**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

Курсовая РАБОТА

**по дисциплине «Программирование»**

Тема: Создание программ в Си

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Гребенюк В.А. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2023

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Гребенюк В.А. | | |
| Группа 3341 | | |
| Тема работы: Обработка текста | | |
| Исходные данные:  Вариант 5.2  Вывод программы должен быть произведен в стандартный поток вывода: stdout.  Ввод данных в программе в стандартный поток ввода: stdin.  В случае использования Makefile название исполняемого файла должно быть: cw.  Важно: первой строкой при запуске программы нужно выводить информацию о варианте курсовой работе и об авторе программы в строго определенном формате:  Course work for option <V>, created by <Name> <Surname>.  Где V – вариант курсовой и Имя и Фамилия, как указано в репозитории группы. Данное предложение должно быть строго первым предложением в выводе программы и является отдельной строкой (заканчивается знаком ‘\n’).  Например:  Course work for option 3.2, created by Ivan Ivanov.  Ввод данных:  После вывода информацию о варианте курсовой работе программа ожидает ввода пользователем числа – номера команды:  0 – вывод текста после первичной обязательной обработки (если предусмотрена заданием данного уровня сложности)  1 – вызов функции под номером 1 из списка задания  2 – вызов функции под номером 2 из списка задания  3 – вызов функции под номером 3 из списка задания  4 – вызов функции под номером 4 из списка задания  5 – вывод справки о функциях, которые реализует программа.  Программа не должна выводить никаких строк, пока пользователь не введет число.  В случае вызова справки (опция 5) текст на вход подаваться не должен, во всех остальных случаях после выбора опции должен быть считан текст.  Признаком конца текста считается два подряд идущих символа переноса строки ‘\n’. После каждой из функций нужно вывести результат работы программы и завершить программу.  В случае ошибки и невозможности выполнить функцию по какой-либо причине, нужно вывести строку:  Error: <причина ошибки>  Задание  Каждое предложение должно выводиться в отдельной строке, пустых строк быть не должно. Текст представляет собой предложения, разделенные точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских или кириллических букв, цифр и других символов кроме точки, пробела или запятой. Длина текста и каждого предложения заранее не известна.  Для хранения предложения и для хранения текста требуется реализовать структуры Sentence и Text  Программа должна сохранить (считать) текст в виде динамического массива предложений и оперировать далее только с ним. Функции обработки также должны принимать на вход либо текст (Text), либо предложение (Sentence).  Программа должна найти и удалить все повторно встречающиеся предложения (сравнивать их следует посимвольно, но без учета регистра).  Программа должна выполнить одно из введенных пользователем действий и завершить работу:   1. Вывести все предложения, в которых каждое слово удовлетворяет введенной строке-условию. Строка условия содержит: символы, \* - 0 или больше любых символов, ? – один любой символ. В строке-условия не может быть больше одного \*.  Например, строка условие может иметь вид “?а?а\*н”, которой соответствуют слова “фазан” и “барабан”. 2. Отсортировать предложения по средней длине слов в предложении. 3. Преобразовать предложения так, чтобы слова располагались в порядке уменьшения длины слова. 4. Удалить все предложения, в котором больше 5 или меньше 2 слов.   Все сортировки и операции со строками должны осуществляться с использованием функций стандартной библиотеки. Использование собственных функций, при наличии аналога среди функций стандартной библиотеки, запрещается.  Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов (например, функции обработки текста в один, функции ввода/вывода в другой). Также, должен быть написан Makefile. | | |
| Содержание пояснительной записки:  «Аннотация», «Содержание», «Введение», «Ход выполнения работы», «Заключение», «Список использованных источников», «Пример работы программы», «Исходный код программы». | | |
|  | | |
| Дата выдачи задания: 16.10.2023 | | |
| Дата сдачи реферата: 28.11.2023 | | |
| Дата защиты реферата: 30.11.2023 | | |
| Студент |  | Гребенюк В.А. |
| Преподаватель |  | Глазунов С.А. |

