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 _____ Шаврин А.П.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2021

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Шаврин А.П.

Группа 1304

Тема работы: обработка текстовых данных

Исходные данные:
С помощью структур и широких символов написать приложение на Си по обработке введённого пользователем текста, согласно выбранным пользователем действиям. Пока пользователь сам не решит остановить программу, ждать команды пользователя.
Содержание пояснительной записки:
«Содержание», «Введение», «Разработка кода», «Сборка», «Тестирование», «Заключение», «Список использованных источников», «Приложение А. Код программы»

Предполагаемый объем пояснительной записки:
Не менее 12 страниц.

Дата выдачи задания: 15.10.2021

Дата сдачи реферата: 20.12.2021

Дата защиты реферата: 23.12.2021

Студент _____ Шаврин А.П.

Преподаватель _____ Чайка К.В.

АННОТАЦИЯ

Разработано приложение на Си, демонстрирующее пользователю возможности обработки текста. Хранение текста организовано в виде структур в динамически выделяемой памяти. Вводимый текст поддерживает кириллические и латинские символы. В работе используются функции стандартной библиотеки языка Си.

SUMMARY

A c application has been developed that demonstrates the text processing capabilities to the user. Storing text is organized as structures in dynamically allocated memory. The entered text supports cyrillic and latin characters. The work uses the functions of the standard library of the c language.

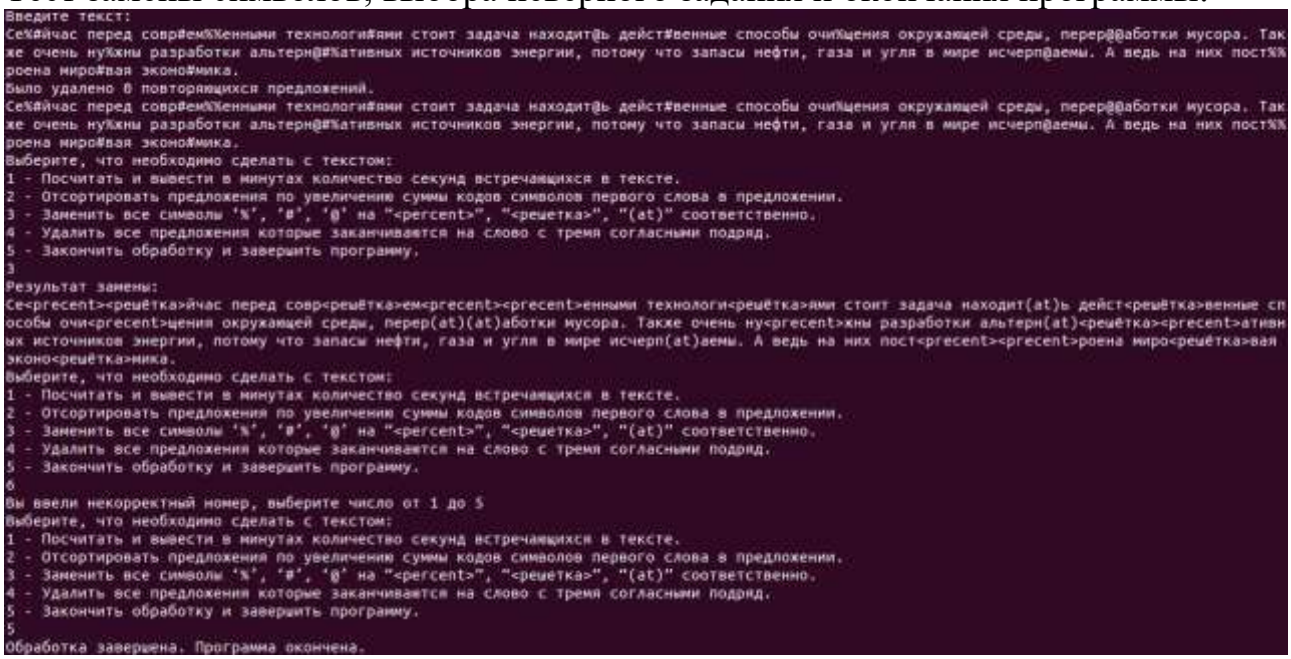
Тест удаления повторов.

```
alex@alex-VirtualBox:~/Shavrin_Alexey_cw/src$ ./main
Введите текст:
Современный мир полон всяческих новинок: науки, техники, медицины. Никогда, наверное, люди не обладали такой суммой знаний
в разных областях. Удивительные вещи и явления окружают нас. Сегодня, образно выражаясь, можно достать из кармана новости,
подключиться к сети, существующей только в информационном поле, передать сообщение человеку, который за тысячи километров о
т тебя. Удивительные вещи и явления окружают нас. Удивительные вещи и явления окружают нас. Благодаря информационным технол
огиям мы можем знакомиться с людьми, что живут далеко-далеко, совсем в других странах и городах. Современные технологии при
званы служить человечеству, помогать ему достигать еще больших высот. Удивительные вещи и явления окружают нас.
Было удалено 3 повторяющихся предложений.
Современный мир полон всяческих новинок: науки, техники, медицины. Никогда, наверное, люди не обладали такой суммой знаний
в разных областях. Удивительные вещи и явления окружают нас. Сегодня, образно выражаясь, можно достать из кармана новости,
подключиться к сети, существующей только в информационном поле, передать сообщение человеку, который за тысячи километров о
т тебя. Благодаря информационным технологиям мы можем знакомиться с людьми, что живут далеко-далеко, совсем в других страна
х и городах. Современные технологии призваны служить человечеству, помогать ему достигать еще больших высот.
Выберите, что необходимо сделать с текстом:
1 - Посчитать и вывести в минутах количество секунд встречающихся в тексте.
2 - Отсортировать предложения по увеличению суммы кодов символов первого слова в предложении.
3 - Заменить все символы %, #, @ на <percent>, <решетка>, (at) соответственно.
4 - Удалить все предложения которые заканчиваются на слово с тремя согласными подряд.
5 - Закончить обработку и завершить программу.
```

