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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: обработка строк на языке Си

Студент гр. 0382 _____ Крючков А.М.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Крючков Артем

Группа 0382

Тема работы: обработка строк на языке Си

Исходные данные:

Вариант 9

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.

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).

Далее, программа должна запрашивать у пользователя одно из следующих доступных действий (программа должна печатать для этого подсказку. Также следует предусмотреть возможность выхода из программы):

1. Изменить все слова в тексте заканчивающиеся на “ться” так, чтобы они заканчивались на “тся” и наоборот.
2. Вывести все предложения в которых встречается второе слово первого предложения. Данное слово необходимо выделить зеленым цветом.

3. Отсортировать предложения по возрастанию количества слов в предложении.
4. Удалить все предложения в которых больше 10 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:
разделы «Аннотация», «Содержание», «Введение» («Цель и задачи»), «Выполнение работы». «Примеры работы программы», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 02.11.2020

Дата сдачи реферата: 20.12.2020

Дата защиты реферата: 26.12.2020

Студент _____ Крючков А.М.

Преподаватель _____ Жангиров Т.Р.

АННОТАЦИЯ

В процессе выполнения курсовой работы была реализована программа для обработки текст на языке Си. Для хранения текста использовались структуры. Обработка и вывод текста производились при помощи использования функций следующих стандартных заголовочных файлов: stdio.h, stdlib.h, wchar.h, locale.h. Для удобства пользователя реализован вывод на консоль контекстного меню выбора необходимой опции обработки текста. При некорректном вводе номера опции выполнение программы завершается. Также был написан makefile для удобной сборки программы.

SUMMARY

In the course of the course work, a program for processing text in the C language was implemented. Structures were used to store text. Processing and output of the text was carried out using the functions of the following standard header files: stdio.h, stdlib.h, wchar.h, locale.h. For the convenience of the user, the output to the console of the context menu for selecting the necessary text processing option is implemented. If the option number is entered incorrectly, the program is terminated. Also, a makefile was written for easy building of the program.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной курсовой работе производится разработка стабильной консольной программы, производящей обработку введённого пользователем текста в соответствии с выбранной пользователем опцией обработки введённого текста.

Программа реализована при помощи использования структур (они используются для хранения данных о введённом тексте). Память под текст в программе выделяется динамически при помощи использования стандартных функций выделения памяти из стандартного заголовочного файла stdlib.h. Также разработанная программа имеет возможность обработки кириллических символов. Это возможность существует благодаря использованию функций из стандартных заголовочных файлов wchar.h, locale.h, предназначенных для работы с широкими символами. Сборка программы реализуется при помощи утилиты Make и написанного файла Makefile.

Программа разработана для операционных систем на базе Linux. Разработка велась на операционной системе Ubuntu 20.04 при помощи интерактивной среды разработки CLion и текстового редактора Vim. Отладка программы производилась средствами интерактивной среды разработки и утилиты Valgrind.

1. ЦЕЛИ И ЗАДАЧИ

Цель: разработка стабильной программы, способной выполнять считывание текста и его обработку в соответствии с заданием.

Задачи:

- Реализовать считывание текста с консоли.
- Реализовать функции обработки текста в соответствии с заданием.
- Разбить функции на файлы, для работы которых использовать Makefile
- произвести отладку программы

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Структуры для считывания текста

Структуры *Sentence* и *Text*.

Поля *Sentence*:

- *wchar_t *words* — динамический массив символов
- *int size* — размер динамического массива символов
- *int n* – количество символов.

Поля *Text*:

- *struct Sentence **sentences* — ссылка на динамический массив предложений.
- *int size* — размер sentences
- *int n* — количество предложений

2.2. Функции для считывания текста

Функции struct Sentence *readSentence() и struct Text readText().

Функция struct Text readText() записывает введённые пользователем символы, разделяя их на предложения(разделитель «.»), используя функцию readText(). Обе функции используют динамическое выделение памяти. Ввод заканчивается символом перевода строки. При вводе удаляется все разделители слов, стоящие в начале предложения.