**Аннотация**

Курсовой проект по варианту 5.2 представляет собой программу, которая обрабатывает текст, введенный пользователем. Программа использует стандартные потоки ввода и вывода, а также Makefile для компиляции. Программа предлагает пользователю выбор из нескольких функций для обработки текста, включая удаление повторяющихся предложений, сортировку предложений по средней длине слов, преобразование предложений так, чтобы слова располагались в порядке уменьшения длины слова, и удаление предложений, в которых больше 5 или меньше 2 слов. Все эти функции реализованы с использованием функций стандартной библиотеки. Каждая подзадача вынесена в отдельную функцию, а функции сгруппированы в несколько файлов. Также написан Makefile. Программа завершает работу после выполнения одного из действий, выбранных пользователем.

Исходный код программы: Приложение А.

Демонстрация работы программы: Приложение

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 7 |
| 1. | Ход выполнения работы | 8 |
| 1.1. | Структуры данных и функции к ним | 8 |
| 1.2. | Ввод/вывод и первичная обработка | 9 |
| 1.3. | Функции для задания | 10 |
| 1.4. | Makefile | 11 |
|  | Заключение | 12 |
|  | Список использованных источников | 13 |
|  | Приложение А. Исходный код программы | 14 |
|  | Приложение Б. Демонстрация работы программы | 0 |

**введение**

Цель данной работы - разработать программу для обработки текста, введенного пользователем, с использованием стандартных потоков ввода и вывода и Makefile для компиляции.

Для достижения поставленной цели требуется решить следующие задачи:

1. Изучение стандартных потоков ввода и вывода.
2. Разработка функций для обработки текста, включая удаление повторяющихся предложений, сортировку предложений по средней длине слов, преобразование предложений так, чтобы слова располагались в порядке уменьшения длины слова, и удаление предложений, в которых больше 5 или меньше 2 слов.
3. Сборка программы с использованием Makefile.
4. Тестирование программы на различных входных данных.

**1. Ход выполнения работы**

* 1. **Структуры данных и функции к ним**

Данные структуры и функции находятся в файлах: vector.c vector.h words.c words.h

Структуры:

struct Sentence или String: структура для хранения строки с поддержкой динамической аллокации, количества слов и средней длинны слова.

struct Text: структура для хранения строк

struct Words: структура, хранящая слова и разделители

Функции:

*void ERROR(char\* message, int code): вывод ошибки и выход с кодом*

*String\*\* new\_string(unsigned int \_size):* аллоцировать и вернуть строку

*String\*\* extend\_string(String\*\* \_str):* реаллоцировать память структуры *String* (увеличить в 2 раза)

*void append\_wchar(Text\*\* \_text, wchar\_t \_ch):* добавить символ в конец строки

*void clear(String\*\* \_str):* заменить строку на пустую (с освобождением памяти)

*Text\*\* new\_text(unsigned int \_size):* аллоцировать и вернуть *Text*

*Text\*\* extend\_text(Text\*\* \_text):* реаллоцировать память структуры *Text* (увеличить в 2 раза*)*

*void append\_new\_string(Text\*\* \_text):* Добавить в текст пустую строку

*String\*\* last\_string(Text\*\* \_text)*: последняя строка в тексте

*void free\_text(Text\*\* \_text)*: освобождение памяти занимаемой структурой *Text*

* 1. **Ввод/вывод и первичная обработка**

Данные функции находятся в файлах: io.c main.c

Функции:

*void read\_text(Text\*\* \_text)*: Считывание текста (предложений)

*void output\_text(Text\*\* \_text)*: Вывод текста (не пустых предложений)

*void remove\_duplicats(Text\*\* \_text)*: Удаление повторяющихся предложений без учёта регистра

*int main(void)*: точка входа в программу, проверка локали, вывод информации о программе

* 1. **Функции для задания**

Данные функции находятся в файлах: exec\_comand.c regex\_task.c sorts.c

Функции:

*void print\_help():* вывод справки о функциях

*void exec\_command():* запрос ввода команды и выполнение её

*void regex\_task(Text\*\* \_text, wchar\_t\* expression)*: замена предложений в которых все слова подходят под строку условие на пустые

*int check\_word(wchar\_t\* word, wchar\_t\* expression):* проверка слова строкой условием

*int compare\_avglen(const void\* \_word1, const void\* \_word2):* сравнение средней длинны слов в двух предложениях (для *qsort*)

*int compare\_len(const void\* \_str1, const void\* \_str2):* сравнение длинны двух слов(для *qsort*)

*void sort\_text\_all(Text\*\* \_text)*: сортировка слов по длине в порядке уменьшения во всех предложениях текста (кроме пустых)

*void sort\_words(String\*\* \_str)*: сортировка слов по длине в порядке уменьшения

* 1. **Makefile**

Вызов *make*:

*all* (default): компиляция всей программы (зависит от *$(EXE)*)

run: выполнение программы (зависит от *$(EXE)*)

*clear*: удаление *$(OBJ)*) файлов и *$(EXE)*

SRC = $(wildcard \*.c) $(wildcard \*\*/\*.c): все \*.c файлы

*EXE* *=* *cw*: название исполняемого файла

*OBJ = $(SRC:.c=.o)*: список скомпилированных файлов (путём замены .c на .o)

*CC* *=* *gcc*:компилятор

*CFLAGS* = *-Ofast -flto -pipe*: флаги компилятора

**заключение**

Обработка текста на Си успешно освоена, задания курсовой выполнены.

Лучше не использовать Си для обработки текста, если в этом нет необходимости.

**список использованных источников**

**приложение А**

**Исходный код программы**

Название файла: main.c

#include <stdio.h>

#include <locale.h>

#include <stdio.h>

#include "exec\_command.h"

#include "util/config.h"

#include "util/error.h"

int main(void) {

if (setlocale(LC\_ALL, LOCALE) == NULL) {

printf("Current locale: %s\n", setlocale(LC\_ALL, NULL));

ERROR("No " LOCALE " encoding", 1);

}

puts("Course work for option 5.2, created by Vadim Grebenyuk.");

exec\_command();

return 0;

}

Название файла: exec\_command.c

#include "exec\_command.h"

#include <locale.h>

#include <stdio.h>

#include <stdlib.h>

#include <wchar.h>

#include "regex\_task.h"

#include "sorts.h"

#include "util/config.h"

#include "util/error.h"

#include "util/io.h"

#include "util/vector.h"

#define REGEX\_MAX 256

void print\_help();

void exec\_command() {

setlocale(LC\_ALL, LOCALE);

wchar\_t\* regex;

static Text\* text;

int command;

if (!wscanf(L"%d", &command))

ERROR("Command should be a number", 2);

getwchar(); // skip newline

if (command == 5) {

print\_help();

return;

}

text = \*new\_text(MIN\_VECTOR\_SIZE);

switch (command) {

case 0:

read\_text(&text);

break;

case 1:

regex = calloc(REGEX\_MAX, sizeof(wchar\_t));

if (regex == NULL)

ERROR("Failed to allocate regex string", 1);

wscanf(L"%ls ", regex);

read\_text(&text);

regex\_task(&text, regex);

free(regex);

break;

case 2:

read\_text(&text);

sort\_avgwordlen(&text);

break;

case 3:

read\_text(&text);

sort\_text\_all(&text);

break;

case 4:

read\_text(&text);

for (unsigned int i = 0; i < text->len; i++)

if (text->body[i]->words < 2 || text->body[i]->words > 5)

clear(&text->body[i]);

break;

default:

ERROR("Wrong command", 2);

break;

}

output\_text(&text);

free\_text(&text);

}

void print\_help() {

puts(

"commands:\n"

" - 0 | remove duplicates\n"

" - 1 | filter by regex\n"

