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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка текста на языке Си

Студентка гр. 3384 _____ Конасова Я. О.

Преподаватель _____ Глазунов С. А.

Санкт-Петербург

2023

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Конасова Я. О.

Группа 3384

Тема работы: Обработка текста на языке Си

Исходные данные:
Вариант 5.5
Вывод программы должен быть произведен в стандартный поток вывода: *stdout*. Ввод данных в программе в стандартный поток ввода: *stdin*.
Первой строкой при запуске программы нужно выводить информацию о варианте курсовой работе и об авторе программы. 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 текст в виде динамического массива предложений и оперировать далее только с ним. Функции обработки также должны принимать на вход либо текст, либо предложение.
Программа должна найти и удалить все повторно встречающиеся предложения.
Программа должна выполнить одно из введенных пользователем действий и завершить работу:

1) Распечатать каждое слово (с учётом регистра) и количество его повторений в тексте.

2) Заменить каждый символ, который не является буквой, на его код.
3) Отсортировать предложения по количеству латинских букв в предложении.
4) Удалить все предложения, которые содержат специальные символы и не содержат заглавные буквы. К специальным символам относятся: #$^%@

Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов. Также, должен быть написан Makefile.

Содержание пояснительной записки:
«Аннотация», «Содержание», «Введение», «Библиотеки и структуры», «Первичная обработка текста», «Реализация основных функций», «Makefile», «Заключение», «Список использованных источников», «Приложение А. Тестирование программы», «Приложение Б. Исходный код»

Предполагаемый объем пояснительной записки:
Не менее 30 страниц.

Дата выдачи задания: 16.10.2023

Дата сдачи реферата: 08.12.2023

Дата защиты реферата: 15.12.2023

Студентка _____ Конасова Я.О

Преподаватель _____ Глазунов С.А

АННОТАЦИЯ

Разработанная программа считывает число, которое вводит пользователь и на основании того, какое число было введено, выполняет либо одну из четырех функций, либо выводит справку о функциях, либо выполняет первичную обработку текста, либо не выполняется вовсе. Для считывания текста написаны две функции, одна из которых считывает предложение и возвращает указатель на динамически выделенную структуру, в которой хранится информация о предложении. Предложение хранится в виде массива широких символов. Вторая функция динамически выделяет память для массива указателей на структуры. В зависимости от введенного пользователем числа, проводится определенная обработка текста, который также вводит пользователь. Ввод завершается двумя символами переноса строки.

Пример работы программы приведен в приложении А.

Исходный код программы приведен в приложении Б.

SUMMARY

The developed program reads the number that the user enters and, based on which number was entered, performs either one of the four functions, or outputs a help about the functions, or performs primary text processing, or is not performed at all. Two functions are written to read the text, one of which reads the sentence and returns a pointer to a dynamically allocated structure that stores information about the sentence. The sentence is stored as an array of wide characters. The second function dynamically allocates memory for an array of pointers to structures. Depending on the number entered by the user, certain text processing is performed, which is also entered by the user. The input ends with two line breaks.

An example of how the program works is given in Appendix A.

The source code of the program is given in Appendix B.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: Написать программу на языке Си, которая считывает и обрабатывает текст.

Задачи, которые необходимо для этого решить:

1) Реализовать функции для ввода и вывода текста.
2) Реализовать функции для стандартной обработки текста.
3) Реализовать функции для специальной обработки текста.

1. БИБЛИОТЕКИ И СТРУКТУРЫ

1.1. Необходимые библиотеки.

Для успешного написания программы необходимо подключить библиотеки. Первая из них - <*stdio.h*> (Standard Input/Output): Эта библиотека предоставляет основные функции ввода/вывода. В коде используются функции, такие как *wprintf* (вывод форматированного текста в стандартный вывод) и *getwchar* (считывание одного широкого символа из стандартного ввода). Также нам необходима <*stdlib.h*> (Standard Library). Эта библиотека предоставляет функции для управления памятью, выполнения системных вызовов и другие общие утилиты. В коде используются функции *malloc* и *realloc* для динамического выделения и перераспределения памяти. <*wchar.h*> (Wide Characters). Эта библиотека предоставляет функции для работы с широкими символами (Unicode). В коде используются тип данных *wchar_t* и функции для работы с широкими символами, такие как *iswblank* (проверка, является ли символ пробельным) и работа с вводом-выводом широких символов. <*wctype.h*> (Wide Characters Classification). Эта библиотека предоставляет функции для классификации широких символов по категориям, таким как буквы, цифры и пр. <*string.h*> (String Handling). Эта библиотека предоставляет функции для работы со строками. Используется функция *memmove* для перемещения блока памяти при удалении предложений. <*locale.h*> (Localization). Эта библиотека позволяет программе адаптироваться к локализации, поддерживая различные языки и региональные настройки. В коде она используется для установки текущей локали при помощи функции *setlocale*.

