МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка текстовых данных

Студент гр. 1304 _____ Байков Е.С.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2021

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Байков Е.С.

Группа 1304
Тема работы: обработка текстовых данных

Исходные данные:
Программе на вход подается текст (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.
Технические требования:
Текст должен быть сохранен в динамический массив строк.
Содержание пояснительной записки:
«Содержание», «Введение», «Основные теоретические положения», «Реализация программы», «Тестирование», «Пользовательская инструкция», «Заключение», «Список использованных источников»

Дата выдачи задания: 15.10.2021

Дата сдачи реферата: 21.12.2021

Дата защиты реферата: 23.12.2021

Студент _____ Байков Е.С.

Преподаватель _____ Чайка К.В

АННОТАЦИЯ

В данной курсовой работе была реализована программа, предлагающая пользователю ввести текст, а затем обработать его. Она запрашивает у него одну из команд, а затем выполняет их. Были реализованы следующие функции: вывод предложений анаграмм; сортировка предложений по количеству заглавных букв в предложениях; замена гласной буквы двумя другими, следующими далее по алфавиту; замена одного слова, заданного пользователем, на другое, заданное пользователем. Также каждая подзадача вынесена в отдельный файл и создан Makefile.

SUMMARY

In this course work, a program was implemented that prompts the user to enter text and then process it. She asks him for one of the commands, and then executes them. The following functions were implemented: output of anagram sentences; sorting sentences by the number of capital letters in sentences; replacement of a vowel letter with two other letters following the alphabet; replacement of one word specified by the user with another specified by the user. Also, each subtask is placed in a separate file and a Makefile is created.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

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. Далее, программа должна запрашивать у пользователя одно из следующих доступных действий и, впоследствии, выполнить его.

На основе выше изложенной цели, были сформулированы следующие задачи:

1. Изучение особенностей работы со строками и символами в языке С
2. Изучение особенностей работы с динамической памятью
3. Изучение работы со структурами
4. Изучение использования указателей и массивов
5. Использование функций стандартных библиотек
6. Создание диалогового окна

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

1.1. Строки в языке программирования С

Строки в С, как и в большинстве языков программирования высокого уровня рассматриваются как отдельный тип, входящий в систему базовых типов языка. Так как язык С по своему происхождению является языком системного программирования, то строковый тип данных в С как таковой отсутствует, а в качестве строк в С используются обычные массивы символов, заканчивающийся нулевым символом.

1.2. Используемые библиотеки

В данной курсовой работе использовались такие библиотеки, как: wchar.h, stdlib.h, string.h, wctype.h,

Из библиотеки wchar.h были использованы следующие функции:

- getwchar() – используется для считывания широкого символа из стандартного потока ввода;
- wprintf() – используется для вывода переданных аргументов в виде строки в стандартный поток вывода;
- wcschr() – используется для проверки вхождения символа в строку;
- wcsncpy() – используется для копирования строки широких символов в количестве n символов;
- wcsncpy() – используется для проверки равенства строк по n символов;
- wcstok() – используется для разбиения строки по символу;
- wcslen() – используется для получения длины строки широких символов;

Из библиотеки stdlib.h были использованы следующие функции:

- malloc() – используется для динамического выделения блока памяти;

- realloc() – используется для изменения величины ранее выделенной памяти;
- calloc() - используется для динамического выделения блока памяти;
- free() – используется для возвращения памяти в кучу;

Из библиотеки string.h были использованы следующие функции:

- memmove() – используется для копирования n количество байт из одной области памяти в другую;

Из библиотеки locale.h были использованы следующие функции:

- setlocale() – устанавливает локаль.

Из библиотеки wctype.h были использованы следующие функции:

- iswupper() – проверяет находится ли широкий символ в верхнем регистре.

2. РЕАЛИЗАЦИЯ ПРОГРАММЫ

2.1. Функция main

В главной функции устанавливается локаль с помощью функции *setlocale*, создается переменная *struct Text text* в которую передается значение функции, которая возвращает *struct Text*. После в поле структуры *Text*, которое отвечает за длину теста передается значение функции *deleteSentences.* Далее с помощью цикла *while* и конструкции *switch-case* производится операции над текстом, как только пользователь решит выйти, текст распечатается в последний раз, отчистится память, выделенная под хранение текста и программа завершает свою работу.