Тест подсчета минут и сортировки предложений.

```
Введите текст:
0 sec Однако, не все технологии идут на пользу человеку. two sec Часто происходит наоборот. Например, современное оружие сп
особно уничтожать и природу, и человеческие жизни в пугающих масштабах. Промышленность дымит, коптит, сливает ядовитые отхо
ды в воду, вторгается в земные недра в поисках ресурсов. ten sec 15 sec А пластиковые отходы уже стали притчей во языцех, о
ни не разлагаются природным путем и откладываются в природе надолго. В Тихом океане уже расположился новый «континент» плав
ающего мусора. 34 sec
Было удалено 0 повторяющихся предложений.
0 sec Однако, не все технологии идут на пользу человеку. two sec Часто происходит наоборот. Например, современное оружие сп
особно уничтожать и природу, и человеческие жизни в пугающих масштабах. Промышленность дымит, коптит, сливает ядовитые отхо
ды в воду, вторгается в земные недра в поисках ресурсов. ten sec 15 sec А пластиковые отходы уже стали притчей во языцех, о
ни не разлагаются природным путем и откладываются в природе надолго. В Тихом океане уже расположился новый «континент» плав
ающего мусора. 34 sec.
Выберите, что необходимо сделать с текстом:
1 - Посчитать и вывести в минутах количество секунд встречающихся в тексте.
2 - Отсортировать предложения по увеличению суммы кодов символов первого слова в предложении.
3 - Заменить все символы %, #, @ на <percent>, <решетка>, (at) соответственно.
4 - Удалить все предложения которые заканчиваются на слово с тремя согласными подряд.
5 - Закончить обработку и завершить программу.
1
Количество секунд встречающихся в тексте равно 0,816667 минут.
Выберите, что необходимо сделать с текстом:
1 - Посчитать и вывести в минутах количество секунд встречающихся в тексте.
2 - Отсортировать предложения по увеличению суммы кодов символов первого слова в предложении.
3 - Заменить все символы %, #, @ на <percent>, <решетка>, (at) соответственно.
4 - Удалить все предложения которые заканчиваются на слово с тремя согласными подряд.
5 - Закончить обработку и завершить программу.
2
Результат сортировки:
0 sec Однако, не все технологии идут на пользу человеку. 34 sec. ten sec 15 sec А пластиковые отходы уже стали притчей во я
зыцех, они не разлагаются природным путем и откладываются в природе надолго. two sec Часто происходит наоборот. В Тихом оке
ане уже расположился новый «континент» плавающего мусора. Например, современное оружие способно уничтожать и природу, и чел
овеческие жизни в пугающих масштабах. Промышленность дымит, коптит, сливает ядовитые отходы в воду, вторгается в земные нед
ра в поисках ресурсов.
Выберите, что необходимо сделать с текстом:
1 - Посчитать и вывести в минутах количество секунд встречающихся в тексте.
2 - Отсортировать предложения по увеличению суммы кодов символов первого слова в предложении.
3 - Заменить все символы %, #, @ на <percent>, <решетка>, (at) соответственно.
4 - Удалить все предложения которые заканчиваются на слово с тремя согласными подряд.
5 - Закончить обработку и завершить программу.
```

Тест замены символов, выбора неверного задания и окончания программы.



СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является изучение встроенных средств языка Си
для работы с текстом и разработка приложения с использованием этих средств.
Дли достижения поставленной цели требуется решить следующие подзадачи:

- изучение теоретических материалов (структуры, функции стандартной библиотеки, цветной вывод текста, широкие символы);
- разработка программного кода в соответствии с тех. заданием;
- сборка программы с использованием заголовочных файлов и мэйкфайла;
- тестирование приложения и отладка;
- создание пользовательской инструкции.

1. РАЗРАБОТКА КОДА

1.1. Техническое задание

Вариант 14

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) Посчитать и вывести в минутах количество секунд встречающихся в тексте. Количество секунд задается в виде подстроки “ <число> sec “.
2) Отсортировать предложения по увеличению сумме кодов символов первого слова в предложении..
3) Заменить все символы ‘%’, ‘#’, ‘@’ на “<percent>”, “<решетка>”, “(at)” соответственно.
4) Удалить все предложения которые заканчиваются на слово с тремя согласными подряд.

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.

1.2. Разбиение на подзадачи

Таким образом, код можно разбить на следующие подзадачи:

- Реализовать считывание текста и организовать его удобное расположение в памяти;
- Реализовать функцию печати сохранённого текста.
- Реализовать сравнение предложения между собой без учёта регистра и удаление повторяющиеся, оставив только первое из них;
- Реализовать цикл с запросом номера действия;
- Реализовать функцию поиска в тексте конструкции “ <число> sec “ и суммирование этих чисел;
- Реализовать функцию поиска символов ‘%’, ‘#’, ‘@’ и их замены на “<percent>”, “<решетка>”, “(at)” соответственно;
- Реализовать функцию-компаратор для встроенной функции qsort();
- Реализовать функцию удаления предложений по условию;
- Реализовать очистку выделенной динамически памяти в конце работы приложения.

Исходя из поставленных задач, целесообразно разделить их по группам:

- функции ввода-вывода;
- функции обработки текста;
- функция реализации программы.

1.3 Запись т хранение текста

Используем динамическую память, поскольку размер текста неопределен.

Разобьём его на предложения, а предложения — на слова, для удобной дальнейшей обработки.

Реализуем структуры:

Text, Sentence и Word.

Word — строка широких символов, которую в дальнейшем может потребоваться сравнивать или перезаписывать, поэтому полезно иметь в структуре поля *size* и *count_c*, чтобы не вычислять их заново, а также символ *sep* — для хранения запятой, точки или пробела после слова.

Sentence — массив слов, тоже необходимо сохранить *size* и *count_w*, чтобы знать его границы.

Text — массив предложений и его *size* и *count_s*.

Весь текст поступает в программу одной строкой, а символ конца строки служит признаком конца ввода(не противоречит тех.заданию).

Функции *readWord(), readSentence() и readText()* конструируют структуры из посимвольного ввода, передавая результат работы на уровень выше. Функция *printText()* печатает текст, чередуя строки-слова и их сохранённые разделители, добавляя после них пробелы, если они не сохранены (поле *sep* хранит лишь один символ, и если после слова стоит запятая, сохраняется она, а пробел подразумевается и добавляется при печати функцией *printText()*).

Поскольку в тех. задании точка названа разделителем предложений, а не символом окончания, предусмотрим вариант, когда последнее предложение оканчивается не точкой, и символом конца строки. В этом случае он заменяется на точку.

1.4 Обработка текста

Функция *removingDuplicateSentences()* сравнивает попарно предложения в тексте, начиная с конца. Если для рассматриваемого предложения находится предшествующее ему с тем же количеством слов, то слова в них сравниваются попарно — строка со строкой без учёта регистра (при помощи встроенной функции *wcscasecmp()*), разделитель с разделителем. Если отличия не найдены, рассматриваемое предложение удаляется.

Для удаления предложений реализована отдельная функция *removingSentence()*, поскольку эта операция понадобится в дальнейшем. Функция очищает динамическую память, занимаемую предложением, и сдвигает память в массиве предложений на освободившееся место (при помощи встроенной функции *memmove()*).

Операция очистки всей динамической памяти, занимаемой текстом, реализуется в функции *freeText()*. Поскольку здесь также требуется очищать память предложений, соответствующую подзадачу было решено вынести в отдельную функцию — *freeSentence()*.