1.2. Использование структур

Для хранения введенного пользователем текста необходимо создать две структуры. Одна из них – *Sentence*. В ней будет хранится указатель на массив широких символов sentence, указатель на указатель на структуры *Word*, представляющие слова в предложении (*pointers_word*). Также в структуре будет

хранится информация о количестве слов в предложении *number_of_words* и количество латинских символов в предложении *amount_of_lat*, что понадобится для функции, реализуемой далее. Вторая необходимая структура *Text*, в ней хранится указатель на указатель на структуры Sentence, представляющие предложения в тексте (*pointers_sentence*). Также в структуре содержится информация о количестве предложений в тексте *number_of_sentence*.

Чтобы реализовать требуемые функции необходимо также добавить структуру Word. В структуре будет хранится указатель word на массив широких символов, которые представляют слово, информацию о длине слова len_of_word и количество вхождений этого слова в текст.

2. ПЕРВИЧНАЯ ОБРАБОТКА ТЕКСТА

2.1. Считывание текста

Для чтения предложения используется функция *read_sentence*. Она выделяет динамическую память для массива широких символов *sent*, представляющего предложение. Используется функция *malloc* с начальным размером *MASSIV*. Происходит чтение символов в цикле с условием выхода при обнаружении точки (*'.'*) или двух подряд идущих символов новой строки (*'\n'*). Происходит динамическое расширение массива при необходимости с использованием *realloc*. Проводится обработка случаев пробелов и управление указателями для корректного чтения предложения. Возвращается указатель на структуру *Sentence*, содержащую считанное предложение.

Далее необходимо реализовать функцию *read_text*, которая будет вызывать функцию *read_sentence*, сохранять возвращаемый указатель в массив, таким образом собирая все указатели на предложения в один массив. Для этого выделяется динамическая память для массива указателей на структуры *Sentence* (*points_sentence*). В цикле считываются предложения с использованием функции *read_sentence*. При необходимости происходит динамическое расширение массива указателей. Цикл завершается при обнаружении двух подряд идущих символов новой строки в конце последнего считанного предложения. Создается структура *Text*, содержащая указатели на считанные предложения и их количество. Возвращается структура *Text*.

Для дальнейшей работы с текстом нам будет удобно реализовать функцию *words*. Она итерируется по предложениям в тексте. Для каждого предложения инициализирует переменные *size_word* и *amount_of_words* для отслеживания размера и количества слов в предложении. Функция выделяет динамическую память для массива указателей на структуры *Word* (*words_points*) с начальным размером *MASSIV*, клонирует предложение, чтобы избежать модификации исходного. Использует *wcstok* для разделения предложения на слова с разделителями *" ,."*. Для каждого слова проверяет и, если необходимо,

увеличивает размер массива указателей на структуры *Word* с использованием *realloc*. Выделяет память для структуры *Word* и для массива широких символов, представляющего слово. Заполняет структуру *Word* информацией о слове, добавляет указатель на структуру *Word* в массив *words_points*, назначает массив *words_points* как указатели на слова в соответствующем предложении. Записывает количество слов в предложении. Освобождает скопированное предложение, так как оно больше не нужно.

2.2. Удаление пустых строк и повторяющихся предложений

Для начала необходимо реализовать функцию, которая удаляет предложения *del_sent*. Она принимает указатель на структуру *Text* и индекс предложения *numb_of_sent* для удаления. Сначала сохраняет указатель на предложение, которое нужно удалить (*del_sentence*). Затем использует функцию *memmove* для перемещения указателей на предложения в массиве так, чтобы удаленное предложение освободило место. Освобождает память, выделенную для удаленного предложения с использованием *free*. Уменьшает счетчик числа предложений в структуре *Text*.

Теперь, когда у нас есть функция, которая удаляет предложения, нам необходимо написать функции, которые проверяют, является ли предложение дубликатом или оно вовсе пустое, чтобы мы могли вызвать функцию удаления предложения. Первая их них - *del_empty_sent*. Она принимает указатель на структуру *Text*, итерируется по всем предложениям в тексте и вызывает *del_sent*, если предложение не содержит слов, т. е. удаляет пустые предложения, анализируя их количество слов. Вторая - *del_similar_sent*. Она принимает указатель на структуру *Text*, использует два вложенных цикла для сравнения каждого предложения с остальными. Если два предложения идентичны (без учета регистра символов), то вызывается *del_sent* для удаления одного из них. Происходит удаление предложений, которые идентичны другим предложениям в тексте.