2.2. Функция считывания текста

Текст считывается при помощи трех функции и сохраняется в структуры *Text* и *Sentence*. Структура, хранящая в себе текст, обладает массивом указателей *struct Sentence** *sentences*, полем *len*, в котором находится количество предложений и *size* – где хранится объем буфера, для массива предложений. Структура, которая хранит в себе предложения обладает полями *string* – указатель на массив символов (предложение)*, size* – размер буфера*, len* – длина предложения*, count* – количество гласных букв в предложении*, count_upper_symbol* – количество букв в верхнем регистре. Функция *readSentence* возвращает указатель на структуру. Ее задача считать предложение до того момента пока не встретиться точка. Функция *readText* сохраняет в структуру *Text* все предложения, которые считывает с помощью предыдущей функцией, до того момента пока не встретит пустую строку – символ окончания считывания.

2.3.Функция удаления повторно встречающихся предложений

deleteSentences():

Удаление происходит с помощью двух циклов *while*, внутри которых сравниваются поля структуры *Sentence* с помощью функции *wcscasecmp*, в случае если она находит похожее предложение, память, отведенная под него

очищается и с помощью *memmove* все предложения сдвигаются. Когда все предложения, похожие на данное, удалены, удаляется и данное предложение таким же способом.

2.4. Вывод анаграмм

findAnagram():

Функция получает на вход структуру *Text*. Функция попарно перебирает предложения, сравнивая их копии, преобразованные в строку, состоящую из букв и цифр каждого предложения игнорируя пробелы, запятые и точки, с помощью функции *compareLetters()*. Если набор букв и цифр в копиях одинаковы то функция печатает предложения.

2.5. Сортировка предложений по количеству заглавных букв

sortSentences():

Функция с помощью функции *findUpper* ищет количество заглавных букв в предложении и сохраняет в поле *count_upper_symbol* структуры *Sentence*. Затем с помощью *qsort* массив предложений сортируется, используя функции *compareSentences*.

2.6. Замена каждой гласной на две последующих буквы алфавита

addLetters():

Функция считает количество гласных букв с помощью функции *isvowel()*, которая проверяет букву на гласность. Затем память для предложения расширяется на количество гласных букв. Затем предложение переписывается с помощью цикла *while* вспомогательной строки.

2.7. Замена слова (введенного пользователем) на другое слово (введенное пользователем)

replace():

Функция считывает два слова с помощью функции *readWord()*. Затем создает две копии предложения и делят одну копию на слова с помощью *wcstok()*. Затем функция сверяет слова в предложении со словом, которое задал пользователь и в случае нахождения, увеличивает или уменьшает память для предложения записывая его во вторую копию, а затем очищает старое

предложение и вписывает новое. Так повторяется пока не закончатся все заданные пользователем слова во всех предложениях

2.8.Функция вывода

printText():

С помощью цикла *for* выводит на экран предложения.

2.9.Функция освобождения памяти

freeText():

С помощью цикла *for* освобождает память каждого предложения, структуры.

2.10 Обработка ошибок

Функции выделяющие память возвращают NULL или 0 в случае когда память не удалось выделить и тогда программа завершается.

3. ТЕСТИРОВАНИЕ

2.1. Тестирование функционала

На вход программе подается различный текст, проверяется корректность работы функций.

Ниже приведены результаты тестирования:

1) Вывод анаграмм

№ Теста	**ввод**	**вывод**
1	Reducto anagram del. ductoRe agraman led. Der. Der. Der. Wetrix meh yu. trixWe hem yu. Exxx.	Reducto anagram del. -- ductoRe agraman led. -- Anagrams Wetrix meh yu. -- trixWe hem yu. -- Anagrams

2) Сортировка предложений по числу заглавных букв

№ Теста	**ввод**	**вывод**
1	TRYSORTME. Ok. It will not BE DifficulT. OR It will be hard.	Ok. OR It will be hard. It will not BE Difficult. TRYSORTME.