Первое пользовательское действие выполняет функция *sumSec(),* выполняя поиск пары слов <число> и “sec“. Сумма записывается в переменную типа *float* для дальнейшего преобразования к минутам. Перевод строки широких символов к числу производится с помощью встроенной функции *wcstol().* Функция выводит результат своей работы с помощью встроенной функции *wprintf().*

Второе пользовательское действие выполняется встроенной функцией *qsort(),* для которой написана функция *funccmp(),* реализующая сравнение предложений по сумме кодов первых слов. Для вычисления сумм кодов написана отдельная функция *sumCode().*

Третье пользовательское действие выполняется функцией r*eplaceSymbol(),* которая находит символы ‘%’, ‘#’, ‘@’ и заменяет их на “<percent>”, “<решетка>”, “(at)” соответственно.

Четвертое пользовательское действие выполняется функцией *removingSertainSentences(),* которая находит предложение оканчивающееся словом с тремя согласными подряд и применяет ранее созданную функцию *removingSentence().*

1.5 Обработка команд

Для обработки выбранных пользователем команд и реализована функция *userChoice(),* в которой предусмотрен постоянный запрос команд, а также их выполнение и вывод результатов обработки до тех пор, пока пользователь не решит остановить приложение.

2. СБОРКА ПРОГРАММЫ

Описание структур для хранения текста описано в файле *my_struct.h,* который включается во все файлы с расширением “.c”, в которых они используются.

Функции ввода-вывода *readword(), readsentence(), readtext(), printText(),* а также *userChoice()* объявлены в отдельном файле input_output_text_function.c, которому соответствует заголовочный файл input_output_text_function.h. Все необходимые для этих функций заголовочные файлы стандартной библиотеки и константные выражения прописаны в input_output_text_function.h.

Функции обработки текста сгруппированы в файле *text_processing_function.c*: *removingDuplicateSentences* (), *freeSentence(), freeText(), removingSentence(), sumSec(), removingSertainSentences(), sumCode(), funccmp(), replaceSymbol().* Все необходимые заголовочные файлы стандартной библиотеки, прототипы самих функций прописаны в *text_processing_function.h.*

Оба заголовочных файла, защищены от повторного включения.

Сборка программы осуществляется утилитой make в соответствии с инструкцией.

Программный код представлен в приложении А настоящего отчёта.

3. ТЕСТИРОВАНИЕ

Результаты тестирования представлены ниже. Полностью детали интерактива продемонстрированы на скриншотах в аннотации.

Введите текст:

Се%#йчас перед совр#ем%%енными технологи#ями стоит задача находит@ь дейст#венные способы очи%щения окружающей среды, перер@@аботки мусора. ten sec 15 sec А пластиковые отходы уже стали притчей во языцех, они не разлагаются природным путем и откладываются в природе надолго. В Тихом океане уже расположился новый «континент» плавающего мусора. 34 sec. Удивительные вещи и явления окружают нас. Сегодня, образно выражаясь, можно достать из кармана новости, подключиться к сети, существующей только в информационном поле, передать сообщение человеку, который за тысячи километров от тебя. Удивительные вещи и явления окружают нас. Удивительные вещи и явления окружают нас. Благодаря информационным технологиям мы можем знакомиться с людьми, что живут далеко-далеко, совсем в других странах и городах.

Было удалено 2 повторяющихся предложений.

Се%#йчас перед совр#ем%%енными технологи#ями стоит задача находит@ь дейст#венные способы очи%щения окружающей среды, перер@@аботки мусора. ten sec 15 sec А пластиковые отходы уже стали притчей во языцех, они не разлагаются природным путем и откладываются в природе надолго. В Тихом океане уже расположился новый «континент» плавающего мусора. 34 sec. Удивительные вещи и явления окружают нас. Сегодня, образно выражаясь, можно достать из кармана новости, подключиться к сети, существующей только в информационном поле, передать сообщение человеку, который за тысячи километров от тебя. Благодаря информационным технологиям мы можем знакомиться с людьми, что живут далеко-далеко, совсем в других странах и городах.

Действие 1. Сумма секунд в минутах.

Количество секунд встречающихся в тексте равно 0,816667 минут.

Действие 2. Отсортировать предложения по увеличению сумме кодов символов первого слова в предложении.

34 sec. ten sec 15 sec А пластиковые отходы уже стали притчей во языцех, они не разлагаются природным путем и откладываются в природе надолго. В Тихом океане уже расположился новый «континент» плавающего мусора. Се%#йчас перед совр#ем%%енными технологи#ями стоит задача находит@ь

дейст#венные способы очи%щения окружающей среды, перер@@аботки мусора. Сегодня, образно выражаясь, можно достать из кармана новости, подключиться к сети, существующей только в информационном поле, передать сообщение человеку, который за тысячи километров от тебя. Благодаря информационным технологиям мы можем знакомиться с людьми, что живут далеко-далеко, совсем в других странах и городах. Удивительные вещи и явления окружают нас.