Переменные struct Text readText():

- int size — текущий размер динамического массива предложений.
- int n – количество предложений
- struct Sentence **sentences – ссылка на динамический массив предложений.
- struct Sentence* sentence – предложение полученное выполнением функции struct Sentence *readSentence().
- struct Text text — возвращаемое значение.

Функция struct Sentence* readSentence() считывает символы с консоли, пока не встретит символ «.» или символ перевода строки «\n». Функция wcschr(L".\n", wc) возвращает NULL, если символа wc нет в строке L".\n". Ввод происходит с консоли при помощи функции fgetwc(stdin).

Переменные struct Text readText():

- int size — текущий размер динамического массива предложений.
- int n – количество символов.
- struct Sentence *words – динамический массив символов.
- wchar_t wc — вводимый символ.
- struct Sentence *sentence— возвращаемое значение.

2.3. Функции обработки текста

struct Text deleteSentence(struct Text text, int indexToDelete) — удаляет сообщение из struct Text text с индексом indexToDelete. Освобождает память выделенную из-под удаленного предложения при помощи функции free().

struct Text deleteRepeatedSentences(struct Text text) – функция удаляет повторяющиеся предложения без учета регистра. Сравнение предложение происходит посредством функции wcscasecmp(). Удаление посредством функции deleteSentence(). Оставляется первое из одинаковых предложений

struct Text deleteBigSentences(struct Text text) — удаляет предложения в которых больше 10 слов. Количество слов определяется при помощи функции countWords(*s). Удаление посредством функции deleteSentence().

struct Text changeEnding(struct Text text) — функция меняет окончания всех слов с «тся» на «ться» и наоборот. Для сравнения окончаний слов используется функция wcsncmp(s1, s2, n), которая сравнивает s1 и s2 вплоть до n символа. Для проверки, что после идёт конец слова, используется функция wcschr(). Для того, чтобы изменить окончания в предложении каждый раз создаётся новый динамический массив символов(newword).

int countWords(struct Sentence s) — функция считает количество слов в предложении s.

int compareSentencesByWords(const void *ss1, const void *ss2) — функция сравнивает предложения по количеству слов в них. Количество слов вычисляется посредством функции countWords().

2.4. Функции вывода текста

void printText(struct Text text) — функция выводит в консоль предложений из text. Каждое предложения отделяется символом перевода строки.

void printSentsWithSecondWordFirstSent(struct Text text) — функция выводит все предложения, в которых встречается второе слово первого предложения. Сначала происходит поиск второго слово первого предложения, это слово записывается в динамический массив символов word, поиск осуществляется с помощью функции wcsstr(). Затем проход по всем предложениям в тексте. Если в предложении найдено word, то тогда это предложение выводится, а слово word выделяется зелёным цветом.

2.5. Функция main

Для корректной работы с кириллическими символами в функции main используется функция setlocale.

Сначала выводится подсказка о начале работы программы с предложением ввести текст. Ввод текста реализован с помощью функции readText(). Затем из текста удаляются повторяющиеся предложения с помощью функции deleteRepeatedSentences(). Затем печатается подсказка о возможнстях программы. Далее программы выполняет работу с текстом, пока пользователь не введёт несуществующую опцию. При вводе несуществующей опции, программа печатает сообщение о завершении программы и очищает выделенную динамическую память.

Доступные действия:

1. Изменить все слова в тексте заканчивающиеся на “ться” так чтобы они заканчивались на “тся” и наоборот. Реализовано с помощью функции changeEnding().

2. Вывести все предложения в которых встречается второе слово первого предложения. Данное слово необходимо выделить зеленым цветом. Реализовано с помощью функции printSentsWithSecondWordFirstSent().
3. Отсортировать предложения по возрастанию количества слов в предложении. Реализовано с помощью функции qsort(), которая в качестве компаратора принимает функцию compareSentencesByWords.
4. Удалить все предложения в которых больше 10 слов. Реализовано с помощью функции deleteBigSentences().