" - 2 | sort by avg word len\n"

" - 3 | sort words by len (descending)\n"

" - 4 | remove sentences with less than 2\n"

" | or more than 5 words\n"

" - 5 | this message\n");

}

Название файла: exec\_command.h

#pragma once

void exec\_command();

Название файла: regex\_task.c

#include "regex\_task.h"

#include <locale.h>

#include <stdlib.h>

#include <wchar.h>

#include "util/config.h"

#include "util/error.h"

int check\_word(wchar\_t\* word, wchar\_t\* expression);

void regex\_task(Text\*\* \_text, wchar\_t\* expression) {

const wchar\_t\* seps = L" ,.";

wchar\_t\* context = NULL;

setlocale(LC\_ALL, LOCALE);

for (unsigned int i = 0; i < (\*\_text)->len; i++) {

char all\_match = 1;

wchar\_t\* str\_dup = wcsdup((\*\_text)->body[i]->body);

wchar\_t\* word = wcstok(str\_dup, seps, &context);

while (word != NULL) {

if (!check\_word(word, expression)) {

all\_match = 0;

break;

}

word = wcstok(NULL, seps, &context);

}

free(str\_dup);

if (!all\_match)

clear(&(\*\_text)->body[i]);

}

}

int check\_word(wchar\_t\* word, wchar\_t\* expression) {

int n = 0;

int match = 0;

if (expression[n] == L'\*' && expression[n + 1] == L'\0' && word[n] != L'\0' ||

(expression[n] == L'\0' && word[n] == L'\0')) {

return 1;

}

while (expression[n] != L'\0' && word[n] != L'\0') {

if (expression[n] == L'\*') {

if (word[n + 1] == L'\0')

return 1;

for (unsigned int i = 0; word[i] != L'\0'; i++) {

if (check\_word(&word[n + i], &expression[n + 1]))

return 1;

}

return 0;

} else if (expression[n] == L'?' || expression[n] == word[n]) {

n++;

} else {

return 0;

break;

}

match = 1;

}

if (expression[n] == L'\0' && word[n] != L'\0')

return 0;

return match;

}

Название файла: regex\_task.h

#pragma once

#include "util/vector.h"

#include <stddef.h>

void regex\_task(Text\*\* \_text, wchar\_t\* expression);

Название файла: sorts.c

#include "sorts.h"

#include <stdlib.h>

#include "util/vector.h"

#include "words.h"

int compare\_avglen(const void\* \_word1, const void\* \_word2) {

return ((\*(String\*\*)\_word1)->avglen > (\*(String\*\*)\_word2)->avglen) - ((\*(String\*\*)\_word1)->avglen < (\*(String\*\*)\_word2)->avglen);

}

int compare\_len(const void\* \_str1, const void\* \_str2) {

return ((\*(String\*\*)\_str1)->len < (\*(String\*\*)\_str2)->len) - ((\*(String\*\*)\_str1)->len > (\*(String\*\*)\_str2)->len);

}

void sort\_avgwordlen(Text\*\* \_text) {

for (unsigned int i = 0; i < (\*\_text)->len; i++) {

Words\* \_tmp = \*from\_string\_with\_sep(&(\*\_text)->body[i]);

(\*\_text)->body[i]->avglen = avg\_len(\_tmp);

free\_words(&\_tmp);

}

qsort((\*\_text)->body, (\*\_text)->len, sizeof(String\*), compare\_avglen);

}

void sort\_text\_all(Text\*\* \_text) {

for (unsigned int i = 0; i < (\*\_text)->len; i++)

if ((\*\_text)->body[i]->len != 0)

sort\_words(&(\*\_text)->body[i]);

}

void sort\_words(String\*\* \_str) {

Words\* \_words = \*from\_string\_with\_sep(\_str);

qsort(\_words->words->body, \_words->words->len, sizeof(String\*), compare\_len);

free(\*\_str);

\*\_str = \*to\_string(&\_words);

free\_words(&\_words);