Действие 3. Заменить все символы '%', '#', '@' на "<percent>", "<решетка>", "(at)" соответственно.

34 sec. ten sec 15 sec А пластиковые отходы уже стали притчей во языцех, они не разлагаются природным путем и откладываются в природе надолго. В Тихом океане уже расположился новый «континент» плавающего мусора. Се<precent><решётка>йчас перед совр<решётка>ем<precent><precent>енными технологи<решётка>ями стоит задача находит(at)ь дейст<решётка>венные способы очи<precent>щения окружающей среды, перер(at)(at)аботки мусора. Сегодня, образно выражаясь, можно достать из кармана новости, подключиться к сети, существующей только в информационном поле, передать сообщение человеку, который за тысячи километров от тебя. Благодаря информационным технологиям мы можем знакомиться с людьми, что живут далеко-далеко, совсем в других странах и городах. Удивительные вещи и явления окружают нас.

Действие 4. Удалить все предложения. которые заканчиваются на слово с тремя согласными подряд.

Было удалено 1 предложений

34 sec. ten sec 15 sec А пластиковые отходы уже стали притчей во языцех, они не разлагаются природным путем и откладываются в природе надолго. В Тихом океане уже расположился новый «континент» плавающего мусора. Се<precent><решётка>йчас перед совр<решётка>ем<precent><precent>енными технологи<решётка>ями стоит задача находит(at)ь дейст<решётка>венные способы очи<precent>щения окружающей среды, перер(at)(at)аботки мусора. Сегодня, образно выражаясь, можно достать из кармана новости, подключиться к сети, существующей только в информационном поле, передать сообщение человеку, который за тысячи километров от тебя. Удивительные вещи и явления окружают нас.

Действие 5. Завершить приложение.

Обработка завершена. Программа окончена.

4. ПОЛЬЗОВАТЕЛЬСКАЯ ИНСТРУКЦИЯ

«Программа позволяет производить заданные действия над введённым текстом. Какое именно действие из набора применить, определяете Вы.

1) **Шаг первый** — запуск приложения

Запустите терминал из папки с файлами приложения. Введите команду «make && ./a.out».

2) **Шаг второй** — ввод текста

Введите в терминал текст и нажмите клавишу «Enter».
Текст вводится в одну строку как последовательность предложений, разделённых точкой или точкой с пробелом.
Слова в предложении разделяются запятой, пробелом или запятой и пробелом.

3) **Шаг третий** — выберите действие

Введите номер действия из представленных на экране, нажмите клавишу «Enter».
Результат отобразится на экране.

4) **Шаг четвёртый** — продолжаем работу

Вы можете продолжать применять действия друг за другом к введённому тексту. Если в результате действия текст изменился, следующее действие будет применено к новой версии текста.

5) **Шаг пятый** — выход из приложения

Чтобы завершить работу, введите номер действия «5».
Вы всегда можете запустить приложение снова и ввести новый текст».

ЗАКЛЮЧЕНИЕ

Были изучены новые средства работы с текстом на языке Си. Разработано интерактивное приложение по техническому заданию. Программа собрана, протестирована и отлажена. Написана инструкция для пользователя.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Правила оформления пояснительной записки. // se.moevm.info URL: http://se.moevm.info/doku.php/courses:programming:report (дата обращения: 19.12.2021).

2. Функции стандартных библиотек. // wikipedia.org URL: https://ru.wikipedia.org/wiki/Wchar.h (дата обращения: 18.12.2021).

3. Функции стандартных библиотек и другие инструкции. // cplusplus.com URL: https://www.cplusplus.com/reference (дата обращения: 18.12.2021).

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

Название файла: *my_struct.h*

```
#pragma once

struct Word{
	wchar_t *wchars;	//pointer to wide string
	wchar_t sep;	//word separator
	int count_c;	//word length
};

struct Sentence{
	struct Word *wwords;	//pointer to struct Word
	int count_w;	//count words
};

struct Text{
	struct Sentence *sents; //pointer to struct Sentence
	int count_s;	//count sents
};
```

Название файла: *main.c*

```
include <locale.h>
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include "my_struct.h"
#include "input_output_text_functions.h"
#include "text_processing_functions.h"
```

```
int main(){
	setlocale(LC_ALL, "");
	struct Text *text = malloc(sizeof(struct Text));
	struct Text *t;
	wprintf(L"Введите текст:\n");
	t = readText();
	if (t == NULL){
		wprintf(L"Возникла ошибка выделения памяти. Введенный вами текст слишком большой.\n");
	}
	else{
		*text = *t;
		removingDuplicateSentences(text);
		printText(text);
		userChoice(text);
	}
	return 0;
}
```