Для удобства, работа каждой опции завершается выводом текста с помощью функции printText().

2.6. Разделение программы на файлы и создание Makefile

Вся программа была разделена на следующий файлы:

- main.c - основной файл программы, в нём содержится функция main. С помощью #inlclude в него включены все остальные заголовочные файлы;
- textStructures.h — файл, содержащий объявления структур;
- input.h - файл, содержащий объявления функций считывания readSentence и readText, инструкции для препроцессора, также в него включён файл textStructures.h;
- stringsOperations.h — файл, содержащий объявления функций countWords, deleteSentence, deleteRepeatedSentences, changeEnding, compareSentencesByWords и deleteBigSentences, инструкции для препроцессора, также в него включён файл textStructures.h;
- output.h — файл, содержащий объявления функций printSentsWithSecondWordFirstSent и printText, инструкции для препроцессора, также в него включены файлы textStructures.h, stringsOperations.h и input.h;
- input.c – файл, содержащий реализацию функций считывания readSentence и readText

- stringsOperations.c — файл, содержащий реализацию функций countWords, deleteSentence, deleteRepeatedSentences, changeEnding, compareSentencesByWords и deleteBigSentences;
- output.c — файл, содержащий реализацию функций printSentsWithSecondWordFirstSent и printText;

  В Makefile прописаны все цели для создания нужных объектных и исполняемогофайлов, а также все зависимости.

ЗАКЛЮЧЕНИЕ

В результате данной курсовой работы были изучены основные принципы обработки строк на языке Си. Получены навыки использования структур, работы с динамической памятью и работы с функциями заголовочных файлов wchar.h и wctype.h.

В ходе разработки программы была достигнута цель: разработана стабильная программа, способная запрашивать у пользователя нужную опцию обработки текста и выводить изменённый текст (исходный код программы см. в приложении Б).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ПРОГРАММИРОВАНИЕ Учебно-методическое пособие / сост: К. В. Кринкин, Т. А. Берленко, М. М. Заславский, К. В. Чайка.: СПбГЭТУ «ЛЭТИ», 2018. 34с.

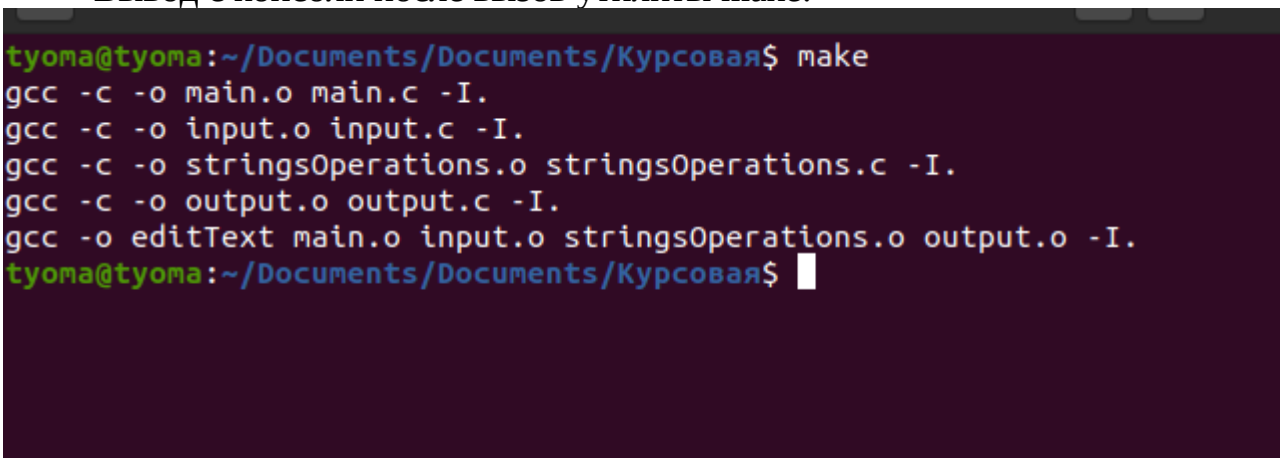
2. Керниган Б. и Ритчи Д. Язык программирования Си. М.: Вильямс, 1978 288 с.