3) Замена гласных букв на две, идущие следующими по алфавиту

№ Теста	**ввод**	**вывод**
1	Now I do not understand this sentence. Oh NO.	Npqw JK dpq npqt vwndfgrstbcnd thjks sfgntfgncfg. PQh NPQ.

4) Замена вхождения слова (заданного пользователем) на другое слово(заданное пользователем)

№ Теста	**Ввод**	**вывод**
1	Ok. OR It will be hard. It will not BE Difficult. TRYSORTME.	Ok. OR It WERE be hard. It WERE not BE Difficult.

	will WERE	TRYSORTME.
2	I will get a bad mark. bad GOOD	I will get a GOOD mark

4.ПОЛЬЗОВАТЕЛЬСКАЯ ИНСТРУКЦИЯ

1) Откройте терминал и запустите программу из папки с помощью команды make && ./program

2) Требуется ввести текст, который вы хотите обработать.

   Текст должен представлять предложения, оканчивающиеся на точку. Предложения - набор слов, разделенные пробелом или запятой, слова – набор кириллических букв, цифр и других символов кроме точки, пробела или запятой. Конечное предложение – два переноса строки, является символом окончания записывания текста.

3) Нужно выбрать одну из предложенных команд.

   На экране отобразится результат.

   Если введенная вами команда не будет соответствовать ни одной из предложенных команд, то программа выведет ошибку вида «Неверная команда».

4) После выполнения команды программа предложит повторно выбрать команду. Если вы захотите завершить работу программу, необходимо выбрать команду номер 5.

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы были изучены основы работы со строковыми данными в языке С, освоена работа со структурами, работа с динамической памятью, ее выделение, добавление и освобождение, наработаны навыки работы с динамическими массивами и с указателями.

Разработана программа, считывающая текст и производящая над ним операции, которые запрашиваются у пользователя. Были использованы функции стандартных библиотек и разработаны собственные. Было создано диалоговое окно, а также был создан Makefile для удобной сборки программы, состоящей из нескольких файлов, в каждом из которых хранится отдельная подзадача.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Сайт cplusplus.com:* *https://www.cplusplus.com*
2. *Сайт Википедия:* *https://ru.wikipedia.org/wiki/Заглавная_страница*
3. *Сайт cppstudia.com:* *http://cppstudio.com/cat/309/*

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>
#include <wctype.h>
#include <string.h>
#include "structures.h"
#include "addLetters.h"
#include "findAnagram.h"
#include "sortSentences.h"
#include "replace.h"
#define MEM_STEP 20

wchar_t delSpace(){
    wchar_t c = getwchar();
    while(c == L' '){
        c = getwchar();
    }
    return c;
}

struct Sentence* readSentence(){
    int size = MEM_STEP;
    wchar_t* t = malloc(size*sizeof(wchar_t));
    wchar_t* buf;
    if(t != NULL){
        buf = t;
    }
    else{
        wprintf(L"Error. readSentence.");
        return NULL;
    }
    int n = 0;
    wchar_t temp;
    do{
        if(n >= size - 2){
            size += MEM_STEP;
            t = realloc(buf, size*sizeof(wchar_t));
            if(t != NULL){
```

```
                buf = t;
            }
            else{
                free(buf);
                return NULL;
            }
        }
        if(n == 0){
            temp = delSpace();
        }
        else{
            temp = getwchar();
        }
        buf[n++] = temp;
        if(buf[0] == L'\n' && buf[1] == L'\n'){
            break;
        }
    } while(temp != L'.');
    buf[n] = L'\0';
    t = realloc(buf, sizeof(wchar_t) * (n + 1));
    if(t != NULL){
        buf = t;
    }
    else{
        wprintf(L"Error. readSentence.");
        return NULL;
    }
    struct Sentence* t_sent = malloc(sizeof(struct Sentence));
    struct Sentence* sentence;
    if(t_sent != NULL){
        sentence = t_sent;
    }
    else{
        free(buf);
        wprintf(L"Error. Structure fail");
        return NULL;
    }

    sentence -> string = buf;
    sentence -> size = size;
    sentence -> len = n + 1; // With /0
    return sentence;
}
```