2.3. Освобождение памяти

Так как мы динамически выделяем память для считывания текста не стоит забывать и о ее освобождении. Для этого реализуем функцию *free_Word*, которая принимает указатель на структуру *Word*. Она освобождает память, выделенную для массива широких символов, представляющего слово. Завершает освобождение памяти, освобождая саму структуру *Word*. Далее реализуем *free_Sentence*. Она принимает указатель на структуру *Sentence*, итерируется по массиву указателей на структуры *Word* в предложении и вызывает *free_Word* для освобождения памяти каждого слова. Также освобождает память для массива указателей на слова в предложении (*pointers_word*) и освобождает память для массива широких символов, представляющих предложение (*sentence*). Завершает освобождение памяти, освобождая саму структуру *Sentence*. Функция *free_Text* принимает указатель на структуру *Text*, итерируется по массиву указателей на структуры *Sentence* в тексте и вызывает *free_Sentence* для освобождения памяти каждого предложения. Затем освобождает память для массива указателей на предложения в тексте (*pointers_sentence*). Завершает освобождение памяти, освобождая саму структуру *Text*.

3. РЕАЛИЗАЦИЯ ОСНОВНЫХ ФУНКЦИЙ

3.1. Функция countWords

Согласно заданию, первая реализуемая мной функция должна распечатать каждое слово с учетом регистра и количество его повторений в тексте. Реализуем функцию *countWords*, которая принимает указатель на структуру *Text* в качестве аргумента и подсчитывает количество вхождений каждого уникального слова в тексте. В функции создается динамический массив *struct WordCount* под названием *word_counts* для хранения уникальных слов и их количества. *size_word_counts* отслеживает количество элементов в этом массиве. Происходит итерация по каждому предложению в *struct Text* и каждому слову в предложении. Для каждого слова проверяется, присутствует ли оно уже в массиве *word_counts*, путем сравнения существующих слов. Если слово не найдено (*word_index* == -1), выделяется память для массива *word_counts*, чтобы вместить новое слово, затем добавляется слово и устанавливается его количество равным 1. Если слово найдено, увеличивается количество этого слова в массиве *word_counts*. После подсчета всех слов выводится результат, включая каждое уникальное слово и его количество. В конце освобождается память, выделенная для каждого слова, а затем для всего массива *word_counts*.

3.2. Функция print_Text_for_two

Вторая функция должна заменить каждый символ, который не является буквой, на его код. Реализуем функцию *print_Text_for_two*, которая принимает указатель на структуру *Text*. Внешний цикл *for* проходит по каждому предложению в структуре *Text*. Внутренний цикл *for* проходит по каждому символу в текущем предложении. Для каждого символа выполняются следующие действия: если символ является буквой *(iswalpha(a))*, он выводится без изменений, с типом данных *wchar_t*. Если символ - точка *(a==L'.')*, выводится точка, а затем символ новой строки для отделения предложений. Если символ - пробел *(a==L' ')*, выводится пробел. Если символ - символ новой строки

(a==L'\n'), выводится символ новой строки. В противном случае (если символ не является буквой, точкой, пробелом или символом новой строки), выводится числовое представление символа. Таким образом, программа сохранит внешний вид текста после первичной обработки и заменит на код только те символы, которые не являются буквами.

3.3. Функция sort_sentence

Третья функция должна отсортировать предложения по количеству латинских букв в предложении. Для этого сначала реализуем функцию *count_lat*, которая будет считать количество латинских символов в предложении и записывать это количество в структуру *Sentence*. Внешний цикл *for* проходит по каждому предложению в структуре *Text*. Первый внутренний цикл *for* проходит по каждому слову в текущем предложении, второй внутренний цикл *for* проходит по каждому символу в текущем слове. Для каждого символа выполняются следующие действия: проверяется, является ли текущий символ латинской буквой, сравнивая его с диапазонами кодов Unicode для строчных и прописных букв латинского алфавита. Если символ является латинской буквой, увеличивается счетчик *amount_lat*. После завершения внутренних циклов количество латинских букв для текущего предложения сохраняется в поле *amount_of_lat* структуры *Sentence*.

Теперь мы можем реализовать саму функцию для сортировки предложений. Эта функция вызывает *count_lat* перед началом сортировки, чтобы подсчитать количество латинских символов в каждом предложении. Внешний цикл функции проходится по каждому предложению в тексте. Внутренний цикл используется для сравнения и перестановки предложений в порядке возрастания количества латинских букв. Если количество латинских букв в текущем предложении больше, чем в следующем предложении, то выполняется перестановка предложений с использованием временной переменной *(temp)*. Таким образом, функция *sort_sentence* использует сортировку пузырьком для упорядочивания предложений в тексте по количеству латинских букв в них.

3.4. Функция delete_sent

Четвертая функция должна удалить все предложения, которые содержат специальные символы и не содержат заглавные буквы. Для этой функции также реализуем сначала вспомогательную, которая проверяет, есть ли в предложении хоть одна заглавная буква. Функция *check_registr* начинается с объявления переменной *flag*, которая будет использоваться для отслеживания наличия заглавных букв в предложении. Изначально *flag* устанавливается в 0. Два вложенных цикла используются для итерации по словам и символам в каждом слове предложения. Внутренний цикл проверяет каждый символ в слове на наличие заглавных букв с использованием функции *iswupper*, если символ является заглавной буквой, увеличиваем счетчик *flag*. После завершения всех итераций, проверяем значение *flag*. Если *flag* остался равным 0, это означает, что в предложении нет заглавных букв, и функция возвращает 1. В противном случае, если хотя бы одна заглавная буква была найдена, функция возвращает 0.

