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

Гребенюк В.А.

Студент гр. 3341 _____

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

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

1.1. 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*

1.2. **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

1.3. 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

1.4. 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:

```
Course work for option 5.2, created by Vadim Grebenyuk.
5
commands:
 - 0  | remove duplicates
 - 1  | filter by regex
 - 2  | sort by avg word len
 - 3  | sort words by len (descending)
 - 4  | remove sentences with less than 2
      | or more than 5 words
 - 5  | this message
```

Команда 0:

```
Course work for option 5.2, created by Vadim Grebenyuk.
0
Дом. Дом? Да,
дом. дОМ. Или сарай?
нет. ааааааааааа. а а а а агга а а а а а а ага а а а а а а а а а а а.

Дом.
Дом? Да,дом.
Или сарай? нет.
ааааааааааа.
а а а а агга а а а а а а ага а а а а а а а а а а а.
```

Команда 1:

```
Course work for option 5.2, created by Vadim Grebenyuk.
1
?*м Дом. Дом? Да,
дом. дОМ. Или сарай?
нет. ааааааааааа. а а а а агга а а а а а а ага а а а а а а а а а а а.

Дом.
```

```
Course work for option 5.2, created by Vadim Grebenyuk.
1
?*м?
МмМ ммммммммммм ломм дама.

МмМ ммммммммммм ломм дама.
```

Команда 2:

```
Course work for option 5.2, created by Vadim Grebenyuk.
2
Дом. Дом? Да,
дом. дОМ. Или сарай?
нет. аааааааааааа. а а а а агга а а а а а а ага а а а а а а а а а а а.

а а а а агга а а а а а а а ага а а а а а а а а а а а.
Дом.
Дом? Да,дом.
Или сарай? нет.
аааааааааааа.
```

Команда 3:

```
Course work for option 5.2, created by Vadim Grebenyuk.
3
Дом. Дом? Да,
дом. дОМ. Или сарай?
нет. аааааааааааа. а а а а агга а а а а а а ага а а а а а а а а а а а.

Дом.
Дом? дом,Да.
сарай? Или нет.
аааааааааааа.
агга ага а а а а а а а а а а а а а а а а а а а а а.
```

Команда 4:

```
Course work for option 5.2, created by Vadim Grebenyuk.
4
Дом. Дом? Да,
дом. дОМ. Или сарай?
нет. аааааааааааа. а а а а агга а а а а а а ага а а а а а а а а а а а.

Дом? Да,дом.
Или сарай? нет.
```

Неверная комманда:

```
Course work for option 5.2, created by Vadim Grebenyuk.
Дом
Error: Command should be a number
```

```
Course work for option 5.2, created by Vadim Grebenyuk.
22
Error: Wrong command
```