3. Cplusplus URL: http://cplusplus.com (дата обращения: 14.12.2020).

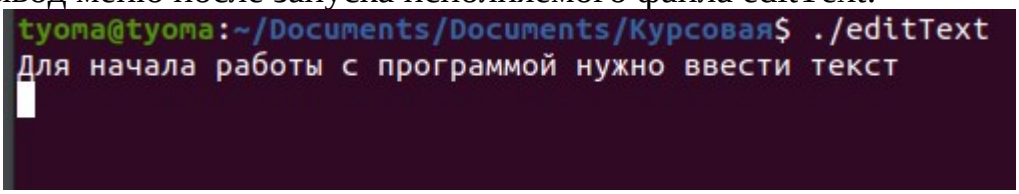
ПРИЛОЖЕНИЕ А

ПРИМЕР РАБОТЫ ПРОГРАММЫ

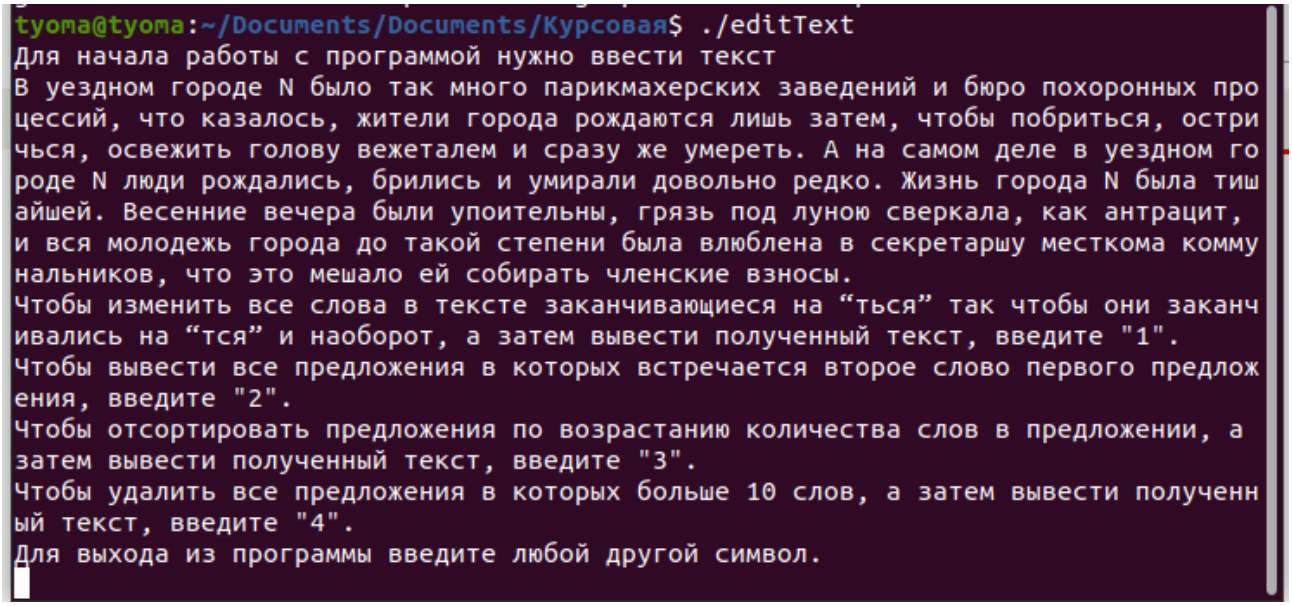
Вывод с консоли после вызов утилиты make:



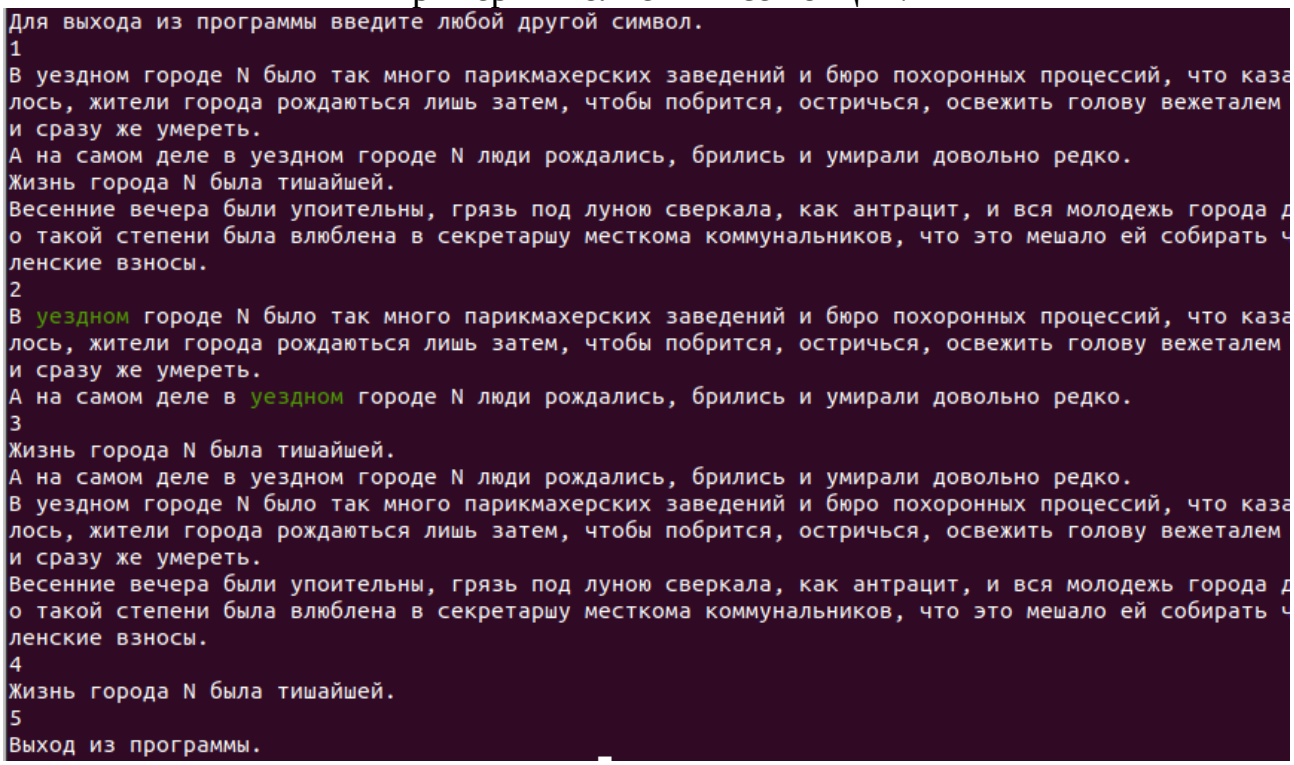
Вывод меню после запуска исполняемого файла editText:



После ввода текста.



Пример выполнения всех опций.



ПРИЛОЖЕНИЕ А

ПРИМЕР РАБОТЫ ПРОГРАММЫ

Файл main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>

#include "textStructures.h"
#include "stringsOperations.h"
#include "input.h"
#include "output.h"