Теперь осталось реализовать основную функцию *delete_sent*. Внешний цикл итерируется по всем предложениям в тексте, создается переменная *flag,* которая будет использоваться для отслеживания наличия определенных символов в предложении. Используем функцию *check_registr* для проверки наличия заглавных букв в предложении. Если предложение содержит заглавные буквы, мы переходим к внутреннему циклу, который проверяет каждый символ в предложении на совпадение с определенными символами *('#', $, ^, %, @).* Если текущий символ совпадает с одним из указанных символов, увеличиваем счетчик *flag*. После завершения проверки всех символов в предложении, мы проверяем, были ли найдены указанные символы. Если да, вызываем функцию *del_sent* для удаления текущего предложения, а затем уменьшаем *i* на 1, чтобы корректно обработать следующее предложение после удаления.

4. MAKEFILE

Makefile - это текстовый файл, содержащий инструкции для утилиты *make*, которая автоматизирует процесс сборки программного проекта. *Makefile* определяет зависимости между файлами и команды для их компиляции, линковки и других задач. В данной работе цель *all* указывает *make*, что по умолчанию должно быть выполнено. Здесь *all* зависит от *cw*. Сама цель *cw* зависит от файлов объектного кода *cw.o, actions_with_sentences.o, main_functions.o,* и *stuctures_discribe.o.* Если один из этих файлов изменится, цель *cw* будет пересобрана. Команда *gcc* компилирует эти объектные файлы в исполняемый файл *cw. actions_with_sentences.o: actions_with_sentences.c actions_with_sentences.h:* Это правило говорит *make*, что объектный файл *actions_with_sentences.o* зависит от исходных файлов *actions_with_sentences.c* и заголовочного файла *actions_with_sentences.h.* Если эти файлы изменяются, *make* перекомпилирует *actions_with_sentences.o.* Команда *gcc* -c *actions_with_sentences.c* компилирует исходный файл в объектный файл без линковки. *main_functions.o: main_functions.c main_functions.h:* Аналогично первому правилу, объектный файл *main_functions.o* зависит от исходных файлов *main_functions.c* и заголовочного файла *main_functions.h*. При изменении этих файлов *make* перекомпилирует *main_functions.o. stuctures_discribe.o: stuctures_discribe.c structures_discribe.h:* Снова аналогичное правило для объектного файла *stuctures_discribe.o,* зависящего от исходных файлов *stuctures_discribe.c* и заголовочного файла *structures_discribe.h.* Цель *clean* предназначена для удаления временных и созданных в процессе компиляции файлов. Когда вводится *make clean*, все файлы с расширением *.o* и исполняемый файл *cw* удаляются.

ЗАКЛЮЧЕНИЕ

Была написана программа, считывающая число, введенное пользователем, а далее, если это число находится в диапазоне между 0 и 4, также считывающая текст. Над текстом проводится первичная обработка – удаление пустых строк и одинаковых предложений. Далее может выполнится одна из четырех функций: подсчет уникальных слов и количество их повторений в тексте; замена символов, не являющихся буквами, на их код; сортировка предложений по количеству в них латинских символов и удаление предложений со специальными символами, но без заглавных букв. Если пользователь ввел число 5, то выводится справка о функциях. Также к программе был написан Makefile.

Для успешной реализации программы была проведена работа со структурами, указателями, динамическими массивами. Изучена специфика работы с широкими символами. Были использованы функции работы со строками и символами. Функции программы распределены по файлам, к которым были написаны заголовочные файлы. Для сборки программы использовался Makefile.

Программа была успешно протестирована. Примеры тестирования см. в приложении А, исходный код см. в приложении Б.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Курс программирования на языке Си: конспект лекций/ С.Т. Касюк. – Челябинск: Издательский центр ЮУрГУ, 2010. – 175с.
2. Основы программирования на языке Си / А.И. Солдатов; Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2015. – 118с.
3. Язык программирования Си / Б. Керниган, Д. Ритчи. Пер. с англ., 3-е изд., испр. – СПб.: «Невский диалект», 2001. 352 с: ил.