```
struct Text readText(){
    int size = MEM_STEP;
    struct Sentence** t = malloc(size * sizeof(struct Sentence*));
    struct Text text;
    struct Sentence** sentence;
    if (t != NULL){
        sentence = t;
    }
    else{
        text.len = 0;
        return text;
    }
    int n = 0;
    struct Sentence* temp;
    do{
        temp = readSentence();

        if(n == size){
            size += MEM_STEP;
            t = realloc(sentence, size * sizeof(struct Sentence*));
            if (t != NULL){
                sentence = t;
            }
            else{
                for(int i = 0; i < text.len; i++){
                    free(text.sentences[i] -> string);
                }
                free(text.sentences);
                text.len = 0;
                return text;
            }
        }

        if(temp != NULL){
            sentence[n++] = temp;
        }
        else{
            for(int i = 0; i < text.len; i++){
                    free(text.sentences[i] -> string);
                    free(text.sentences[i]);
                }
                free(text.sentences);
                text.len = 0;
                return text;
```

```
        }
    } while(wcscmp(sentence[n-1] -> string, L"\n\n"));

    text.size = size;
    text.sentences = sentence;
    text.len = n;

    return text;
}

void printText(struct Text text){
    for(int i = 0; i < text.len - 1; i++){
        wprintf(L"%ls ", text.sentences[i] -> string);
    }
    wprintf(L"\n");
}

void freeText(struct Text text){
    for(int i = 0; i < text.len; i++){
        free(text.sentences[i] -> string);
        free(text.sentences[i]);
    }
    free(text.sentences);
}

int deleteSentence(struct Text text){
    struct Sentence** sen = text.sentences;
    int new_len = text.len;
    int i = 1;
    int flag = 0;
    while(sen < text.sentences + new_len - 1){
        while(sen + i < text.sentences + new_len){
            if(!wcscasecmp((*sen) -> string, (*(sen + i)) -> string)){
                free((*(sen + i)) -> string);
                free((*(sen + i)));
                new_len--;
                memmove(sen + i, sen + i + 1, sizeof(struct Sentence*) *
(text.sentences + new_len - sen - i));
                flag = 1;
            }
            else{
                i++;
            }
        }
```

```
        if(flag){
            flag = 0;
            i = 1;
            free((*sen) -> string);
            free((*sen));
            new_len--;
            memmove(sen,   sen   +   1,   sizeof(struct   Sentence*)   *
(text.sentences + new_len - sen));
        }
        else{
            i = 1;
            sen++;
        }
    }
    return new_len;
}

int main(){
    setlocale(LC_ALL, "");
    wprintf(L"Введите текст, который хотите обработать\n");
    struct Text text = readText();
    wchar_t* word;
    text.len = deleteSentence(text);
    int command = 0;
    while(command != 5){
        wprintf(L"Введите команду\n(1) - Поиск анаграм\n(2) - Сортировка
предложений по заглавным буквам\n(3) - Замена гласных букв\n(4) - Замена
слова\n(5) - выход\n");
        wscanf(L"%d", &command);
        switch (command)
        {
            case 1:
                findAnagram(text);
                break;
            case 2:
                sortSentences(text);
                printText(text);
                break;
            case 3:
                addLetters(text);
                printText(text);
                break;
            case 4:
                getwchar();
```

```
                replace(text);
                printText(text);
                break;
            case 5:
                wprintf(L"Завершение работы программы\n");
                break;
            default:
                wprintf(L"Неверная команда\n");
                break;
        }
    }
    printText(text);
    freeText(text);
    return 0;
}
```

Файл findAnagram.c