Название файла: *input_output_text_function.c*

```
include <wchar.h>
#include <stdlib.h>
#include <stdio.h>
#include "my_struct.h"
#include "text_processing_functions.h"
#define MEM_STEP 5
#define BUF_CHOICE 5
#define WORDS_SEPARATORS L" ,"
#define SENTS_SEPARATORS L".\n"
```

```
struct Word *readWord(){
    int size = MEM_STEP;    //buffer size
    int count_c = 0;        //word length(iteration variable)
    wchar_t *temp = malloc(size*sizeof(wchar_t));   //buffer
    wchar_t sep = L' ';     //words separator
    wchar_t wchar = L' ';   //input char
    if (temp != NULL){      //checking memory allocation
        wchar_t *word = temp;
        while (wchar == L' ')   //skipping space
            wchar = fgetwc(stdin);
        while (!wcschr(WORDS_SEPARATORS, wchar) &&
!wcschr(SENTS_SEPARATORS, wchar)){
            word[count_c++] = wchar;
            if (count_c == size){   //checking for buffer overflow
                size += MEM_STEP;       //increase buffer
                temp = realloc(word, size*sizeof(wchar_t));
                if (temp != NULL)       //checking memory allocation
                    word = temp;
                else{
                    free(word);     //cleaning memory
                    return NULL;
                }
            }
            wchar = fgetwc(stdin);
        }
        word[count_c] = L'\0';
        sep = wchar;
        return &(struct Word){word, sep, count_c};      //casting to type struct
Word
    }
```

```
        free(temp);
        return NULL;
}

struct Sentence *readSentence(){
        int size = MEM_STEP;    //buffer size
        struct Word *temp = malloc(size*sizeof(struct Word));   //buffer
        if (temp != NULL){      //checking memory allocation
                struct Word *sent = temp;
                int count_w = 0;        //count words(iteration variable)
                struct Word *ww = malloc(sizeof(struct Word));  //input wide word
                struct Word *cn;        //variable for check NULL
                do{
                        cn = readWord();
                        if (cn == NULL){        //checking memory allocation
                                freeSentence((struct Sentence){sent, count_w});   //cleaning
memory
                                return NULL;
                        }
                        *ww = *cn;
                        sent[count_w++] = *ww;
                        if (count_w == size){
                                size += MEM_STEP;
                                temp = realloc(sent, size*sizeof(struct Word));
                                if (temp != NULL)       //checking memory allocation
                                        sent = temp;
                                else{
                                        freeSentence((struct Sentence){sent, count_w});   //cleaning
memory
                                        return NULL;
```

```
                }
            }
        }while (!wcschr(SENTS_SEPARATORS, ww->sep));
        free(ww);       //cleaning memory
        return &(struct Sentence){sent, count_w}; //casting to type struct Sentence
    }
    return NULL;
}

struct Text *readText(){
    int size = MEM_STEP;    //buffer size
    struct Sentence *temp = malloc(size*sizeof(struct Sentence));   //buffer
    if (temp != NULL){      //checking memory allocation
        struct Sentence *txt = temp;
        int count_s = 0;        //count sentences(iteration variable)
        struct Sentence *sent = malloc(sizeof(struct Sentence));        //input
sentence
        struct Sentence *cn;    //variable for check NULL
        do{
            cn = readSentence();
            if (cn == NULL){        //checking memory allocation
                freeText(&(struct Text){txt, count_s});   //cleaning memory
                return NULL;
            }
            *sent = *cn;
            if (sent->count_w == 1 && !sent->wwords[0].count_c && sent-
>wwords[0].sep == L'\n'){    //check end text
                free(sent -> wwords[0].wchars); //cleaning memory
                free(sent -> wwords);   //cleaning memory
                break;
```

```
                }
                txt[count_s++] = *sent;
                if (count_s == size){   //checking for buffer overflow
                        size += MEM_STEP;       //increase buffer
                        temp = realloc(txt, size*sizeof(struct Sentence));      //buffer
                        if (temp != NULL){      //checking memory allocation
                                txt = temp;
                        }
                        else{
                                freeText(&(struct Text){txt, count_s});   //cleaning memory
                                return NULL;
                        }
                }
        }while (sent->wwords[sent->count_w - 1].sep != L'\n');
        if ((sent->wwords[sent->count_w - 1].sep) == L'\n')
                (sent->wwords[sent->count_w - 1].sep) = L'.';   //adding a dot to an
unfinished sentence
        free(sent);     //cleaning memory
        return &(struct Text){txt, count_s};
    }
    return NULL;
}

void printText(struct Text *text){
    for (struct Sentence *sent = text -> sents; sent < text -> sents + text -> count_s;
sent++){
        for (struct Word *word = sent -> wwords; word < sent -> wwords + sent -
> count_w; word++){
                fputws(word -> wchars, stdout);
                fputwc(word -> sep, stdout);
```

```
            if (word -> sep != L' ')
                fputwc(L' ', stdout);
        }
    }
    fputwc(L'\n', stdout);
}

void userChoice(struct Text *text){
    wchar_t choice[BUF_CHOICE] = L"";
    while (choice[0] != L'5'){
        wprintf(L"Выберите, что необходимо сделать с текстом:\n");
        wprintf(L"1 - Посчитать и вывести в минутах количество секунд встречающихся в тексте.\n");
        wprintf(L"2 - Отсортировать предложения по увеличению суммы кодов символов первого слова в предложении.\n");
        wprintf(L"3 - Заменить все символы ‘%’, ‘#’, ‘@’ на “<percent>”, “<решетка>”, “(at)” соответственно.\n");
        wprintf(L"4 - Удалить все предложения которые заканчиваются на слово с тремя согласными подряд.\n");
        wprintf(L"5 - Закончить обработку и завершить программу.\n");
        fgetws(choice, BUF_CHOICE, stdin);
        switch (choice[0]){
            case L'1':
                sumSec(text);
                break;
            case L'2':
                wprintf(L"Результат сортировки:\n");
                qsort(text -> sents, text -> count_s, sizeof(struct Sentence), funccmp);
                printText(text);
```

```
                break;
            case L'3':
                wprintf(L"Результат замены:\n");
                replaceSymbol(text);
                printText(text);
                break;
            case L'4':
                removingSertainSentences(text);
                printText(text);
                break;
            case L'5':
                wprintf(L"Обработка завершена. Программа окончена.\n");
                break;
            default:
                wprintf(L"Вы ввели некорректный номер, выберите число от
1 до 5\n");
        }
    }
}
```