ПРИЛОЖЕНИЕ А

ТЕСТИРОВАНИЕ ПРОГРАММЫ

Тестирование 1. Первичная обработка текста.

```
PS D:\study\course> ./cw
Course work for option 5.5, created by Konasova Yana.
0
It's test. SAY HI! jojo. jojo.

русский? окей. таблуляция       окей.
kjsfljsg ^*&^%*(. %$^&*^&*9. 5677678 sfgjsffm.

It's test.
SAY HI! jojo.
jojo.
русский? окей.
таблуляция      окей.
kjsfljsg ^*&^%*(.
%$^&*^&*9.
5677678 sfgjsffm.
```

Тестирование 2. Функция print_Text_for_two.

```
PS D:\study\course> ./cw
Course work for option 5.5, created by Konasova Yana.
2
hhhaahahah j768 l!@@ sjkfsf$$%%^^sfdj.
sfkk. &(_ASF

hhhaahahah j555456 l336464 sjkfsf363637379494sfdj.
sfkk.
384095ASF
```

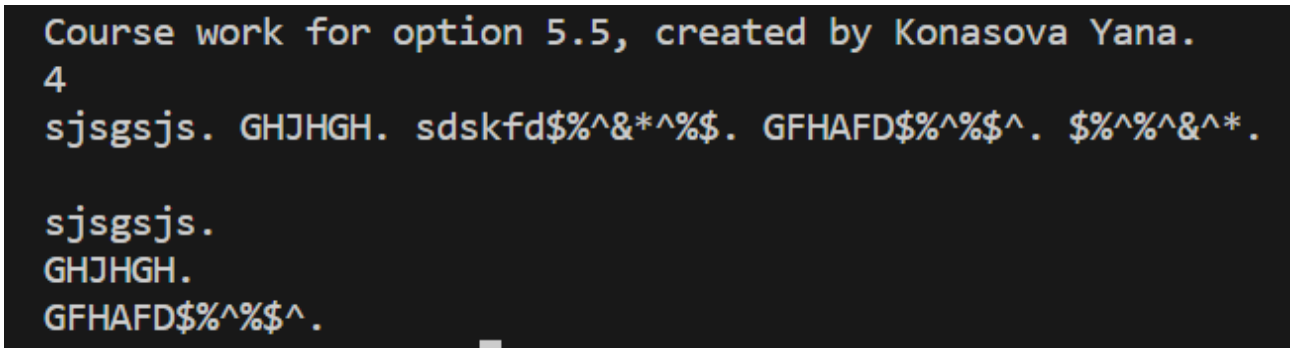
Тестирование 3. Функция countWords.

```
PS D:\study\course> ./cw
Course work for option 5.5, created by Konasova Yana.
1
ok, i gonna ckeck. jojo jojo jojo.
what's better 3476 2738 77 ^^*&*^67.
Русский не забываем. нет нет нет.
Хорошо, отлично.

ok 1
i 1
gonna 1
ckeck 1
jojo 3
what's 1
better 1
3476 1
2738 1
77 1
^^*&*^67 1
Русский 1
не 1
забываем 1
нет 3
Хорошо 1
отлично 1
```

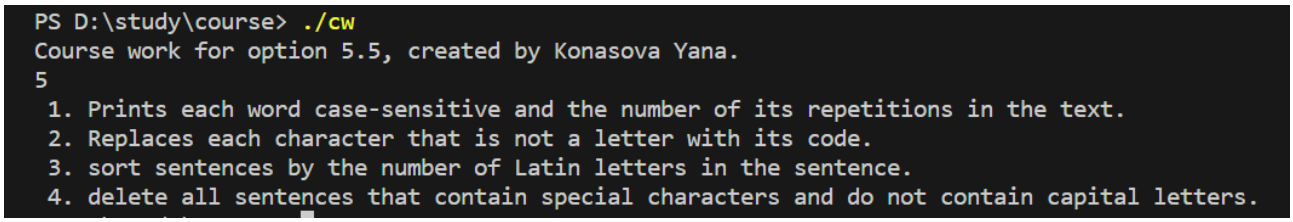
Тестирование 4. Функция sort_sentence.

```
PS D:\study\course> ./cw
Course work for option 5.5, created by Konasova Yana.
3
j. jj. jjjjjjj. w6742. %^&$(#@*#. %^^&^fhhs.

%^&$(#@*#.
j.
w6742.
jj.
%^^&^fhhs.
jjjjjjj.
```

Тестирование 5. Функция delete_sent.



Тестирование 6. Вывод справки.



ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД

Название файла: actions_with_sentences.c

```
void del_sent(struct Text *text, int numb_of_sent){
    struct Sentence *del_sentence = text-
>pointers_sentence[numb_of_sent];
    memmove(&text->pointers_sentence[numb_of_sent], &text-
>pointers_sentence[numb_of_sent+1], sizeof(struct Sentence*)*(text-
>number_of_sentence-numb_of_sent-1));
    free(del_sentence);
    text->number_of_sentence=text->number_of_sentence-1;
}

void del_empty_sent (struct Text *text){
    for (int i=0; i<text->number_of_sentence; i++){
        if (text->pointers_sentence[i]->number_of_words==0){
            del_sent(text, i);
        }
    }
}

void del_similar_sent (struct Text *text){
    for (int i =0; i<text->number_of_sentence-1; i++){
        int j=i+1;
        while(j<text->number_of_sentence){
            if (wcscasecmp(text->pointers_sentence[i]->sentence, text-
>pointers_sentence[j]->sentence)==0){
                del_sent(text, j);
            }else{
                j+=1;
            }
        }
    }
}

int check_registr(struct Sentence *sentence){
    int flag=0;
    for (int j =0; j<sentence->number_of_words; j++){
        for (int l =0; l<sentence->pointers_word[j]->len_of_word; l++){
            wchar_t a=sentence->pointers_word[j]->word[l];
            if (iswupper(a)){
                flag++;
            }
        }
    }
    if (flag==0){
        return 1;
    }else{
        return 0;
    }
}

void count_lat(struct Text *text){
    for (int i =0; i<text->number_of_sentence; i++){
        int amount_lat=0;
```

```
        for (int j =0; j<text->pointers_sentence[i]->number_of_words;
j++){
            for (int l =0; l<text->pointers_sentence[i]-
>pointers_word[j]->len_of_word; l++){
                wchar_t a=text->pointers_sentence[i]->pointers_word[j]-
>word[l];
                if (((a>=L'a')&&(a<=L'z'))||((a>=L'A')&&(a<=L'Z'))){
                    amount_lat++;
                }
            }
        }
        text->pointers_sentence[i]->amount_of_lat=amount_lat;
    }
}
void trim(struct Text *text){
    for (int i = 0; i<text->number_of_sentence; i++){
        int begin=0;
        int end = wcslen(text->pointers_sentence[i]->sentence)-1;
        while(iswspace(text->pointers_sentence[i]->sentence[begin])){
            begin++;
        }
        while(end>=begin && iswspace(text->pointers_sentence[i]-
>sentence[end])){
            end--;
        }
        int j=0;
        for (j; j<=end - begin; j++){
            text->pointers_sentence[i]->sentence[j]=text-
>pointers_sentence[i]->sentence[begin+j];
        }
        text->pointers_sentence[i]->sentence[j]=L'\0';
    }
}
```

Название файла: actions_with_sentences.h

```
#pragma once

#include "structures.h"
int wcscasecmp(const wchar_t* s1, const wchar_t* s2);
wchar_t* wcsdup(const wchar_t* str);
void del_sent(struct Text *text, int numb_of_sent);
void del_empty_sent (struct Text *text);
void del_similar_sent (struct Text *text);
int check_registr(struct Sentence *sentence);
void count_lat(struct Text *text);
void trim(struct Text *text);
```

Название файла: cw.c

```
#include "structures.h"
#include "actions_with_sentences.h"
#include "structures_discribe.h"
#include "main_functions.h"
int main(){
    setlocale(LC_ALL, "");
    wprintf(L"Course work for option 5.5, created by Konasova Yana.\n");
    int enter;
    struct Text text;
```

```
    wscanf(L"%d", &enter);
    if (enter==5){
        print_information_about_func();
    }
    else if ((enter>=0)&&(enter<=4)){
        text=read_text();
        trim(&text);
        words(&text);
        del_empty_sent (&text);
        del_similar_sent (&text);
    }
    else{
        wprintf(L"Incorrect data.");
    }
    switch(enter){
        case 0: {
            print_Text(&text);
            free_Text(&text);
            break;
        }
        case 1: {
            countWords(&text);
            free_Text(&text);
            break;
        }
        case 2: {
            print_Text_for_two(&text);

            free_Text(&text);
            break;
        }
        case 3: {
            sort_sentence(&text);
            print_Text(&text);
            free_Text(&text);
            break;
        }
        case 4: {
            delete_sent(&text);
            print_Text(&text);
            free_Text(&text);
            break;
        }
    }


    return 0;
}
```

