

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

КУРСОВАЯ РАБОТА
по дисциплине «Программирование»
Тема: Обработка теста на языке Си

Студент гр. 0382

Диденко Д.В.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Диденко Д.В.

Группа 0382

Тема работы: Обработка текста на языке Си

Исходные данные:

Программе на вход подается текст (текст представляет собой предложения, разделенные точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских или кириллических букв, цифр и других символов кроме точки, пробела или запятой) Длина текста и каждого предложения заранее не известна.

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 24.11.2020

Дата сдачи реферата: 20.12.2020

Дата защиты реферата: 22.12.2020

Студент

Диденко Д.В.

Преподаватель

Жангиров Т.Р.

АННОТАЦИЯ

В курсовой работе была реализована обработка текста произвольной длины, для этого были использованы структуры и динамические массивы. Также в работе активно использовались стандартные библиотеки языка Си, содержащиеся в них функции и типы данных. В программе реализован элементарный интерфейс общения с пользователем и выполнение запрашиваемых им действий. При некорректном выборе действия, пользователю предлагается выбрать другое действие. Для сборки программы используется Makefile.

СОДЕРЖАНИЕ

Введение.	5
1. Цель и задание работы.	6
1.1. Цель работы.	6
1.2. Задание работы.	6
2. Ход реализации работы.	8
2.1. Объявление структур.	8
2.2. Ввод и хранение информации.	9
2.3. Решение задачи 1.	11
2.4. Решение задачи 2.	12
2.5. Решение задачи 3.	12
2.6. Решение задачи 4.	12
2.7. Вывод результата обработки.	13
2.8. Создание Makefile.	14
3. Тестирование.	15
Заключение.	19
Список использованных источников.	20
Приложение А. Исходный код программы.	21

ВВЕДЕНИЕ

Требуется создать программу, производящую выбранную пользователем обработку данных. Реализация программы должна содержать работу со структурами (для хранения текста и отдельных предложений, а также дополнительных данных), работу с динамически выделенной памятью и использование стандартных библиотек, в том числе для работы с национальным алфавитом (`wchar.h` , `wctype.h`).

Программа реализована для операционных систем на базе Linux. Разработка велась на базе операционной системы Ubuntu Linux в IDE Clion. Компиляция и линковка производилась с помощью Makefile.

В результате была создана программа, считывающая вводимый пользователем с консоли текст, выводящая меню со списком доступных функций и выполняющая выбранные. При выборе каждого действия на консоль выводится соответствующий результат обработки. Также производятся действия по очистке динамически выделенной памяти и корректному завершению работы.

1. ЦЕЛЬ И ЗАДАНИЕ РАБОТЫ

1.1. Цель работы.

Цель работы: создать программу, производящую выбранную пользователем обработку данных.

Для достижения поставленной цели требуется решить следующие задачи:

- Реализовать ввод и хранение текста.
- Реализовать функции, решающие обозначенные задачи.
- Создать Makefile.

1.2. Задание курсовой работы.

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):

Для каждого предложения вывести строку образец удовлетворяющую каждому слову в предложении. Строка условия содержит: символы, ? - 1 или больше любых символов, в начале и конце образца могут быть символы * -

обозначающие 0 или больше символов. Например, для слов “Аристотель” и “Артишок”, строка образец будет иметь вид “Ар???о?*”.

Удалить все предложения, в которых нет заглавных букв в начале слова.

Отсортировать слова в предложении по количеству гласных букв в слове.

Для каждого предложения вывести количество одинаковых слов в строке.

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.

2. РЕАЛИЗАЦИЯ РАБОТЫ.

2.1. Объявление структур.

Для хранения текста объявлены анонимные структуры Sentence, Text и Word.

Поля структуры Word:

- `wchar_t* word` – указатель на слово.
- `int size_word` - длина слова.
- `int count_vowels` - количество гласных в слове.
- `int up_let` - начинается ли слово с большой буквы
- `int count_equal_words` - количество одинаковых слов в предложении.
- `int is_first_rep` - повторяемое ли слово.

Поля структуры Sentence:

- `wchar_t* sent` - указатель на начало предложения.
- `wchar_t* copy_sent` – копия `sent` для выделения слов.
- `Word* words` – указатель на массив содержащий элементы типа Word.
- `int size_sent` - размер предложения.
- `int index_sent` - индекс предложения в тексте.
- `int ind_stop` – будет ли предложение последним в тексте.
- `int is_begin` - начало ли предложения.
- `int spaces` - количество подряд идущих пробелов между словами.
- `int count_words` - количество слов в предложении.
- `wchar_t** masks` – указатель на массив указателей на маски слов.
- `int count_mask` - количество масок.

Поля структуры Text:

- `Sentence* text` - указатель на массив содержащий элементы типа `Sentence`.
- `int count_sent` - количество предложений в тексте.

2.2. Ввод и хранение текста.

Весь текст хранится в переменной `text` типа `Text`. Переменная `text` представляет собой массив предложений.

Для ввода использовались функции `get_text()`, `get_sent()` и вспомогательные к ним – `get_words()`, `get_proper()`, `correcting_sent()` и `repeate()`.

Функция `get_text()`.

Получает на вход указатель на структуру `Text`. Ничего не возвращает.

В теле функции полю `text` структуры выделяется динамически память с помощью `malloc()`, а при недостатке памяти – добавляется с помощью `realloc()`.

Функция `get_sent()` записывает предложение в `text`, затем следует проверка, встречалось ли предложение ранее в тексте с помощью функции `repeate()`, сравнение происходит без учета регистра. Если предложение встречалось, то память, выделенная в `get_sent()` под `sent`, освобождается и предложение не записывается в `text`, как и состоящее только из точки(проверка в следующей строке). Иначе – записывается и передается в функцию `get_words()`, где предложение разбивается на слова. Далее предложению присваивается индекс и увеличивается количество предложений. Ввод завершается, как только пользователь использует перенос строки дважды. При этом, если в предложении до двух переносов строк были символы, оно не запишется, т.к. считается незавершенным (нет точки).

Функция `get_sent()`.

Получает на вход указатель на структуру `Sentence`. Ничего не возвращает.

В теле функции полю `sent` структуры выделяется динамически память с помощью `malloc()`, а при недостатке памяти – добавляется с помощью `realloc()`.