int main() {
    setlocale(LC_ALL, "");
    wprintf(L"Для начала работы с программой нужно ввести текст\n");
    struct Text text = readText();
    text = deleteRepeatedSentences(text);
    wchar_t key = L'1';
    wprintf(L"Чтобы изменить все слова в тексте заканчивающиеся на “ться” 
так чтобы они заканчивались на “тся” и наоборот, а затем вывести 
полученный текст, введите \"1\".\n"
            "Чтобы вывести все предложения в которых встречается второе 
слово первого предложения, введите \"2\".\n"
            "Чтобы отсортировать предложения по возрастанию количества 
слов в предложении, а затем вывести полученный текст, введите \"3\".\n"
            "Чтобы удалить все предложения в которых больше 10 слов, а 
затем вывести полученный текст, введите \"4\".\n"
            "Для выхода из программы введите любой другой символ.\n"
    );
    while (wcschr(L"1234", key) != NULL) {
        key = fgetwc(stdin);
        fgetwc(stdin);
        switch (key) {
            case L'1':
                text = changeEnding(text);
                printText(text);
                break;
            case L'2':
                printSentsWithSecondWordFirstSent(text);
                break;
            case L'3':
                qsort(text.sentences, text.n, sizeof(struct Sentence *), 
compareSentencesByWords);
                printText(text);
                break;
            case L'4':
                text = deleteBigSentences(text);
                printText(text);
                break;
            default:
                wprintf(L"Выход из программы.\n");
                break;
        }
    }
```

```
    for (int i = 0; i < text.n; ++i) {
        free(text.sentences[i]->words);
        free(text.sentences[i]);
    }
    free(text.sentences);
    return 0;
}
```

Файл stringsOperations.c:

```
#include "stringsOperations.h"
struct Text deleteSentence(struct Text text, int indexToDelete) {

    free(text.sentences[indexToDelete]->words);
    free(text.sentences[indexToDelete]);
    for (int i = indexToDelete; i < text.n; i++) {
        if (i != indexToDelete) {
            text.sentences[i - 1] = text.sentences[i];
        }
    }
    text.n = text.n - 1;
    return text;
}

struct Text deleteRepeatedSentences(struct Text text) {
    int ntext = text.n;
    for (int i = 0; i < ntext; i++) {
        for (int j = i + 1; j < ntext; j++) {
            if ((wcscasecmp(text.sentences[i]->words, text.sentences[j]-
>words) == 0)) {
                text = deleteSentence(text, j);
                j--;
                ntext--;
            }
        }
    }
    text.n = ntext;
    return text;
}

struct Text deleteBigSentences(struct Text text) {
    struct Sentence *s;
    for (int i = 0; (int) i < text.n; ++i) {
        s = text.sentences[i];
        if (countWords(*s) > 10) {
            text = deleteSentence(text, i);
            i--;
        }
    }
    return text;
}

struct Text changeEnding(struct Text text) {
    struct Sentence *s;
    wchar_t *newword;
    int newsize;
    int newn;
    for (int i = 0; i < text.n; ++i) {
        s = text.sentences[i];
```

```
        newsize = s->size;
        newword = malloc(newsize * sizeof(wchar_t));
        newn = 0;
        for (int j = 0; j < (s->n + 3); ++j) {
            if (newn == newsize - 2) {
                newsize += INIT;
                newword = realloc(newword, newsize * sizeof(wchar_t));
            }
            if (j < (s->n - 2)) {
                if ((wcsncmp(L"тся", (*s).words + j, 3) == 0) &&
wcschr(L" .,\n", ((*s).words[j + 3])) != NULL) { //тся
                    newword[newn++] = s->words[j];
                    newword[newn++] = L'ь';
                } else if (j < s->n - 3) {//ться
                    if ((wcsncmp(L"ться", (*s).words + j, 4) == 0) &&
                        wcschr(L" .,\n", ((*s).words[j + 4])) != NULL)
{ //тся
                        newword[newn++] = s->words[j];
                        newword[newn++] = L'с';
                        newword[newn++] = L'я';
                        j += 3;
                    } else {
                        newword[newn++] = s->words[j];
                    }
                } else {
                    newword[newn++] = s->words[j];
                }
            } else {
                newword[newn++] = s->words[j];
            }
            if (s->words[j] == L'\0') {
                break;
            }
        }
        free(text.sentences[i]->words);
        text.sentences[i]->words = newword;
        text.sentences[i]->n = newn - 1;
        text.sentences[i]->size = newsize;
    }
    return text;
}


int countWords(struct Sentence s) {
    int res = 0;
    int k = 0;
    for (int i = 0; i < s.n - 1; ++i) {
        if ((s.words[i] == ' ' && k == 0) ||
            (s.words[i] == ',' && k == 0)) { // k необходимо для проверки
на множественный ввод пробелов или запятых
            res++;
            k = 1;
        } else if (s.words[i] != ' ' && s.words[i] != ',' && s.words[i] !
= '\n') {
            k = 0;
        }
    }
```