Название файла: main_functions.c

```
#include "main_functions.h"
#include "actions_with_sentences.h"
void countWords(struct Text* text){
    struct WordCount* word_counts = NULL;
    int size_word_counts = 0;
    for (int i = 0; i < text->number_of_sentence; i++) {
```

```
        for (int j = 0; j < text->pointers_sentence[i]->number_of_words;
j++) {
            wchar_t* current_word = text->pointers_sentence[i]-
>pointers_word[j]->word;
            int word_index = -1;
            for (int k = 0; k < size_word_counts; k++) {
                if (wcscmp(current_word, word_counts[k].word) == 0) {
                    word_index = k;
                    break;
                }
            }
            if (word_index == -1) {
                if (size_word_counts == 0) {
                    word_counts = malloc(sizeof(struct WordCount));
                } else {
                    word_counts = realloc(word_counts, (size_word_counts
+ 1) * sizeof(struct WordCount));
                }
                if (word_counts == NULL) {
                    wprintf(L"Error: Ошибка перераспределения
памяти.\n");
                    exit(0);
                }
                word_counts[size_word_counts].word =
wcsdup(current_word);
                word_counts[size_word_counts].count = 1;
                size_word_counts++;
            } else {
                word_counts[word_index].count++;
            }
        }
    }

    for (int i = 0; i < size_word_counts; i++) {
        wprintf(L"%ls: %d\n", word_counts[i].word, word_counts[i].count);
        free(word_counts[i].word);
    }

    free(word_counts);
}

void delete_sent(struct Text* text){
    for (int i =0; i<text->number_of_sentence; i++){
        int flag=0;
        if (check_registr(text->pointers_sentence[i])){
            for (int j =0; j<text->pointers_sentence[i]->number_of_words;
j++){
                for (int l =0; l<text->pointers_sentence[i]-
>pointers_word[j]->len_of_word; l++){
                    wchar_t a=text->pointers_sentence[i]-
>pointers_word[j]->word[l];
                    if ((a==L'#')||(a==L'$')||(a==L'^')|| (a==L'%') ||
(a==L'@')){
                        flag++;
                    }
                }
            }
```

```
        }
        if (flag!=0){
            del_sent(text, i);
            i--;
        }
    }
}

void print_Text_for_two(struct Text* text){
    for (int i=0; i<text->number_of_sentence; i++){
        for (int j=0; j<wcslen(text->pointers_sentence[i]->sentence);
j++){
            wchar_t a = text->pointers_sentence[i]->sentence[j];
            if ((iswalpha(a))||(a >= L'\u0400' && a <= L'\u04FF')){
                wprintf(L"%c", a);
            }else if(a==L'.'){
                wprintf(L"%c\n", a);
            }else if(a==L' '){
                wprintf(L" ", a);
            }else if(a==L'\n'){
                wprintf(L"\n", a);
            }else {
                wprintf(L"%d", a);
            }
        }
    }
}


void sort_sentence(struct Text* text){
    count_lat(text);
    for (int i = 0; i < text->number_of_sentence; i++) {

        for (int j = 0; j < text->number_of_sentence-i-1; j++) {
            if (text->pointers_sentence[j]->amount_of_lat > text-
>pointers_sentence[j+1]->amount_of_lat) {

                struct Sentence* temp = text->pointers_sentence[j];
                text->pointers_sentence[j] = text-
>pointers_sentence[j+1];
                text->pointers_sentence[j+1]= temp;
            }
        }
    }
}

void print_information_about_func(){
    wprintf(L"1. Prints each word case-sensitive and the number of its
repetitions in the text.\n 2. Replaces each character that is not a
letter with its code.\n 3. sort sentences by the number of Latin letters
in the sentence.\n 4. delete all sentences that contain special
characters and do not contain capital letters.");
}
```

Название файла: main_functions.h

```
#pragma once
```

```
#include "structures.h"
void countWords(struct Text* text);
void delete_sent(struct Text* text);
void sort_sentence(struct Text* text);
void print_information_about_func();
void print_Text_for_two(struct Text* text);
```

Название файла: structures.h

```
#pragma once
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <wctype.h>
#include <string.h>
#include <locale.h>

#define MASSIV 100
#define STEP 20

struct Word{
    wchar_t* word;
    int len_of_word;
    int amount_in_text;
};

struct Sentence{
    wchar_t* sentence;
    struct Word ** pointers_word;
    int number_of_words;
    int amount_of_lat;
};

struct Text{
    struct Sentence ** pointers_sentence;
    int number_of_sentence;
};

struct WordCount {
    wchar_t* word;
    int count;
};
```

Название файла: structures_discribe.h

```
#pragma once

#include "structures.h"

struct Sentence* read_sentence();
struct Text read_text();
void words(struct Text *text);
void print_Text(struct Text *text);
void free_Text(struct Text *text);
void free_Sentence(struct Sentence *sentence);
void free_Word(struct Word *word);
wchar_t* wcsdup(const wchar_t* str);
```