Присваиваются начальные значения полям `space`, `ind_stop`, `is_begin`. Каждый новый символ предложения получается с помощью функции `getwchar()`. Затем с помощью функции `correcting_sent()`, которая корректирует предложение до определенного вида - из предложения удаляются все незначащие пробелы (в начале предложения, до и после запятой, в конце предложения, лишние пробелы между словами) и проверяет наличие двух переносов строки. Если они находятся, ввод прекращается. После окончания предложения (`.`), в конец записывается `'\0'`. Предложение не может начинаться с запятой.

Функция `get_word()`.

Получает на вход указатель на `Sentence`. Ничего не возвращает.

В теле функции полю `words` выделяется динамически память. Поле `copy_sent` копируется из `sent` с помощью функции `wscpy()`. Затем `copy_sent` разбивается на слова с помощью функции `wcstok`. Каждый элемент поля `words` - указатель на начало слова из `copy_sent`. Каждому слову присваиваются необходимые для задач свойства в функции `get_proper()`. Количество слов записывается в поле `count_words`.

Функция `get_proper()`.

Получает на вход указатель на структуру `Word`. Ничего не возвращает.

Объявляется массив `vovels`, содержащий все гласные буквы английского и русского алфавита в двух регистрах. С помощью него считается количество гласных букв в слове и записывается в поле `count_vovels`. С помощью функции `iswupper()` проверяется, начинается ли слово с прописной буквы, если начинается, то полю `up_let` присваивается 1, иначе - 0. Задаются значения полей `size_word`, `is_first_rep`, `count_equal_words`.

Функция `repeate()`.

Получает на вход указатель на структуру `Text` и указатель на структуру `Sentence`. Возвращает 0, если последняя не нашлась в тексте, и 1, если нашлась. Проверка производится с помощью функции `wscascmp()` без учета регистра.

2.3. Решение задачи 1.

Для решения этой задачи используется функция `task_1()`, принимающая на вход указатель на структуру `Sentence`.

Для наглядности объяснения работы алгоритма решения задачи возьмем в качестве примера предложение : араса прекрас красивый.

Необходимо сравнить каждый символ одного элемента с каждым символом другого элемента. Для этого мы каждый раз сдвигаем одно слово вправо относительно другого, начиная с последнего символа:

прекрас
араса —→

В этом случае мы сравниваем «п» с «а» и получаем единственную возможную маску - `*?*`. (По условию, `*` - 0 или больше символов, `?` - 1 или больше символов). Маски сохраняются в переменную `tmp`, которая переопределяется при каждом сдвиге и обеспечивается памятью. Маски нужно сохранять для последующего использования, для это мы объявляем указатель на указатель на `wchar_t mask_2` и динамически выделяем и добавляем, когда нужно, память. Но эту маску сохранять будет лишним, добавляются только маски со значащими символами (не «`?`» и не «`*`»). Количество значащих символов считает функция `count()`. Пропустим 3 незначащих цикла:

прекрас
араса —→

В этом случае мы получим маску `*?p???*`, которую следует добавить в `mask_2`. Последний случай будет сравнивать последний элемент верхнего слова и первый нижнего.

Далее верхнее слово меняется на следующее, но сравнивать стоит с имеющими масками. Для этого мы копируем все элементы `mask_2` в `mask_1`, и каждый элемент `mask_1` сравниваем со словом, записывая непустые маски в `mask_2`. В итоге получаем список из всех вероятных масок, из которого выбираем те, в которых больше всего значащих символов и их длина больше остальных. Полученный список и его длину записываем в поля `masks` и `count_mask` соответственно переданной структуры.

2.4. Решение задачи 2.

Задача решается в функции `task_2()`, принимающей указатель на `Text`. Функция ничего не возвращает. Задача требует удалить из текста те предложения, в которых хотя бы одно слово начинается с маленькой буквы. Функция считает количество слов, начинающихся с большой буквы, суммируя поля `up_let` всех слов предложения, если эта сумма равна количеству слов в предложении, то его удалять не нужно. Если сумма меньше, то освобождается память, занятая этим предложением, и ячейке присваивается адрес следующего предложения, при этом индекс следующего предложения уменьшается на 1, т.к. оно становится на место предыдущего, если же это предложение последнее, то этапа с присвоением следующего адреса нет.

2.4. Решение задачи 3.

Используется функция `task_3()`, которая принимает на вход указатель на структуру `Text`. Ничего не возвращает.

В теле функции с помощью функции `qsort()`, сортируется по возрастанию количества гласных поле `words` текущего предложения (обработка происходит в цикле). Функция `qsort()` использует функцию-компаратор `compare()`, которая принимает на вход два указателя на элемент типа `Word` и возвращает результат сравнения их полей `count_vowels`. Таким образом, в массиве `words` (который является полем предложения) теперь слова расположены в порядке возрастания количества гласных.

2.5. Решение задачи 4.

Используется функция `task_4()`, принимающая на вход указатель на структуру `Text`. Функция ничего не возвращает.

В теле функции 3 цикла `for()`. Два из них вложены в первый. Цикл «первого» уровня двигается по индексам предложений, второго – по индексам массива слов текущего предложения, третий цикл запускается в том случае, если слово с индексом `j` ранее в предложении не встречалось (за это отвечает поле слова `is_first_rep`, равное изначально 1), третий цикл сравнивает слово с индексом `j` и все следующие за ним слова в массиве `words` с помощью функции

wscmp()). Если в предложении найдется равное слово, то его поле `is_first_rep` станет равно 0 и в цикле второго уровня оно пропустится, а поле `count_equal_words` слова с индексом `j` увеличится на 1.

2.7. Вывод результата обработки.

Вывод реализован в функции `main()`.

Для корректного вывода кириллических символов установлен `setlocale(LC_ALL, "")`. В главной функции объявляется структура `text` типа `Text` и передается в функцию `get_text()` и далее по функциям ввода.

Для удобства пользователя реализован минимальный пользовательский интерфейс с подсказками. Пользователем вводится текст после подсказки «Введите текст». Далее предлагается выбрать задачу, которую необходимо выполнить. Пользователь вводит один символ, который записывается в переменную `input`. Если введенный символ соответствует предложенным, то выполняются соответствующие команды, иначе выводится просьба ввести другой символ.

Выбор варианта обработки текста реализован с помощью условного оператора `switch()`.

В случае символа «1» выполняется первая задача, показывающая общий шаблон для всех слов показанного предложения. Шаблонов может быть несколько. Если общий шаблон не найден, то на консоль выводится «*?*», что соответствует любому возможному слову. Если же введенное предложение состоит из одного слова, на консоль выводится это же слово, т.к. оно является шаблоном к самому себе.