```
    if (res > 0 || (res == 0 && s.n > 1)) res++;
    return res;
}

int compareSentencesByWords(const void *ss1, const void *ss2) {
    struct Sentence *s1 = *(struct Sentence **) ss1;
    struct Sentence *s2 = *(struct Sentence **) ss2;
    int ns1 = countWords(*s1);
    int ns2 = countWords(*s2);
    return ns1 - ns2;
}
```

Файл input.c:

```
#include "input.h"
struct Sentence *readSentence() {
    int size = INIT;
    int n = 0;
    wchar_t *words = malloc(size * sizeof(wchar_t));
    wchar_t wc;
    do {
        wc = fgetwc(stdin);
        if (wcschr(L" ", wc) && n == 0) {
            continue;
        }

        words[n] = wc;
        n++;
        if (n == size - 2) {
            size += INIT;
            words = realloc(words, size * sizeof(wchar_t));
        }
        if (wcschr(L"\n", wc) && n != 0) {
            break;
        }
    } while (!wcschr(L".\n", wc));
    words[n] = L'\0';
    struct Sentence *sentence = malloc(size * sizeof(struct Sentence));
    sentence->words = words;
    sentence->size = size;
    sentence->n = n;
    return sentence;
}

struct Text readText() {
    int size = INIT;
    int n = 0;
    struct Sentence **sentences = malloc(size * sizeof(struct Sentence
*));
    struct Sentence *sentence;
    while (1) {
        sentence = readSentence();
        if (sentence->words[sentence->n - 1] == L'\n') {
            if (sentence->n == 1) {
                free(sentence->words);
                free(sentence);
            } else {
                sentence->words[sentence->n-1]=L'\0';
```

```
                sentences[n] = sentence;
                n++;
            }
            break;
        } else {
            sentences[n] = sentence;
            n++;
            if (n == size - 2) {
                size += INIT;
                sentences = realloc(sentences, size * sizeof(struct
Sentence *));
            }
        }
    }
    struct Text text;
    text.sentences = sentences;
    text.size = size;
    text.n = n;
    return text;
}
```

Файл input.c:

```
#include "input.h"
struct Sentence *readSentence() {
    int size = INIT;
    int n = 0;
    wchar_t *words = malloc(size * sizeof(wchar_t));
    wchar_t wc;
    do {
        wc = fgetwc(stdin);
        if (wcschr(L" ", wc) && n == 0) {
            continue;
        }

        words[n] = wc;
        n++;
        if (n == size - 2) {
            size += INIT;
            words = realloc(words, size * sizeof(wchar_t));
        }
        if (wcschr(L"\n", wc) && n != 0) {
            break;
        }
    } while (!wcschr(L".\n", wc));
    words[n] = L'\0';
    struct Sentence *sentence = malloc(size * sizeof(struct Sentence));
    sentence->words = words;
    sentence->size = size;
    sentence->n = n;
    return sentence;
}

struct Text readText() {
    int size = INIT;
    int n = 0;
```

```
    struct Sentence **sentences = malloc(size * sizeof(struct Sentence
*));
    struct Sentence *sentence;
    while (1) {
        sentence = readSentence();
        if (sentence->words[sentence->n - 1] == L'\n') {
            if (sentence->n == 1) {
                free(sentence->words);
                free(sentence);
            } else {
                sentence->words[sentence->n-1]=L'\0';
                sentences[n] = sentence;
                n++;
            }
            break;
        } else {
            sentences[n] = sentence;
            n++;
            if (n == size - 2) {
                size += INIT;
                sentences = realloc(sentences, size * sizeof(struct
Sentence *));
            }
        }
    }
    struct Text text;
    text.sentences = sentences;
    text.size = size;
    text.n = n;
    return text;
}
```

Файл output.c:

```
#include "output.h"

void printText(struct Text text) {
    struct Sentence *s;
    struct Sentence **sen = text.sentences;
    for (unsigned int i = 0; i < text.n; i++) {
        s = sen[i];
        wprintf(L"%ls\n", s->words);
    }
}

void printSentsWithSecondWordFirstSent(struct Text text) {
    if (text.n == 0) return;
    struct Sentence *firstSent = text.sentences[0];
    if (firstSent->n == 1) return;
    int size = INIT;
    int n = 0, i = 0;
    wchar_t *word = malloc(size * sizeof(wchar_t));
    wchar_t wc;
    int found = 0;
    while (1) {
        wc = firstSent->words[n];
        n++;
        if (wcschr(L".", wc) && n != 0) {
            break;
```

```
        }
        if ((wcschr(L" .,", wc) && n != 0)) {
            if (found)break;
            found = 1;
            continue;
        }
        if (found) {
            word[i++] = wc;
            if (i == size - 2) {
                size += INIT;
                word = realloc(word, size * sizeof(wchar_t));
            }
        }
    };
    word[i] = L'\0';
    struct Sentence *s;
    int isIntSentOk;
    if (i == 0) return; // если только одно слово в первом предложении
    for (int j = 0; j < text.n; ++j) {
        s = text.sentences[j];
        isIntSentOk = wcsstr(s->words, word) != NULL;
        if (isIntSentOk) {
            for (int k = 0; k < s->n; ++k) {
                if (wcsncmp(word, (*s).words + k, i) == 0 &&
                    (wcschr(L" .,", *((*s).words + k + i)) != 0) &&
                    (k == 0 || (wcschr(L" .,", *((*s).words + k - 1)) !=
0))) {
                    wprintf(L"\033[32m\%ls\033[0m", word);
                    k += i - 1;
                } else {
                    wc = *((*s).words + k);
                    wprintf(L"%lc", wc);
                }
            }
            wprintf(L"\n");
        }
    }
    free(word);
}
```

Файл stringOperations.h:

```
#pragma once

#include <locale.h>
#include <wchar.h>
#include <stdlib.h>
#include "input.h"

#include "textStructures.h"

int countWords(struct Sentence s);

struct Text deleteSentence(struct Text text, int indexToDelete);

struct Text deleteRepeatedSentences(struct Text text);
```

```
struct Text deleteBigSentences(struct Text text);

struct Text changeEnding(struct Text text);

int compareSentencesByWords(const void *ss1, const void *ss2);
```

Файл textStructures.h:

```
#pragma once
struct Sentence {
    wchar_t *words;
    int size;
    int n;
};
struct Text {
    struct Sentence **sentences;
    int size;
    int n;
};
```

Файл input.h:

```
#pragma once

#include <locale.h>
#include <wchar.h>
#include <stdlib.h>
#include <stdio.h>

#include "textStructures.h"

#define INIT 20 // дополнительный размер выделяемый при заполнении

struct Sentence *readSentence();

struct Text readText();
```

Файл output.h:

```
#pragma once

#include <locale.h>
#include <wchar.h>
#include <stdlib.h>
#include <stdio.h>
#include "input.h"
#include "textStructures.h"
#include "stringsOperations.h"

void printSentsWithSecondWordFirstSent(struct Text text);

void printText(struct Text text);
```

Makefile:

```
CC=gcc
CFLAGS= -I.
DEPS = inout.h stringsOperations.h textStructures.h
OBJ = main.o inout.o stringsOperations.o

%.o: %.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)
```

```
editText: $(OBJ)
	$(CC) -o $@ $^ $(CFLAGS)


.PHONY: clean

clean:
	rm -f $(OBJ)
```