Название файла: stuctures_discribe.c

```
#include "structures_discribe.h"
#include "actions_with_sentences.h"

struct Sentence* read_sentence(){
    wchar_t* sent = malloc(MASSIV*sizeof(wchar_t));
    wchar_t symbol_now;
    wchar_t symbol_prev;
    int size_sent=MASSIV;
    int len_sent=0;
    int already_blank=0;

    do{
        symbol_now=getwchar();
        if ((iswblank(symbol_now))&&(already_blank==1)){
            continue;
        }if ((iswblank(symbol_now))&&(already_blank==0)){
            already_blank=1;
        }if (!(iswblank(symbol_now))) {
            already_blank=0;
        }
        if (len_sent<size_sent-1){
            sent[len_sent]=symbol_now;
            sent[len_sent+1]=L'\0';
            len_sent++;
        } else{
            size_sent=size_sent+STEP;
            sent = (wchar_t*)realloc(sent, size_sent*sizeof(wchar_t));
            if ( sent== NULL) {
                wprintf(L"Error: Ошибка перераспределения памяти.\n");
                free(sent);
                exit(0);
            }
            else{
                sent[len_sent]=symbol_now;
                sent[len_sent+1]=L'\0';
                len_sent++;
            }

        } if
((symbol_now==L'.')||((symbol_now==L'\n')&&(symbol_prev==L'\n'))) break;
        symbol_prev=symbol_now;

    }while (1);
    struct Sentence *sentence = malloc(sizeof(struct Sentence));
    sentence->sentence=sent;
    return sentence;
}

struct Text read_text(){
    int size_text = MASSIV;
    int amount_of_sentence=0;
    struct Sentence **points_sentence = malloc(size_text*sizeof(struct
Sentence*));
    if ( points_sentence== NULL) {
        wprintf(L"Error: Ошибка перераспределения памяти.\n");
        free(points_sentence);
        exit(0);
```

```
    }
    struct Sentence *sentence_now;
    do{
        if (amount_of_sentence<size_text){
            sentence_now=read_sentence();
        }else{
            size_text+=STEP;
            struct Sentence **n_points_sentence =
realloc(points_sentence, size_text*sizeof(struct Sentence*));
            if ( n_points_sentence== NULL) {
                wprintf(L"Error: Ошибка перераспределения памяти.\n");
                free(points_sentence);
                exit(0);
            }else{
                points_sentence=n_points_sentence;
                sentence_now=read_sentence();
            }
        }
        points_sentence[amount_of_sentence]=sentence_now;
        amount_of_sentence++;
    }while((sentence_now->sentence[wcslen(sentence_now->sentence)-
1]!=L'\n')&&(sentence_now->sentence[wcslen(sentence_now->sentence)-
2]!=L'\n'));
    struct Text text;
    text.pointers_sentence=points_sentence;
    text.number_of_sentence=amount_of_sentence;
    return text;

}

void words(struct Text *text){
    for (int i=0; i<text->number_of_sentence; i++){
        int size_word=MASSIV;
        int amount_of_words=0;
        struct Word** words_points= malloc(size_word*sizeof(struct
Word*));
        if (words_points==NULL){
            wprintf(L"Error: Ошибка перераспределения памяти.\n");
            free(words_points);
            exit(0);
        }
        wchar_t* sentence = wcsdup(text->pointers_sentence[i]->sentence);
        wchar_t* token=wcstok(sentence, L" ,.", &sentence);
        while (token!=NULL){
            if (amount_of_words>=size_word){
                size_word+=STEP;
                struct Word** n_words_points= realloc(words_points,
size_word*sizeof(struct Word*));
                if (n_words_points==NULL){
                    wprintf(L"Error: Ошибка перераспределения
памяти.\n");
                    free(words_points);
                    exit(0);
                }else{
                    words_points=n_words_points;
                }
            }
```

```
            struct Word* word = malloc(sizeof(struct Word));
            word->word=wcsdup(token);
            word->len_of_word=wcslen(token);
    words_points[amount_of_words]=word;
            amount_of_words++;
            token=wcstok(NULL, L" ,.", &sentence);
        }
        text->pointers_sentence[i]->pointers_word=words_points;
        text->pointers_sentence[i]->number_of_words=amount_of_words;
        free(sentence);
    }
}

void print_Text(struct Text* text){
    for (int i=0; i<text->number_of_sentence; i++){
        wprintf(L"%ls\n", text->pointers_sentence[i]->sentence);
    }
}

void free_Word(struct Word *word) {
    free(word->word);
    free(word);
}

void free_Sentence(struct Sentence *sentence) {
    for (int i = 0; i < sentence->number_of_words; i++) {
        free_Word(sentence->pointers_word[i]);
    }
    free(sentence->pointers_word);
    free(sentence->sentence);
    free(sentence);
}

void free_Text(struct Text *text) {
    for (int i = 0; i < text->number_of_sentence; i++) {
        free_Sentence(text->pointers_sentence[i]);
    }
    free(text->pointers_sentence);
}
```

Название файла: Makefile

```
all: cw

cw: cw.o actions_with_sentences.o main_functions.o stuctures_discribe.o
     gcc cw.o actions_with_sentences.o main_functions.o
stuctures_discribe.o -o cw

cw.o: cw.c actions_with_sentences.h main_functions.h
structures_discribe.h
     gcc -c cw.c

actions_with_sentences.o: actions_with_sentences.c
actions_with_sentences.h
     gcc -c actions_with_sentences.c

main_functions.o: main_functions.c main_functions.h
     gcc -c main_functions.c
```

```
stuctures_discribe.o: stuctures_discribe.c structures_discribe.h
	gcc -c stuctures_discribe.c

clean:
	rm -f *.o cw
```