```
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>
#include <string.h>
#include "structures.h"
#include "findAnagram.h"

int compareLetters(const void* arg1, const void* arg2){
    wchar_t* sign1 = (wchar_t*) arg1;
    wchar_t* sign2 = (wchar_t*) arg2;
    return *sign1 - *sign2;
}

void kickSymbols(struct Sentence* sen){
    int count = 0;
    wchar_t* str = sen -> string;
    for(int i = 0; i < sen -> len; i++){
        if(str[i] != L' ' && str[i] != L'.' && str[i] != L','){
            count++;
        }
    }
    sen -> count = count;
}

wchar_t* copyAndSort(struct Sentence* sen){
    kickSymbols(sen);
```

```
    wchar_t* str = malloc(sizeof(wchar_t) * (sen -> count));
    wchar_t c;
    int j = 0;
    for(int i = 0; i < sen -> len; i++){
        c = sen -> string[i];
        if(c != L'.' && c != L'\'' && c != L',' && c != L' '){
            str[j++] = c;
        }
    }
    qsort(str, sen -> count - 1, sizeof(wchar_t), compareLetters);
    return str;
}

void findAnagram(struct Text text){
    wchar_t* sen_1;
    wchar_t* sen_2;
    for(int i = 0; i < text.len - 2; i++){
        sen_1 = copyAndSort(text.sentences[i]);
        for(int j = i + 1; j < text.len - 1; j++){
            sen_2 = copyAndSort(text.sentences[j]);
            if(wcslen(sen_1) == wcslen(sen_2)){
                if(!(wcsncmp(sen_1, sen_2, wcslen(sen_1)))){
                    wprintf(L"%ls     --     %ls     --     Anagrams\n",
text.sentences[i] -> string, text.sentences[j] -> string);
                }
            }
            free(sen_2);
        }
        free(sen_1);
    }
}
```

Файл addLetters.c

```
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>
#include <wctype.h>
#include <string.h>
#include "structures.h"
#include "sortSentences.h"


void findUpper(struct Sentence* sen){
```

```
    int count = 0;
    wchar_t* str = sen -> string;
    for(int i = 0; i < sen -> len - 1; i++){
        if(iswupper(str[i])){
            count++;
        }
    }
    sen -> count_upper_symbol = count;
}

int compareSentences(const void* arg1, const void* arg2){
    struct Sentence** sen1 = (struct Sentence**) arg1;
    struct Sentence** sen2 = (struct Sentence**) arg2;
    return    ((*sen1)    ->    count_upper_symbol    -    (*sen2)    ->
count_upper_symbol);
}

void sortSentences(struct Text text){
    for (int i = 0; i < text.len - 1; i++){
        findUpper(text.sentences[i]);
    }
    qsort(text.sentences,   text.len   -   1,   sizeof(struct   Sentence*),
compareSentences);
}
```

Файл replace.c

```
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>
#include <string.h>
#include "replace.h"
#include "structures.h"

#define MEM_STEP 20;

wchar_t* readWord(){
    wprintf(L"Введите слова\n");
    int size = MEM_STEP;
    wchar_t* t = calloc(size, sizeof (wchar_t));
    wchar_t* word;
    wchar_t c;
    int i = 0;
    if(t != NULL){
```

```
        word = t;
    }
    do{
        if(i >= size - 2){
            size += MEM_STEP;
            t = realloc(word, sizeof(wchar_t) * size);
            if (t != NULL){
                word = t;
            }
            else{
                wprintf(L"Oops\n");
                return NULL;
            }
        }
        c = getwchar();
        word[i++] = c;
    } while (c != L'\n');
    word[i-1] = '\0';
    t = realloc(word, sizeof (wchar_t) * i);
    if (t != NULL){
        word = t;
    }
    else{
        wprintf(L"Oops\n");
        return NULL;
    }
    return word;
}

void replace(struct Text text){
    wchar_t* word_1 = readWord();
    wchar_t* word_2 = readWord();
    int diff = wcslen(word_2) - wcslen(word_1);
    for(int i = 0; i < text.len - 1; i++) {
        wchar_t      *sent_copy      =      calloc(text.sentences[i]->len,
sizeof(wchar_t));
        wcscpy(sent_copy, text.sentences[i]->string);
        wchar_t *pt;
        wchar_t *word = wcstok(sent_copy, L",. ", &pt);
        wchar_t *sent_out;
        size_t index;
        while (word != NULL) {
            if (wcscmp(word, word_1) == 0) {
                index = word - sent_copy;
```