}

Название файла: sorts.h

#pragma once

#include "util/vector.h"

void sort\_avgwordlen(Text\*\* \_text);

void sort\_text\_all(Text\*\* \_text);

void sort\_words(String\*\* \_str);

Название файла: words.c

#include "words.h"

#include <stdlib.h>

#include <wchar.h>

#include "util/error.h"

#include "util/vector.h"

inline float avg\_len(Words\* \_words);

Words\*\* from\_string\_with\_sep(String\*\* \_str) {

static Words\* \_new\_words;

\_new\_words = calloc(1, sizeof(Words));

if (\_new\_words == 0) {

ERROR("Failed to allocate new words", 1);

exit(0); // fix warning

}

\_new\_words->words = \*new\_text((\*\_str)->words + 1);

\_new\_words->separators = \*new\_text((\*\_str)->words + 1);

\_new\_words->strlen = (\*\_str)->len;

int sep = 1;

for (unsigned int i = 0; i < (\*\_str)->len; i++) {

switch ((\*\_str)->body[i]) {

case L' ':

case L',':

case L'.':

if (sep != 1) {

append\_new\_string(&\_new\_words->separators);

sep = 1;

}

append\_wchar(&\_new\_words->separators, (\*\_str)->body[i]);

break;

default:

if (sep != 0) {

append\_new\_string(&\_new\_words->words);

sep = 0;

}

append\_wchar(&\_new\_words->words, (\*\_str)->body[i]);

break;

}

}

return &\_new\_words;

}

String\*\* to\_string(Words\*\* \_words) {

static String\* \_new\_string;

\_new\_string = \*new\_string((\*\_words)->strlen + 2);

\_new\_string->len = (\*\_words)->strlen;

\_new\_string->words = (\*\_words)->words->len;

for (unsigned int i = 0; i < (\*\_words)->words->len; i++) {

wcscat(\_new\_string->body, (\*\_words)->words->body[i]->body);

wcscat(\_new\_string->body, (\*\_words)->separators->body[i]->body);

}

return &\_new\_string;

}

float avg\_len(Words\* \_words) {

float sum = 0;

if (\_words->words->len == 0)

return 0;

for (unsigned int i = 0; i < \_words->words->len; i++)

sum += \_words->words->body[i]->len;

return sum / \_words->words->len;

}

void free\_words(Words\*\* \_words) {

free\_text(&(\*\_words)->words);

free\_text(&(\*\_words)->separators);

free(\*\_words);

}

Название файла: words.h

#pragma once

#include "util/vector.h"

typedef struct Words {

Text\* words;

Text\* separators;

unsigned int strlen;

} Words;

Words\*\* from\_string\_with\_sep(String\*\* \_str);

String\*\* to\_string(Words\*\* \_words);

void free\_words(Words\*\* \_words);

float avg\_len(Words\* \_words);

Название файла: io.c

#include "io.h"

#include <locale.h>

#include <stdio.h>

#include <wchar.h>

#include <wctype.h>

#include "config.h"

#include "vector.h"

#define PARSE\_ELLIPSIS 0

void remove\_duplicats(Text\*\* \_text);

void read\_text(Text\*\* \_text) {

setlocale(LC\_ALL, LOCALE);

append\_new\_string(\_text);

int newline = 0;

int ended = 0;

int dots = 0;

int space = 0;

int word = 0;

for (wchar\_t ch = getwchar(); !(newline == 1 && ch == L'\n'); ch = getwchar()) {

switch (ch) {

case L'.':

dots++;

word = 0;

space = 0;

#if !PARSE\_ELLIPSIS

if (dots < 2)

#endif

append\_wchar(\_text, ch);

if (dots)

ended = 1;

#if PARSE\_ELLIPSIS

if (dots == 3)

ended = 0;

#endif

break;

case L'\n':

newline++;

if (!space && !ended) {

append\_wchar(\_text, L' ');

space = 1;