В случае символа «2» выполняется вторая задача, удаляющая все предложения, в которых нет заглавных букв в начале слова. На консоль выводится преобразованный текст.

В случае символа «3» выполняется третья задача, сортирующая слова предложения по неубыванию. При этом на консоль выводится отсортированный список слов предложения, но само предложение в тексте изменено не будет.

В случае символа «4» выполняется четвертая задача, показывающая количество одинаковых слов в предложении. На консоль выводится предложение и далее слово и количество его повторений. Если в предложении нет повторяющихся слов, то выводится соответствующее объявление.

После выполнения каждого задания программа запрашивает следующее действие. После ввода пользователем «5» программа завершает выполнение.

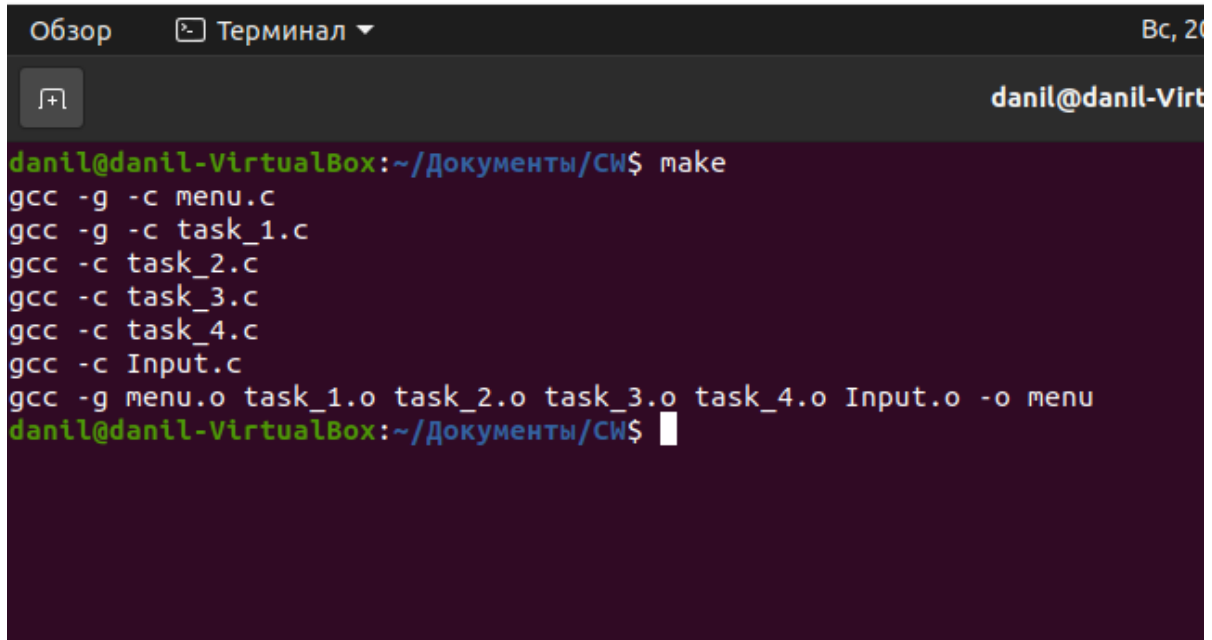
В конце функции `main()` высвобождается память.

2.8. Создание Makefile.

Функции программы поделены на несколько блоков. Для каждой задачи созданы отдельные заголовочные и с-файлы, названы `task_1.c` , `task_1.h` ; `task_2.c`, `task_2.h` и т.д. Для функций ввода создан файл `Input.c` и `Input.h`. Для функций выбора задачи и вывода результата файл `menu.c`. Объявление всех структур записано в заголовочном файле `structs.h`. В каждом заголовочном файле записан `#pragma once` для предотвращения повторного включения кода в программу. С использованием всех вышеописанных файлов создан `Makefile`, компилирующий программу `menu`, после компиляции введя `make clean` можно удалить все образованные объектные и исполняемый файлы.

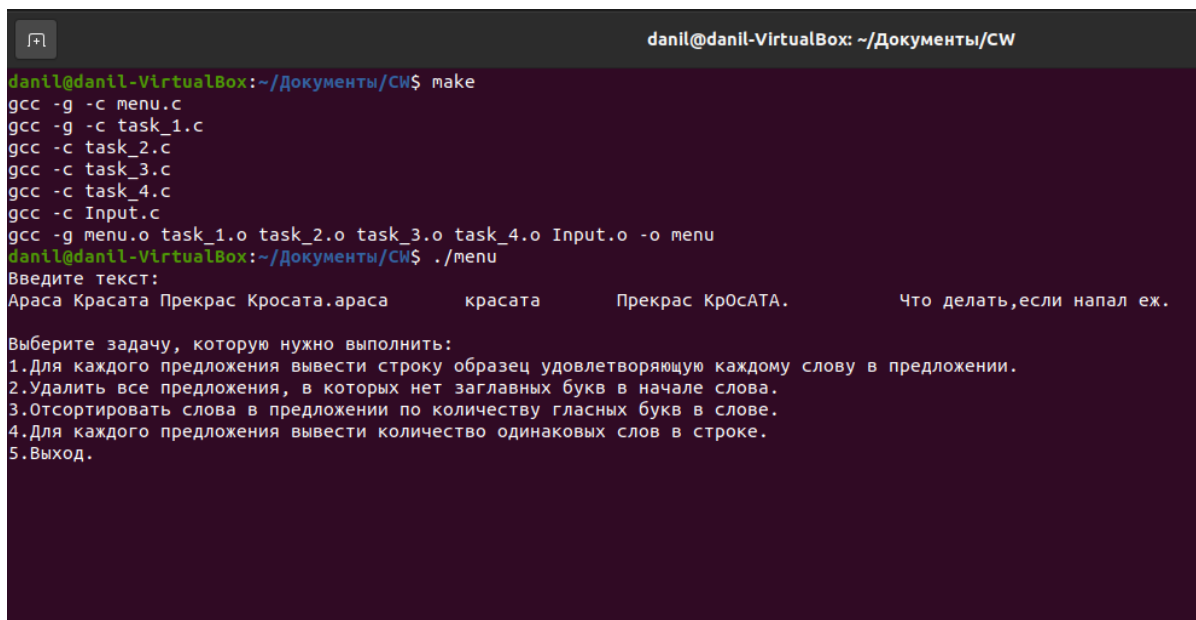
3. ТЕСТИРОВАНИЕ

1. Работа Makefile.



```
danil@danil-VirtualBox: ~/Документы/CW$ make
gcc -g -c menu.c
gcc -g -c task_1.c
gcc -c task_2.c
gcc -c task_3.c
gcc -c task_4.c
gcc -c Input.c
gcc -g menu.o task_1.o task_2.o task_3.o task_4.o Input.o -o menu
danil@danil-VirtualBox: ~/Документы/CW$
```