Название файла: *input_output_text_function.h*

```
#pragma once
struct Sentence *readSentence();
struct Text *readText();
struct Word *readWord();
void printText(struct Text *text);
void userChoice(struct Text *text);
```

Название файла: *text_processing_function.c*

```
#include <wchar.h>
```

```
#include <stdlib.h>
#include <wctype.h>
#include <string.h>
#include "my_struct.h"
#define COUNSONANTS
L"BbCcDdFfGgHhJjLlKkMmNnPpQqRrSsTtVvWwXxZzБбВвГгДдЖжЗзЙйКкЛлМмНнПпРрСсТтФфХхЦцЧчШшЩщ"

void freeSentence(struct Sentence sent){
	for (struct Word *word = sent.wwords; word < sent.wwords + sent.count_w; word++){
		free(word -> wchars);   //cleaning memory
	}
	free(sent.wwords);      //cleaning memory
}

void freeText(struct Text *text){
	for (int i = 0; i < text -> count_s; i++){
		freeSentence(text -> sents[i]); //cleaning memory
	}
	free(text -> sents);    //cleaning memory
	text -> sents = NULL;
	text -> count_s = 0;
}

void removingSentence(struct Text *text, int index){
	freeSentence(text -> sents[index]);     //cleaning memory
	memmove(text -> sents+index, text -> sents+index+1, (text -> count_s-index-1)*sizeof(struct Sentence)); //shift of sentences in the text
	text -> count_s--;      //overwrite count of sentences in the text
```

```
}

void removingDuplicateSentences(struct Text *text){
	int count = 0;  //count removing sentences
	int coincidence1, coincidence2;
	struct Word *word1;
	struct Word *word2;
	for (int rs = text -> count_s-1; rs > 0; rs--){
		for (int ls = rs-1; ls >= 0; ls--){
			if (text -> sents[ls].count_w == text -> sents[rs].count_w){
				word1 = text -> sents[ls].wwords;
				word2 = text -> sents[rs].wwords;
				coincidence1 = coincidence2 = 0;
				for (int i = 0; i < text -> sents[ls].count_w; i++){
coincidence1 = word1[i].sep != word2[i].sep;
					coincidence2 = wcscasecmp(word1[i].wchars,
word2[i].wchars);
					if (coincidence1 || coincidence2)
						break;
				}
				if (!coincidence1 && !coincidence2){
					removingSentence(text, rs);
					count++;
					break;
				}
			}
		}
	}
	wprintf(L"Было удалено %d повторяющихся предложений.\n", count);
}
```

```
void sumSec(struct Text *text){
	float sum_sec = 0;
	int num;
	int is_sec;
	for (struct Sentence *sent = text -> sents; sent < text -> sents + text -> count_s; sent++){
		if (sent -> count_w >= 2){
			for (struct Word *word = sent -> wwords; word < sent -> wwords + sent -> count_w - 1; word++){
				wchar_t *pEnd;
				num = wcstol(word -> wchars, &pEnd, 10);
				is_sec = wcscasecmp(L"sec", (word+1) -> wchars);
				if (num && !is_sec){
					sum_sec += num;
				}
			}
		}
	}
	wprintf(L"Количество секунд встречающихся в тексте равно %f минут.\n", sum_sec/60);
}

void removingSertainSentences(struct Text *text){
	int count = 0;
	int count_counsonants = 0;
	for (int i = text -> count_s -1; i >= 0; i--){
		int last_word = text -> sents[i].count_w-1;
		struct Word word = text -> sents[i].wwords[last_word];
		for (int j = 0; j < word.count_c; j++){
```

```
                wchar_t s = word.wchars[j];
if (wcschr(COUNSONANTS, s))
                    count_counsonants++;
                else
                    count_counsonants = 0;
                if (count_counsonants == 3){
                    removingSentence(text, i);
                    count++;
                    break;
                }
            }
        }
        wprintf(L"Было удалено %d предложений\n", count);
}

int sumCode(struct Sentence *sent){
        int s = 0;
        for (int i = 0; i < sent -> wwords[0].count_c; i++){
