\_WORD\_LENGTH);

++array\_size;

} else {

printf("Word has no translation\n");

clean\_sources(file);

return NULL;

}

} else {

printf("Line has no word\n");

clean\_sources(file);

return PairArray;

}

}

clean\_sources(file);

\*array\_size0 = array\_size;

return PairArray;

}

#endif

Файл *main.c*:

#include "../headers/data\_structure.h"

#include "../headers/censor\_text\_lib.h"

#include "../headers/extract\_dictionary\_lib.h"

void display\_array(struct Pair\* PairArray, int\* array\_size){

printf("Dictionary of size: %d\n",\*array\_size);

for(int i=0;i<\*array\_size;++i){

printf("%d)%s : %s\n",i+1,PairArray[i].first,PairArray[i].second);

}

printf("\n");

}

int main(){

const char\* dictionary\_filename = "/home/vadimir\_kluchinskiy/Desktop/ClionProjects/Project3/documents/dict\_source.txt";

int\* array\_size = calloc(1,sizeof(int));

struct Pair\* PairArray = extract\_text(dictionary\_filename,array\_size);

display\_array(PairArray, array\_size);

const char\* input\_document = "/home/vadimir\_kluchinskiy/Desktop/ClionProjects/Project3/documents/input\_text.txt";

const char\* output\_document = "/home/vadimir\_kluchinskiy/Desktop/ClionProjects/Project3/documents/output\_text.txt";

censor\_text(PairArray, array\_size, input\_document, output\_document);

free(PairArray);

free(array\_size);

}

Файл *makefile*:

all:MyProject

MyProject: main.o extract\_dictionary\_lib.o censor\_text\_lib.o

gcc obj/main.o obj/extract\_dictionary\_lib.o obj/сensor\_text\_lib.o -o MyProject

main.o: src/main.c

gcc -c src/main.c -o obj/main.o

extract\_dictionary\_lib.o: src/extract\_dictionary\_lib.c

gcc -c src/extract\_dictionary\_lib.c -o obj/extract\_dictionary\_lib.o

censor\_text\_lib.o: src/сensor\_text\_lib.c

gcc -c src/сensor\_text\_lib.c -o obj/сensor\_text\_lib.o

clean:

rm obj/main.o obj/extract\_dictionary\_lib.o obj/сensor\_text\_lib.o