2. Пример работы цикла программы.



```
danil@danil-VirtualBox: ~/Документы/CW$ make
gcc -g -c menu.c
gcc -g -c task_1.c
gcc -c task_2.c
gcc -c task_3.c
gcc -c task_4.c
gcc -c Input.c
gcc -g menu.o task_1.o task_2.o task_3.o task_4.o Input.o -o menu
danil@danil-VirtualBox: ~/Документы/CW$ ./menu
Введите текст:
Араса Красата Прекрас Кросата.араса      красата      Прекрас КрОсАТА.      Что делать,если напал еж.

Выберите задачу, которую нужно выполнить:
1.Для каждого предложения вывести строку образец удовлетворяющую каждому слову в предложении.
2.Удалить все предложения, в которых нет заглавных букв в начале слова.
3.Отсортировать слова в предложении по количеству гласных букв в слове.
4.Для каждого предложения вывести количество одинаковых слов в строке.
5.Выход.
```

2.1. Пример работы 1 задачи.

```
danil@danil-VirtualBox: ~/Документы/CW$ make
gcc -g -c menu.c
gcc -g -c task_1.c
gcc -c task_2.c
gcc -c task_3.c
gcc -c task_4.c
gcc -c Input.c
gcc -g menu.o task_1.o task_2.o task_3.o task_4.o Input.o -o menu
danil@danil-VirtualBox:~/Документы/CW$ ./menu
Введите текст:
Араса Красата Прекрас Кросата.араса      красата      Прекрас КрОсАТА.      Что делать,если напал еж.

Выберите задачу, которую нужно выполнить:
1.Для каждого предложения вывести строку образец удовлетворяющую каждому слову в предложении.
2.Удалить все предложения, в которых нет заглавных букв в начале слова.
3.Отсортировать слова в предложении по количеству гласных букв в слове.
4.Для каждого предложения вывести количество одинаковых слов в строке.
5.Выход.
1
Араса Красата Прекрас Кросата.
Подходящие шаблоны для слов предложения: *?p?c*
Что делать,если напал еж.
Подходящие шаблоны для слов предложения: *?*
Следующее действие:
█
```

Из ввода удаляется второе предложение, т.к. оно идентично первому(сравнение производилось регистронезависимо и без незначащих пробелов). Задача обрабатывает два предложения и выводит общий шаблон для каждого слова. Обратим внимание на шаблон второго предложения: т.е. не нашлось подходящих совпадений в символах всех слов на консоль вывелось «*?*», что удовлетворяет условию задачи.

2.2. Пример работы 3 задачи.

```
danil@danil-VirtualBox:~/Документы/CW$ make
gcc -g -c menu.c
gcc -g -c task_1.c
gcc -c task_2.c
gcc -c task_3.c
gcc -c task_4.c
gcc -c Input.c
gcc -g menu.o task_1.o task_2.o task_3.o task_4.o Input.o -o menu
danil@danil-VirtualBox:~/Документы/CW$ ./menu
Введите текст:
Араса Красата Прекрас Кросата.араса      красата      Прекрас КрОсАТА.      Что делать,если напал еж.

Выберите задачу, которую нужно выполнить:
1.Для каждого предложения вывести строку образец удовлетворяющую каждому слову в предложении.
2.Удалить все предложения, в которых нет заглавных букв в начале слова.
3.Отсортировать слова в предложении по количеству гласных букв в слове.
4.Для каждого предложения вывести количество одинаковых слов в строке.
5.Выход.
1
Араса Красата Прекрас Кросата.
Подходящие шаблоны для слов предложения: *?p?c*
Что делать,если напал еж.
Подходящие шаблоны для слов предложения: *?*
Следующее действие:
3
Прекрас Кросата Красата Араса.
еж Что напал если делать.
Следующее действие:
```


2.3. Пример работы задачи 4.

```
danil@danil-VirtualBox: ~/Документы/CW
gcc -c task_4.c
gcc -c Input.c
gcc -g menu.o task_1.o task_2.o task_3.o task_4.o Input.o -o menu
danil@danil-VirtualBox:~/Документы/CW$ ./menu
Введите текст:
Араса Красата Прекрас Кросата.араса          красата          Прекрас КрОсАТА.          Что делать,если напал еж.

Выберите задачу, которую нужно выполнить:
1.Для каждого предложения вывести строку образец удовлетворяющую каждому слову в предложении.
2.Удалить все предложения, в которых нет заглавных букв в начале слова.
3.Отсортировать слова в предложении по количеству гласных букв в слове.
4.Для каждого предложения вывести количество одинаковых слов в строке.
5.Выход.
1
Араса Красата Прекрас Кросата.
Подходящие шаблоны для слов предложения: *?p?c*
Что делать,если напал еж.
Подходящие шаблоны для слов предложения: *?*
Следующее действие:
3
Прекрас Кросата Красата Араса.
еж Что напал если делать.
Следующее действие:
4
Араса Красата Прекрас Кросата.
Повторяющихся слов нет.
Что делать,если напал еж.
Повторяющихся слов нет.
Следующее действие:
```

После задачи 3 остальные задачи работает с изначальными предложениями, что не отражается на функциональности.