            s += sent -> wwords[0].wchars[i];
        }
        return s;
}

int funccmp(const void *x1, const void *x2){
        int sc1 = sumCode((struct Sentence*)x1);
        int sc2 = sumCode((struct Sentence*)x2);
        return sc1 - sc2;
}

void replaceSymbol(struct Text *text){
```

```
for (struct Sentence *sent = text -> sents; sent < text -> sents + text -> count_s; sent++){
        for (struct Word *word = sent -> wwords; word < sent -> wwords + sent -> count_w; word++){
                int size = word -> count_c + 1;
                wchar_t *new_word = malloc(size*sizeof(wchar_t));
                int j = 0;
                for (int i = 0; i <= word -> count_c; i++){
                        if (word -> wchars[i] != L'%' && word -> wchars[i] != L'#' && word -> wchars[i] != L'@'){
                                new_word[j++] = word -> wchars[i];
                        }
                        else{
                                if (word -> wchars[i] == L'%' || word -> wchars[i] == L'#'){
                                        size += 8;
                                        new_word = realloc(new_word, size*sizeof(wchar_t));
if (word -> wchars[i] == L'%'){
                                          wchar_t prec[] = L"<precent>\0";
                                          for (int k = 0; prec[k]; k++){
                                                new_word[j++] = prec[k];
                                          }
                                        }
                                        if (word -> wchars[i] == L'#'){
                                         wchar_t resh[] = L"<решётка>\0";
                                                for (int k = 0; resh[k]; k++){
                                                        new_word[j++] = resh[k];
                                                }
                                        }
                                }
```

```
                    else{
                        size += 3;
                        new_word = realloc(new_word, size*sizeof(wchar_t));
                        if (word -> wchars[i] == L'@'){
                            new_word[j++] = L'(';
                            new_word[j++] = L'a';
                            new_word[j++] = L't';
                            new_word[j++] = L')';
                        }
                    }
                }
            }
            free(word -> wchars);
            word -> wchars = new_word;
            word -> count_c = j;
        }
    }
}
```

Название файла: *text_processing_function.h*

```
#pragma once
void removingSentence(struct Sentence sent);
void freeSentence(struct Sentence sent);
void freeText(struct Text *text);
void removingDuplicateSentences(struct Text *text);
void sumSec(struct Text *text);
void removingSertainSentences(struct Text *text);
int sumCode(struct Sentence *sent);
int funccmp(const void *x1, const void *x2);
void replaceSymbol(struct Text *text);
```

Название файла: *Makefile*

```
CC = gcc

all: main.o input_output_text_functions.o text_processing_functions.o
	$(CC) main.o input_output_text_functions.o text_processing_functions.o

main.o: main.c input_output_text_functions.h text_processing_functions.h
my_struct.h
	$(CC) -c main.c

input_output_text.o: input_output_text_functions.c text_processing_functions.h
my_struct.h
	$(CC) -c input_output_text_functions.c

text_processing_functions.o: text_processing_functions.c my_struct.h
	$(CC) -c text_processing_functions.c

clean:
	rm -r *.o
```