```
                if (diff > 0) {
                    sent_out = calloc(wcslen(text.sentences[i]->string) +
diff + 1, sizeof(wchar_t));
                } else {
                    sent_out = calloc(wcslen(text.sentences[i]->string) +
1, sizeof(wchar_t));
                }
                wcscpy(sent_out, text.sentences[i]->string);
                wmemmove(sent_out + index + wcslen(word_2), sent_out +
index + wcslen(word_1),
                         wcslen(sent_out) - index - wcslen(word_1) + 1);
                wcsncpy(sent_out + index, word_2, wcslen(word_2));
                free(text.sentences[i]->string);
                text.sentences[i]->string = sent_out;
                text.sentences[i]->len = (int) wcslen(sent_out) + 1;
                sent_copy = realloc(sent_copy, text.sentences[i]->len *
sizeof(wchar_t));
                wcscpy(sent_copy, text.sentences[i]->string);
                word = wcstok(sent_copy, L",. ", &pt);
            }
            word = wcstok(NULL, L",. ", &pt);
        }
    }
    free(word_1);
    free(word_2);
}
```

Файл Makefile

```
all: main.o findAnagram.o addLetters.o replace.o sortSentences.o
     gcc main.o findAnagram.o addLetters.o replace.o sortSentences.o -o
program

main.o: main.c
     gcc -c main.c

findAnagram.o: findAnagram.c
     gcc -c findAnagram.c

addLetters.o: addLetters.c
     gcc -c addLetters.c

replace.o: replace.c
     gcc -c replace.c
```

```
sortSentences.o: sortSentences.c
	gcc -c sortSentences.c
```

Заголовочные Файлы

replace.h

```
#include "structures.h"
#pragma once
void replace(struct Text text);
```

addLetters.h

```
#include "structures.h"
#pragma once
void addLetters(struct Text text);
```

findAnagram.h