В обоих предложениях не нашлось повторяющихся слов, приведем еще пример:

```
danil@danil-VirtualBox: ~/Документы/CW
danil@danil-VirtualBox:~/Документы/CW$ ./menu
Введите текст:
привет привет Привет лрпор пока Пока. Пока пока Пока привет.

Выберите задачу, которую нужно выполнить:
1.Для каждого предложения вывести строку образец удовлетворяющую каждому слову в предложении.
2.Удалить все предложения, в которых нет заглавных букв в начале слова.
3.Отсортировать слова в предложении по количеству гласных букв в слове.
4.Для каждого предложения вывести количество одинаковых слов в строке.
5.Выход.
4
привет привет Привет лрпор пока Пока.
привет: 2.
Пока пока Пока привет.
Пока: 2.
Следующее действие:
```

2.4. Пример работы задачи 2.

```
Прекрас Кросата Красата Араса.  
еж Что напал если делать.  
Следующее действие:  
4  
  
Араса Красата Прекрас Кросата.  
Повторяющихся слов нет.  
Что делать,если напал еж.  
Повторяющихся слов нет.  
Следующее действие:  
2  
  
Араса Красата Прекрас Кросата.  
Следующее действие:
```

2.5. Несоответствующий номер задания.

```
danil@danil-VirtualBox: ~/Документы/CW  
Выберите задачу, которую нужно выполнить:  
1.Для каждого предложения вывести строку образец удовлетворяющую каждому слову в предложении.  
2.Удалить все предложения, в которых нет заглавных букв в начале слова.  
3.Отсортировать слова в предложении по количеству гласных букв в слове.  
4.Для каждого предложения вывести количество одинаковых слов в строке.  
5.Выход.  
1  
Араса Красата Прекрас Кросата.  
Подходящие шаблоны для слов предложения: *?p?c*  
Что делать,если напал еж.  
Подходящие шаблоны для слов предложения: *?*  
Следующее действие:  
3  
  
Прекрас Кросата Красата Араса.  
еж Что напал если делать.  
Следующее действие:  
4  
  
Араса Красата Прекрас Кросата.  
Повторяющихся слов нет.  
Что делать,если напал еж.  
Повторяющихся слов нет.  
Следующее действие:  
2  
  
Араса Красата Прекрас Кросата.  
Следующее действие:  
6  
  
Такой задачи не существует. Выберите другое действие:
```

2.6. Выход из программы.

```
danil@danil-VirtualBox: ~/Документы/CW  
3.Отсортировать слова в предложении по количеству гласных букв в слове.  
4.Для каждого предложения вывести количество одинаковых слов в строке.  
5.Выход.  
1  
Араса Красата Прекрас Кросата.  
Подходящие шаблоны для слов предложения: *?p?c*  
Что делать,если напал еж.  
Подходящие шаблоны для слов предложения: *?*  
Следующее действие:  
3  
  
Прекрас Кросата Красата Араса.  
еж Что напал если делать.  
Следующее действие:  
4  
  
Араса Красата Прекрас Кросата.  
Повторяющихся слов нет.  
Что делать,если напал еж.  
Повторяющихся слов нет.  
Следующее действие:  
2  
  
Араса Красата Прекрас Кросата.  
Следующее действие:  
6  
  
Такой задачи не существует. Выберите другое действие:  
5  
  
До свидания!  
danil@danil-VirtualBox:~/Документы/CW$
```

ЗАКЛЮЧЕНИЕ

Разработана программа, производящая выбранную пользователем обработку данных. Реализация программы содержит работу со структурами (для хранения текста и отдельных предложений, а также дополнительных данных), работу с динамически выделенной памятью и использование стандартных библиотек, в том числе для работы с национальным алфавитом (`wchar.h` , `wctype.h`).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. <https://www.cplusplus.com/>

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ.

1. Файл menu.c

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

#include "structs.h"
#include "task_1.h"
#include "task_2.h"
#include "task_3.h"
#include "task_4.h"
#include "Input.h"

int main(){
    setlocale(LC_ALL, "");
    int i;
    Text text;
    wprintf(L"Введите текст:\n");
    get_text(&text);

    wprintf(L"Выберите задачу, которую нужно выполнить: \n"
           "1.Для каждого предложения вывести строку образец
удовлетворяющую каждому слову в предложении.\n"
           "2.Удалить все предложения, в которых нет заглавных букв в
начале слова.\n"
           "3.Отсортировать слова в предложении по количеству гласных
букв в слове.\n"
           "4.Для каждого предложения вывести количество одинаковых слов
в строке.\n"
           "5.Выход.\n");
    wchar_t input = getwchar();
    while (input != '5'){
        switch (input) {
            case '1': //общий шаблон для всех слов в предложении
                for (i = 0; i < text.count_sent; i++) {
                    task_1(&text.text[i]);
                    wprintf(L"%ls\n", text.text[i].sent);
                    wprintf(L"Подходящие шаблоны для слов предложения:
");
                }
            
```

```

        for (int p = 0; p < text.text[i].count_mask; p++) {
            wprintf(L"%ls ", text.text[i].masks[p]);
        }
        if(text.text[i].count_words == 1){
            wprintf(L"%ls",text.text[i].words[0].word);
        }else if(text.text[i].count_mask == 0){
            wprintf(L"*?*");
        }

        wprintf(L"\n");
    }
    wprintf(L"Следующее действие: ");
    break;
    case '2'://Удаляет из текста предложения , в которых слова
начинаются с маленьких букв
        task_2(&text);
        for (i = 0; i < text.count_sent; i++) {
            wprintf(L"%ls\n", text.text[i].sent);
        }
        wprintf(L"Следующее действие: ");
        break;
    case '3'://Сортирует по неубыванию слова в предложениях по
количеству гласных.
        task_3(&text);
        for (i = 0; i < text.count_sent; i++) {
            for (int j = 0; j < text.text[i].count_words-1; j++)
{
                wprintf(L"%ls ", text.text[i].words[j].word);
            }
            wprintf(L"%ls.\n",
text.text[i].words[text.text[i].count_words-1].word);
        }
        wprintf(L"Следующее действие: ");
        break;
    case '4'://Выводит количество повторяющихся слов в
предложении
        task_4(&text);
        for (i = 0; i < text.count_sent; i++) {
            int count_print = 0;
            wprintf(L"%ls\n", text.text[i].sent);
            for (int j = 0; j < text.text[i].count_words; j++) {
                if (text.text[i].words[j].is_first_rep == 1 &&
text.text[i].words[j].count_equal_words != 1) {

```