}

break;

case L'\t':

case L' ':

case L',':

#if PARSE\_ELLIPSIS

if (dots == 1)

#else

if (dots)

#endif

ended = 1;

else

dots = 0;

space = 1;

word = 0;

if (!ended)

append\_wchar(\_text, ch);

break;

default:

if (ended)

append\_new\_string(\_text);

if (!word) {

(\*last\_string(\_text))->words++;

word = 1;

}

ended = 0;

newline = 0;

dots = 0;

space = 0;

append\_wchar(\_text, ch);

break;

}

}

remove\_duplicats(\_text);

}

void output\_text(Text\*\* \_text) {

setlocale(LC\_ALL, LOCALE);

for (unsigned int i = 0; i < (\*\_text)->len; i++)

if ((\*\_text)->body[i]->len != 0)

printf("%ls\n", (\*\_text)->body[i]->body);

}

void remove\_duplicats(Text\*\* \_text) {

for (unsigned int i = 0; i < (\*\_text)->len - 1; i++)

if ((\*\_text)->body[i]->len != 0)

for (unsigned int j = i + 1; j < (\*\_text)->len; j++)

if (wcscasecmp((\*\_text)->body[i]->body, (\*\_text)->body[j]->body) == 0)

clear(&(\*\_text)->body[j]);

};

Название файла: io.h

#pragma once

#include "vector.h"

void read\_text(Text\*\* \_text);

void output\_text(Text\*\* \_text);

Название файла: vector.c

#include "vector.h"

#include <stdlib.h>

#include <string.h>

#include <wchar.h>

#include "error.h"

String\*\* last\_string(Text\*\* \_text) {

return &(\*\_text)->body[(\*\_text)->len - 1];

}

String\*\* extend\_string(String\*\* \_str) {

static String\* \_new\_str;

\_new\_str = \*new\_string((\*\_str)->allocated \* 2);

wmemcpy(\_new\_str->body, (\*\_str)->body, (\*\_str)->len);

\_new\_str->len = (\*\_str)->len;

\_new\_str->words = (\*\_str)->words;

free(\*\_str);

return &\_new\_str;

}

void clear(String\*\* \_str) {

free(\*\_str);

\*\_str = \*new\_string(MIN\_VECTOR\_SIZE);

}

Text\*\* extend\_text(Text\*\* \_text) {

static Text\* \_new\_text;

\_new\_text = \*new\_text((\*\_text)->allocated \* 2);

memcpy(\_new\_text->body, (\*\_text)->body, (\*\_text)->len \* sizeof(String\*));

\_new\_text->len = (\*\_text)->len;

free(\*\_text);

return &\_new\_text;

}

String\*\* new\_string(unsigned int \_size) {

static String\* \_new\_str;

\_new\_str = calloc(sizeof(String) + \_size \* sizeof(wchar\_t), 1);

if (\_new\_str == NULL) {

ERROR("Failed to allocate new string", 1);

exit(0); // fix warning

}

\_new\_str->allocated = \_size;

\_new\_str->len = 0;

return &\_new\_str;

}

Text\*\* new\_text(unsigned int \_size) {

static Text\* \_new\_text;

\_new\_text = calloc(sizeof(Text) + \_size \* sizeof(String\*), 1);

if (\_new\_text == NULL) {

ERROR("Failed to allocate new text", 1);

exit(0); // fix warning

}

\_new\_text->allocated = \_size;

return &\_new\_text;

}

void append\_wchar(Text\*\* \_text, wchar\_t \_ch) {

String\*\* \_last = last\_string(\_text);

if ((\*\_last)->allocated == (\*\_last)->len + 1)

\*\_last = \*extend\_string(\_last);

(\*\_last)->body[(\*\_last)->len++] = \_ch;

};

void append\_new\_string(Text\*\* \_text) {

if ((\*\_text)->allocated == (\*\_text)->len)

\*\_text = \*extend\_text(\_text);

(\*\_text)->len++;