```
#include "structures.h"
#pragma once
void findAnagram(struct Text text);
```

sortSentences.h

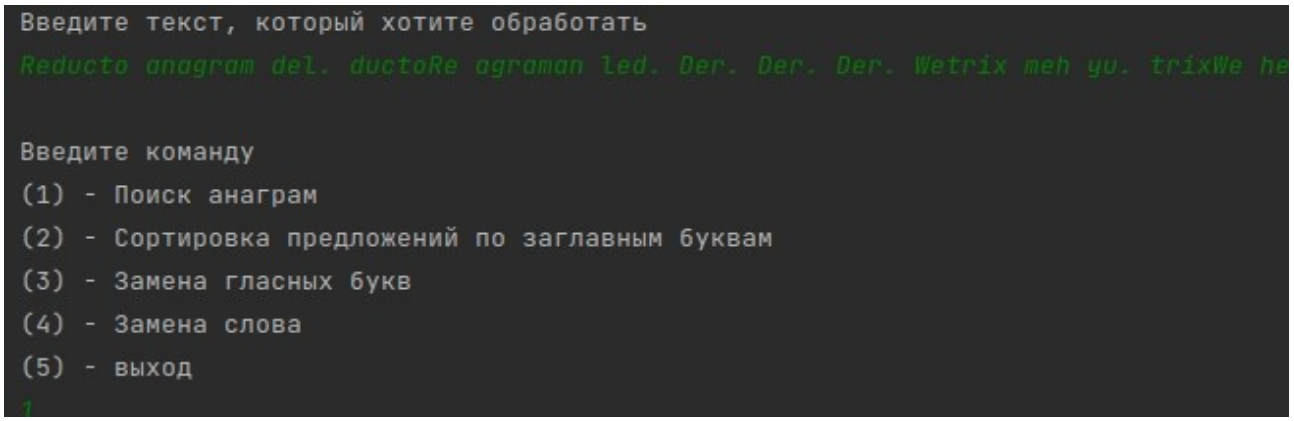
```
#include "structures.h"
#pragma once
void sortSentences(struct Text text);
```

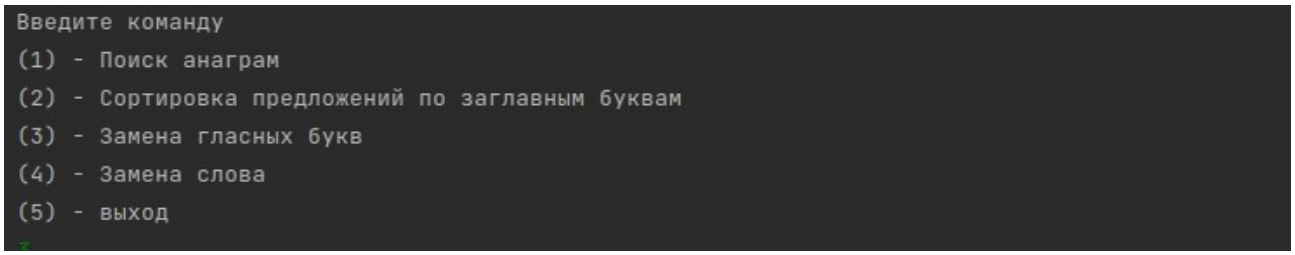
structures.h

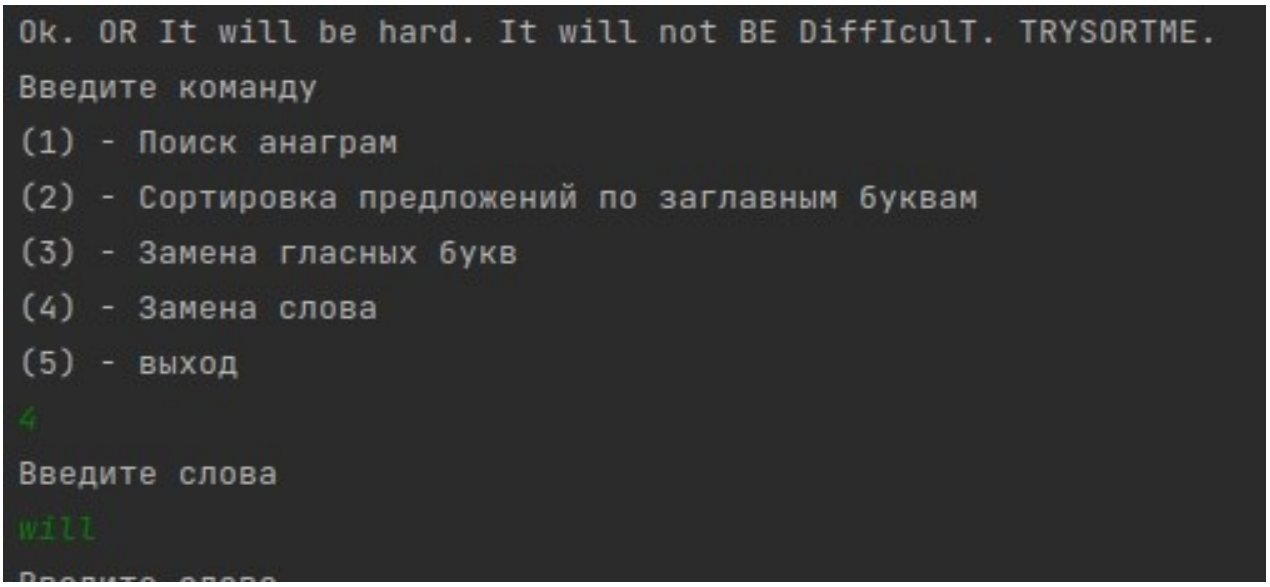
```
#include <wchar.h>
#pragma once
struct Sentence
{
    wchar_t* string;
    int size, len, count, count_upper_symbol;
};

struct Text
{
    struct Sentence** sentences;
    int size;
    int len;
};
```

ПРИЛОЖЕНИЕ Б

ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ







ПРИЛОЖЕНИЕ В
ПРИМЕРЫ ОБРАБОТКИ ОШИБОК

```
Введите команду
(1) - Поиск анаграм
(2) - Сортировка предложений по заглавным буквам
(3) - Замена гласных букв
(4) - Замена слова
(5) - выход
6
Неверная команда
Введите команду
(1) - Поиск анаграм
(2) - Сортировка предложений по заглавным буквам
```