```

                                wprintf(L"%ls:                                %d.\n",
text.text[i].words[j].word, text.text[i].words[j].count_equal_words);
                                count_print++;
                                }
                                if (j == text.text[i].count_words - 1 &&
count_print == 0) {
                                wprintf(L"Повторяющихся слов нет.\n");
                                }
                                }
                                }
                                wprintf(L"Следующее действие: ");
                                break;
                                case '\n':case' ': break;
                                default:
                                wprintf(L"Такой задачи не существует. Выберите другое
действие: ");
                                }
                                input = getwchar();
                                wprintf(L"\n");
                                }
                                wprintf(L"До свидания!\n");
                                for (i = text.count_sent; i > 0;i--){
                                for(int j = 0; j < text.text[i].count_mask;j++){
                                free(text.text[i].masks[j]);
                                }
                                free(text.text[i].masks);
                                free(text.text[i].sent);
                                free(text.text[i].copy_sent);
                                free(text.text[i].words);

                                }
                                free(text.text);
                                return 0;
                                }

```

2. Файл Input.h

```
#pragma once
```

```

void get_text();
int repeate();
int correcting_sent();
void get_proper();
void get_words();
void get_sent();

```

3. Файл Input.c

```
#include <stdlib.h>
#include <locale.h>
#include <wchar.h>
#include <wctype.h>
#include "structs.h"
#include "Input.h"

int repeate(Text* text, Sentence* sent){
    for (int i = 0; i < text->count_sent; i++){
        if (!(wcscasecmp(text->text[i].sent, sent->sent))){
            return 1;
        }
    }
    return 0;
} //Используется при вводе

int correcting_sent(Sentence* sent, wchar_t* sym){
    if (*sym == '\\t'){*sym = ' ';}
    if(*sym == ' '){
        if(sent->is_begin == 1){sent->is_begin = 0; sent->spaces++; return 0;}

        else if(sent->spaces == 1){return 0;}
        if(sent->sent[sent->size_sent-1] == ','){return 0;}
        sent->spaces++;
    }else if(*sym == ','){
        if (sent->sent[sent->size_sent-1] == ' '){
            sent->sent[sent->size_sent-1] = ','; //Первым незначащим
            СИМВОЛОМ в предложении не может быть запятая
            return 0;}
    }else if(*sym == '\\n'){
        sent->spaces = 0;
        sent->ind_stop++;
        sent->is_begin = 0;
        if (sent->ind_stop == 2){return 1;}
        return 0;
    }else{
        sent->is_begin = 0;
        sent->spaces = 0;
        sent->ind_stop = 0;}
}
```



```

    sent->sent[sent->size_sent] = *sym;

    return 2;
} //Используется при вводе

void get_proper(Word* word){
    wchar_t* vowels = L"aeiouyAEIOUYауоыиэяюёеАУОЫИЭЯЮЁЕ";
    int begin_word = 1;
    word->up_let = 0;
    word->count_vowels = 0;
    int i = 0;
    while (word->word[i] != '\\0'){
        if (begin_word == 1 && iswupper(word->word[i])){
            word->up_let = 1;
            begin_word = 0;
        }else begin_word = 0;
        int j = 0;
        while (vowels[j] != '\\0'){
            if (word->word[i] == vowels[j]){
                word->count_vowels++;
                break;
            }
            j++;
        }
        i++;
    }
    word->size_word = i;
    word->is_first_rep = 1; //Используется для задачи 4, первое ли
повторяющееся слово
    word->count_equal_words = 1; //Используется для задачи 4, количество
повторяющихся слов, по начало 1 (оно само)
} //Используется при вводе

void get_words(Sentence* sent){
    sent->count_words = 0;
    Word* tmp;
    int count_words_cur = 30;
    sent->words = (Word*)malloc(count_words_cur*sizeof(Word));
    sent->copy_sent = (wchar_t*)malloc((sent->size_sent+2)*sizeof(wchar_t));
    wcsncpy(sent->copy_sent, sent->sent); //Новая строка, которую будем
резать
    wchar_t* p; //Массив, в котором храним указатели на текущее место среза
    sent->words[0].word = wcstok(sent->copy_sent, L" .", &p);

```

```

    get_proper(&(sent->words[0]));
    int i = 0;
    while(1){
        if(sent->count_words == count_words_cur-1){
            count_words_cur+=30;
            tmp = (Word*)realloc(sent->words, count_words_cur*sizeof(Word));
            sent->words = tmp;
        }
        i++;
        sent->words[i].word = wcstok(NULL, L" .", &p); //Последнее слово
        Null всегда, но в кол-во слов в предложении не входит
        if (sent->words[i].word == NULL){break;}
        get_proper(&(sent->words[i]));
    }
    sent->count_words = i;
}

void get_sent(Sentence* sent){
    wchar_t* tmp;
    int size_sent_cur = 10;
    sent->size_sent = 0;
    sent->sent = (wchar_t*)malloc(size_sent_cur*sizeof(wchar_t));
    wchar_t sym = ' ';
    sent->spaces = 0;
    sent->ind_stop = 0;
    sent->is_begin = 1;
    while(sym != '.'){
        if(sent->size_sent == size_sent_cur-2){
            size_sent_cur+=30;
            tmp = (wchar_t*)realloc(sent->sent, size_sent_cur*sizeof(wchar_t));
            sent->sent = tmp;
        }
        sym = (wchar_t)getwchar();
        int variant = correcting_sent(sent, &sym); //Функция по
        форматированию предложения.
        if (variant == 0){continue;}
        else if(variant == 1){ break;}
        else if(variant == 2){ sent->size_sent++;}
    }
    if(sent->sent[sent->size_sent-2] == ' '){sent->sent[sent->size_sent-2] = '.'; sent->size_sent--;}
    sent->sent[sent->size_sent] = '\\0';
}

```

```

} //Используется при вводе

void get_text(Text* text){
    Sentence* tmp;
    int size_text = 5;
    text->count_sent = 0;
    text->text = (Sentence*)malloc(size_text*sizeof(Sentence));
    while(1){
        if(text->count_sent == size_text-1){
            size_text+=30;
            tmp = (Sentence*)realloc(text->text,
size_text*sizeof(Sentence));
            text->text = tmp;
        }
        get_sent(&text->text[text->count_sent]); //Получаем структуру
предложения
        if (repeate(text,&text->text[text->count_sent])){
            free(text->text[text->count_sent].sent);
            continue;} //Если предложение встречалось, не добавляем его
        if (text->text[text->count_sent].size_sent == 1 && text-
>text[text->count_sent].sent[0] == '.'){
            free(text->text[text->count_sent].sent);
            continue;}
        if (text->text[text->count_sent].ind_stop == 2){break;} //Условное
завершение ввода
        get_words(&text->text[text->count_sent]); //Разбиваем предложение
на слова
        text->text[text->count_sent].index_sent = text->count_sent; //Даем
индекс предложению
        text->count_sent++;
    }
} //Используется при вводе

```

4. Файл structs.h

```
#pragma once
```

```

typedef struct{
    wchar_t* word;
    int size_word; //Размер слова
    int count_vowels; //Количество гласных
    int up_let; //Начинается ли слово с большой буквы
    int count_equal_words; //Количество одинаковых слов в предложении
    int is_first_rep; //Повторяемое ли слово
}Word;

```

```

typedef struct{
    wchar_t* sent;//Хранит указатели на строку предложения
    wchar_t* copy_sent;
    Word* words;
    int size_sent;//Размер предложения
    int index_sent;//Индекс предложения в тексте
    int ind_stop;//Когда заканчивать ввод текста
    int is_begin;//Начало ли предложения
    int spaces;//Количество подряд идущих пробелов
    int count_words;//Количество слов в предложении
    wchar_t** masks;//Шаблоны для слов
    int count_mask;//Количество шаблонов
}Sentence;

typedef struct {
    Sentence* text;//Хранит указатель на структуру предложения
    int count_sent;//Количество предложений в тексте
}Text;

```

5. Файл task_1.h

```

#pragma once

int count();
int repeate_mask();
void task_1();

```

6. Файл task_1.c

```

#include <stdlib.h>
#include <locale.h>
#include <wchar.h>
#include <wctype.h>
#include "structs.h"
#include "task_1.h"

int count(const wchar_t* tmp,int N){
    int count_al = 0;
    for (int k = 0; k < N; k++) {
        if (tmp[k] != '?' && tmp[k] != '*') {
            count_al++;
        }
    }
    return count_al;
}

```

```

int repeate_mask(wchar_t** masks, wchar_t* mask,int N){
    for (int i = 0; i < N;i++){
        if (!wcscmp(masks[i],mask)){
            return 0;
        }
    }
    return 1;
}

void task_1(Sentence * sent){
    int size_mask = 30;
    int size_mask2 = 100;
    int size_mask_word = 20;
    wchar_t** tmp2;
    wchar_t** tmp4;
    wchar_t* tmp3;
    wchar_t** mask_1 = (wchar_t**)
malloc(size_mask*sizeof(wchar_t*)); //Первичные маски
    wchar_t** mask_2 = (wchar_t**)
malloc(size_mask2*sizeof(wchar_t*)); //Готовые маски
    int count_al;
    int m;
    int mv1;
    int zi1; //Смещение 1 слова (от i)
    int zi2; //Смещение 2 слова (от j)
    int count_mask = 1;
    mask_1[0] = sent->words[0].word;

    int max_count_al = 0;
    int count_mask_2 = 0;
    for (int i = 1; i < sent->count_words;i++){
        for (int j = 0;j<count_mask;j++){
            zi1 = 0;
            zi2 = 0;
            if (mask_1[j][wcslen(mask_1[j])-1] == '*'){
                mask_1[j][wcslen(mask_1[j])-1] = '\\0';
            }
            if (mask_1[j][0] == '*'){
                for (int k = 0; k<wcslen(mask_1[j]);k++){
                    mask_1[j][k] = mask_1[j][k+1];
                }
            }
            if(sent->words[i].size_word >= wcslen(mask_1[j])){

```

```

        while (zi2 < wcslen(mask_1[j])) {
            if (count_mask_2 >= size_mask2 -1){
                size_mask2+=500;
                tmp4 =
(wchar_t**)realloc(mask_2,size_mask2*sizeof(wchar_t*));
                mask_2 = tmp4;
            }
            wchar_t *tmp = (wchar_t *) malloc(size_mask_word *
sizeof(wchar_t));
            if (zil < sent->words[i].size_word-1) { //Эта строка
гарантирует, что нулевые элементы сравниваются в другом условии
                m = 1;
                mv1 = 0;
                tmp[0] = '*';
                while (m <= zil+1 && m < wcslen(mask_1[j])+1) {
                    if(m == size_mask_word-2){
                        size_mask_word+=15;
                        tmp3 =
(wchar_t*)realloc(tmp,size_mask_word*sizeof(wchar_t));
                        tmp = tmp3;
                    }
                    if (mask_1[j][m-1] ==
                        sent->words[i].word[sent-
>words[i].size_word - 1 - zil + mv1]) {
                        tmp[m] = mask_1[j][m-1];
                    } else { tmp[m] = '?'; }
                    m++;
                    mv1++;
                }
                tmp[m] = '*';
                tmp[m+1] = '\\0';
                count_al = count(tmp,m);
                //wprintf(L"%ls ", tmp);
                if (count_al != 0 && repeate_mask(mask_2,tmp,
count_mask_2)) {
                    mask_2[count_mask_2] = tmp;
                    count_mask_2++;
                }

                zil++;
            } else {
                m = 0;
                mv1 = 0;
                tmp[0] = '*';

```

```

        m++;
        while (zi2 + m-1 < wcslen(mask_1[j]) && zi2 + m-1
< sent->words[i].size_word) {
            if(m == size_mask_word-2){
                size_mask_word+=15;
                tmp3 =
(wchar_t*)realloc(tmp,size_mask_word*sizeof(wchar_t));
                tmp = tmp3;
            }
            if (mask_1[j][zi2 + m-1] == sent-
>words[i].word[m-1]) {
                tmp[m] = sent->words[i].word[m-1];
            } else { tmp[m] = '?'; }
            m++;
            mv1++;
        }

        tmp[m] = '*';
        tmp[m+1] = '\\0';
        count_al = count(tmp,m);
        //wprintf(L"%ls ", tmp);
        if (count_al != 0 && repeate_mask(mask_2,tmp,
count_mask_2)) {
            mask_2[count_mask_2] = tmp;
            count_mask_2++;
        }

        zi2++;
    }
}
}else{
    while (zi2 < sent->words[i].size_word) {
        if (count_mask_2 >= size_mask2 -1){
            size_mask2+=50;
            tmp4 =
(wchar_t**)realloc(mask_2,size_mask2*sizeof(wchar_t));
            mask_2 = tmp4;
        }
        wchar_t *tmp = (wchar_t *) malloc(size_mask_word *
sizeof(wchar_t));
        if (zi1 < wcslen(mask_1[j])-1){
            m = 1;
            mv1 = 0;
            tmp[0] = '*';

```

```

        while (m-1 <= zil && m-1 < sent-
>words[i].size_word) {
            if(m == size_mask_word-2) {
                size_mask_word+=15;
                tmp3 =
(wchar_t*)realloc(tmp,size_mask_word*sizeof(wchar_t));
                tmp = tmp3;
            }
            if (sent->words[i].word[m-1] ==
mask_1[j][wcslen(mask_1[j])-1-zil+mv1]) {
                tmp[m] = mask_1[j][wcslen(mask_1[j])-1-
zil+mv1];

            } else { tmp[m] = '?'; }
            m++;
            mv1++;
        }
        tmp[m] = '*';
        tmp[m+1] = '\\0';
        count_al = count(tmp,m);
        //wprintf(L"%ls ", tmp);
        if (count_al != 0 && repeate_mask(mask_2,tmp,
count_mask_2)) {
            mask_2[count_mask_2] = tmp;
            count_mask_2++;
        }

        zil++;
    } else {
        m = 1;
        mv1 = 0;
        tmp[0] = '*';
        while (zi2 + m-1 < sent->words[i].size_word &&
zi2 + m-1 < wcslen(mask_1[j])) {
            if(m == size_mask_word-2) {
                size_mask_word+=15;
                tmp3 =
(wchar_t*)realloc(tmp,size_mask_word*sizeof(wchar_t));
                tmp = tmp3;
            }
            if (sent->words[i].word[zi2 + m-1] ==
mask_1[j][m-1]) {
                tmp[m] = sent->words[i].word[zi2+m-1];
            } else { tmp[m] = '?'; }
            m++;

```



```

    }
    tmp[m]='*';
    tmp[m+1] = '\\0';

    count_al = count(tmp,m);
    //wprintf(L"%ls %d,", tmp,count_al);
    if (count_al != 0 && repeate_mask(mask_2,tmp,
count_mask_2)) {

        mask_2[count_mask_2] = tmp;
        count_mask_2++;
    }

    zi2++;
}
}
}
for(int p = 0;p < count_mask_2;p++){
    if(p >= size_mask-2){
        size_mask+=20;
        tmp2 =
(wchar_t**)realloc(mask_1,size_mask*sizeof(wchar_t*));
        mask_1 = tmp2;
    }
    mask_1[p] = mask_2[p];
}
count_mask = count_mask_2;
count_mask_2 = 0;
}

int max_size_mask = 0;
int rec = 0;
int count_ready_mask = 0;
wchar_t** rec_masks = (wchar_t**)malloc(count_mask*sizeof(wchar_t*));
for(int q = 0 ;q < count_mask;q++){
    if(count(mask_1[q],(int)wcslen(mask_1[q])) >= max_count_al){
        max_count_al = count(mask_1[q],(int)wcslen(mask_1[q]));
        max_size_mask = (int)wcslen(mask_1[q]);
    }
}

for (int q = 0; q < count_mask;q++){
    if (count(mask_1[q],(int)wcslen(mask_1[q])) == max_count_al){

```

```

        rec_masks[rec] = mask_1[q];
        if (max_size_mask < wcslen(rec_masks[rec])){
            max_size_mask = (int)wcslen(rec_masks[rec]);
        }
        rec++;
    }else{
        free(mask_1[q]);
    }
}

wchar_t** ready_masks = (wchar_t**)malloc(rec*sizeof(wchar_t*));
for (int q = 0; q < rec;q++){
    if(wcslen(rec_masks[q]) == max_size_mask){
        ready_masks[count_ready_mask] = rec_masks[q];
        count_ready_mask++;
    }
}
sent->masks = ready_masks;
sent->count_mask = count_ready_mask;
}

```

7. Файл task_2.h

```
#pragma once
```

```
void task_2();
```

8. Файл task_2.c

```

#include <stdlib.h>
#include <locale.h>
#include <wchar.h>
#include <wctype.h>
#include "structs.h"
#include "task_2.h"

void task_2(Text* text){
    int words_up_let;
    int i = 0;
    while (i < text->count_sent){
        words_up_let = 0;
        for (int j = 0; j < text->text[i].count_words; j++){
            if (text->text[i].words[j].up_let){
                words_up_let++;
            }
        }
    }
}

```

```

        }
    }

    if(words_up_let != text->text[i].count_words){
        if(i == text->count_sent-1){
            text->count_sent--;
            break;
        }
        for(int j = 0; j < text->text[i].count_mask;j++){
            free(text->text[i].masks[j]);
        }
        free(text->text[i].masks);
        free(text->text[i].sent);
        free(text->text[i].words);
        if(i != text->count_sent-1){
            for (int k = i;k < text->count_sent-1;k++){
                text->text[k+1].index_sent = text->text[k].index_sent;
                text->text[k] = text->text[k+1];
            }
        }
        text->count_sent--;
        i--;
    }
    i++;
}
}

```

9. Файл task_3.h

```
#pragma once
```

```
int compare();
```

```
void task_3();
```

10. Файл task_3.c

```
#include <stdlib.h>
```

```
#include <locale.h>
```

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

```
#include <wctype.h>
```

```
#include "structs.h"
```

```
#include "task_3.h"
```

```
int compare(Word* el1,Word* el2){
```

```

        return el1->count_vowels >= el2->count_vowels;
    }
    void task_3(Text* text) {
        for (int i = 0; i < text->count_sent; i++) {
            qsort(text->text[i].words, text->text[i].count_words,
sizeof(Word), (int (*)(const void *, const void *)) compare);
        }
    }
}

```

11. Файл task_4.h

```

#pragma once
void task_4();

```

12. Файл task_4.c

```

#include <stdlib.h>
#include <locale.h>
#include <wchar.h>
#include <wctype.h>
#include "structs.h"
#include "task_4.h"

void task_4(Text* text){
    for(int i = 0; i < text->count_sent; i++){
        for(int j = 0; j < text->text[i].count_words; j++){
            if (text->text[i].words[j].is_first_rep){//Если таких слов не
было, то ищем, иначе нет
                for (int k = j+1; k < text->text[i].count_words; k++){
                    if(!wcscmp(text->text[i].words[j].word, text-
>text[i].words[k].word)){
                        text->text[i].words[k].is_first_rep =
0;//Найденному слову присваиваем значение, что оно уже встречалось в
предложении
                        text-
>text[i].words[j].count_equal_words++;//Увеличиваем кол-во одинаковых
слов
                    }
                }
            }
        }
    }
}

```