\*last\_string(\_text) = \*new\_string(MIN\_VECTOR\_SIZE);

};

void free\_text(Text\*\* \_text) {

for (unsigned int i = 0; i < (\*\_text)->len; i++)

free((\*\_text)->body[i]);

free(\*\_text);

}

Название файла: vector.h

#pragma once

// #include <stddef.h>

#define MIN\_VECTOR\_SIZE 4

#include <stddef.h>

typedef struct Sentence {

unsigned int len;

unsigned int allocated;

unsigned int words;

float avglen;

wchar\_t body[];

} String;

typedef struct Text {

unsigned int len;

unsigned int allocated;

String\* body[];

} Text;

String\*\* new\_string(unsigned int \_size);

String\*\* extend\_string(String\*\* \_str);

void append\_wchar(Text\*\* \_text, wchar\_t \_ch);

void clear(String\*\* \_str);

Text\*\* new\_text(unsigned int \_size);

Text\*\* extend\_text(Text\*\* \_text);

void append\_new\_string(Text\*\* \_text);

String\*\* last\_string(Text\*\* \_text);

void free\_text(Text\*\* \_text);

Название файла: error.c

#include "error.h"

#include <stdio.h>

#include <stdlib.h>

#define ERROR\_STD stdout

void ERROR(char\* message, int code) {

fprintf(ERROR\_STD, "Error: %s\n", message);

exit(code);

}

Название файла: error.h

#pragma once

void ERROR(char\* message, int code);

Название файла: config.h

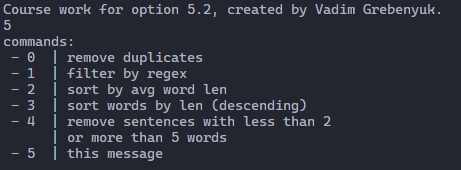
#pragma once

#define LOCALE "C.UTF-8"

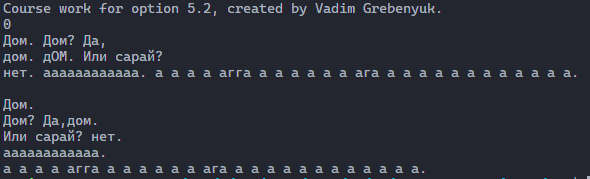
**приложение б**

**Демонстрация работы программы**

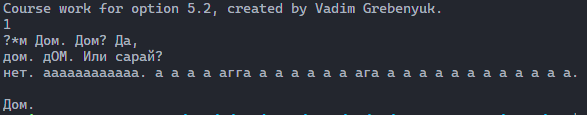
Команда 5:

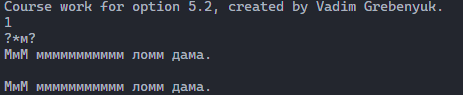


Команда 0:

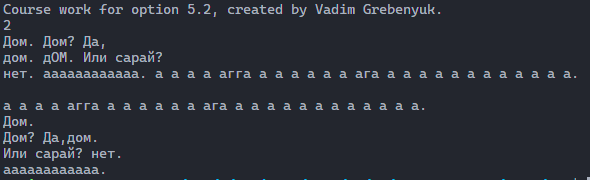


Команда 1:

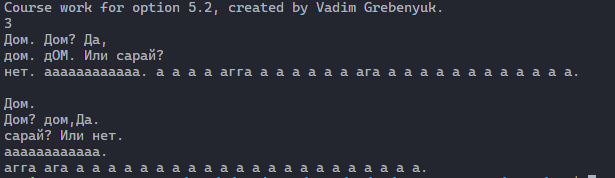




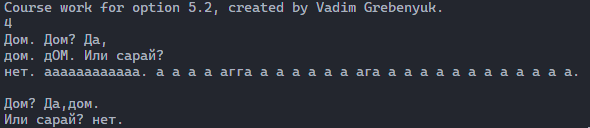
Команда 2:



Команда 3:



Команда 4:



Неверная комманда:



