

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

Дешура Д.В.

Преподаватель

Чайка К.В.

Санкт-Петербург

2021

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Дешура Д.В.

Группа 1304

Тема работы: Обработка текстовых данных

Исходные данные:

Консольное приложение на языке Си, считывающее текст произвольной длины, сохраняющее предложения в виде структур, удаляющее дублирующиеся предложения, поддерживающее как кириллические, так и латинские символы.

Содержание пояснительной записки:

«Содержание», «Введение», «Разработка кода», «Сборка программы», «Тестирование», «Инструкция», «Заключение», «Список использованных источников», «Приложение А. Программный код».

Предполагаемый объем пояснительной записки:

Не менее 12 страниц.

Дата выдачи задания: 15.12.2021

Дата сдачи реферата: 14.12.2021

Дата защиты реферата: 16.12.2021

Студент

Дешура Д.В.

Преподаватель

Чайка К.В.

АННОТАЦИЯ

Разработано приложение, считывающее текст произвольной длины, разбивающее его на предложения и сохраняющее их в структуры предложений и текста, память для хранения выделяется динамически. Программа удаляет повторяющиеся предложения и обрабатывает их одним из 4 предложенных способов. Благодаря использованию широких символов поддерживаются кириллические символы. Используются функции стандартной библиотеки Си.

SUMMARY

An application that reads text of arbitrary length, splits it into sentences and stores them in sentence and text structures has been developed; storage memory is dynamically allocated. The program removes duplicate sentences and processes them in one of the 4 suggested ways. Cyrillic characters are supported by using wide characters. The functions of the C standard library are used.

Рис 3.

```

dmitriy@dmitriy-Virtual-Machine: ~/Рабочий стол/Учёба/Программирование/Курсовая/1
5. Для вывода текущего состояния текста.
Для завершения программы введите любую другую цифру
2
выполняю команду 2
Для продолжения введите номер команды от 1 до 5:
1. В каждом предложении заменить первое слово на второе слово из предыдущего
рвого предложения, второе слово надо брать из последнего.
2. Отсортировать предложения по длине третьего слова. Если слов меньше трех,
лова равняется нулю.
3. Вывести на экран все предложения, в которых в середине слова встречаются
нужно выделить зеленым цветом.
4. В каждом предложении, в слове, все символы, которые встречаются несколько
одним таким символом.
5. Для вывода текущего состояния текста.
Для завершения программы введите любую другую цифру
5
дед.
words: 1
ааааааа.
words: 1
sssss.
words: 1
д1ед репку из земли тянуть: тянет-потянет, вытянуть не может.
words: 9

```

Рис 4.

```

dmitriy@dmitriy-Virtual-Machine: ~/Рабочий стол/Учёба/Программирование/Курсовая/1
внучка за внучк23у, внучка за бабу, бабка за дедку, дедка за репку: тянут-потянут,
words: 16
за кошка мышккккку.
words: 3
Для продолжения введите номер команды от 1 до 5:
1. В каждом предложении заменить первое слово на второе слово из предыдущего
рвого предложения, второе слово надо брать из последнего.
2. Отсортировать предложения по длине третьего слова. Если слов меньше трех,
лова равняется нулю.
3. Вывести на экран все предложения, в которых в середине слова встречаются
нужно выделить зеленым цветом.
4. В каждом предложении, в слове, все символы, которые встречаются несколько
одним таким символом.
5. Для вывода текущего состояния текста.
Для завершения программы введите любую другую цифру
3
выполняю команду 3
д1ед репку из земли тянуть: тянет-потянет, вытянуть не может.
д1ед дед на помощь бабу.
кошка ааааааа за кошку, кошка за Жучку, Жучка за внучку, 34внучка за 63534абку467, 3
ку57, дедк876а за репку тянут-потянуттттт – вытянули репку, sgsgdf sdghdsj dfkgsdkjl
fhs sdhfg, dfhsh, sdfhs, dfh dfsh jg34внучка за 63534абку467, 3456а3456ка3 з5а3 дедк
ку тянут-потянуттттт – вытянули репку.
внучка за внучк23у, внучка за бабу, бабка за дедку, дедка за репку: тянут-потянут,
Для продолжения введите номер команды от 1 до 5:
1. В каждом предложении заменить первое слово на второе слово из предыдущего
рвого предложения, второе слово надо брать из последнего.

```


Рис 5.

```
dmitriy@dmitriy-Virtual-Machine: ~/Рабочий стол/Учёба/Программирование/Курсовая/1
5. Для вывода текущего состояния текста.
Для завершения программы введите любую другую цифру
4
Выполняю команду 4
Для продолжения введите номер команды от 1 до 5:
1. В каждом предложении заменить первое слово на второе слово из предыдущего
   первого предложения, второе слово надо брать из последнего.
2. Отсортировать предложения по длине третьего слова. Если слов меньше трех,
   слова равняется нулю.
3. Вывести на экран все предложения, в которых в середине слова встречаются
   нужно выделить зеленым цветом.
4. В каждом предложении, в слове, все символы, которые встречаются несколько
   одним таким символом.
5. Для вывода текущего состояния текста.
Для завершения программы введите любую другую цифру
5
дед.
words: 1
а.
words: 1
с.
words: 1
дядя репку из земли тянуть: тянет-потянет, вытянуть не может.
words: 9
```

Рис 6.

```
dmitriy@dmitriy-Virtual-Machine: ~/Рабочий стол/Учёба/Программирование/Курсовая/1
words: 10
бабка за бабу, бабу за деду, дедка за репку: тянут-потянут, вытянуть не могут.
words: 13
за внука Жучку.
words: 3
за Жучка кошку.
words: 3
Жучка за Жучку, Жучка за внука, внука за бабу, бабу за деду, дедка за репку: тя
ь не могут.
words: 19
за бабу внука.
words: 3
внука за внучку, внука за бабу, бабу за деду, дедка за репку: тянут-потянут,
words: 16
за кошка мышку.
words: 3
Для продолжения введите номер команды от 1 до 5:
1. В каждом предложении заменить первое слово на второе слово из предыдущего
   первого предложения, второе слово надо брать из последнего.
2. Отсортировать предложения по длине третьего слова. Если слов меньше трех,
   слова равняется нулю.
```

СОДЕРЖАНИЕ

Введение	8
1. Разработка кода.	9
1.1. Техническое задание.	9
1.2. Выделение подзадач.	10
1.3. Чтение и запись текста.	11
1.4. Функции обработки текста.	12
1.5. Функция <code>main()</code> .	14
2. Сборка программы.	15
3. Тестирование.	16
4. Инструкция	19
Заключение	20
Список использованных источников	21
Приложение А. Программный код.	22

ВВЕДЕНИЕ

Целью этой работы является изучение стандартных функций языка СИ для работы с текстом и создание приложения обрабатывающего введенный текст определенным способом.

Для достижения поставленной цели требуется решить следующие задачи:

1. Изучение функций стандартной библиотеки для работы с текстом.
2. Изучение функций позволяющих работать с типом данных `wchar_t`.
3. Разработка кода программы.
4. Сборка программы, написание заголовочных файлов и `Makefile`.
5. Отладка программы.
6. Тестирование.
7. Разработка пользовательской инструкции.

1. РАЗРАБОТКА КОДА

1.1. Техническое задание.

Вариант 4

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. В каждом предложении заменить первое слово на второе слово из предыдущего предложения. Для первого предложения, второе слово надо брать из последнего.
2. Отсортировать предложения по длине третьего слова. Если слов меньше трех, то длина третьего слова равняется нулю.
3. Вывести на экран все предложения, в которых в середине слова встречаются цифры. Данные слова нужно выделить зеленым цветом.
4. В каждом предложении, в слове, все символы, которые встречаются несколько раз подряд заменить одним таким символом.

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. Выделение подзадач.

Таким образом, мы можем выделить следующие подзадачи:

- Считывание текста;
- Разбиение текста на отдельные предложения и запись их в структуры;
- Удаление повторяющихся предложений;
- “Бесконечный” вывод подсказки и запрос команды;
- Реализация команды 1 для замены первого слова предложения на второе слово в предыдущем (для 1го предложения берём слово из последнего предложения);
- Реализация вспомогательной функции для извлечения второго слова;
- Реализация команды 2 для сортировки предложений по длине 3-го слова;
- Реализация вспомогательной функции `compare()`, для функции `qsort()`;
- Реализация команды 3, выводящей все предложения внутри которых есть слова с цифрами внутри и раскрашивающей эти слова в зелёный цвет;
- Отдельная функция, проверяющая содержит ли слово цифры внутри;
- Реализация команды 4 для замены всех повторяющихся символов в словах;
- Реализация команды 5 для вывода текущего состояния текста

Отсюда мы можем сгруппировать наши подзадачи в 3 файла:

1. Считывание текста в файл `inputtext.c`;
2. Обработка текста в файл `formattext.c`;
3. Структура программы и дополнительные действия в файле `main.c`.

1.3. Запись и хранение текста

Поскольку мы не знаем заранее насколько большим может быть текст, мы можем реализовать его считывание при помощи динамически выделяемого массива строк длиной 100 символов. В функции `input_text()` мы в цикле расширяем массив строк на ещё один указатель, выделяем строки длиной 100 элементов `wchar_t` и заполняем их при помощи функции `fgetws()` и записываем в массив указатель на новую введенную строку.

Для хранения предложений и текста реализуем структуры `struct Sentence`, состоящую из `str` (указатель на `char`) и `words_number` (целое число – количество слов в предложении), а также `struct Text`, состоящую из `sentences` (массива `struct Sentence`) и `sents_number` (целое число - количество строк).

При помощи функции `make_sentences()` мы разбиваем получившийся массив на отдельные предложения и записываем их в структуру. Идея в том, чтобы, обрабатывая одну строку за другой, при помощи функции `wcsstr()` находить в них точки и извлекать срез начиная с элемента “предыдущая точка + 2” – с первого символа нового предложения (либо с 0-го элемента, если это начало строки), заканчивая элементом точки (либо конца строки, если точка не была найдена), затем записывать указатель на копию строки.

Последним шагом в рамках этой группы задач является удаление повторяющихся предложений (необходимо сравнивать их без учёта регистра). За это отвечает функция `delete_repeated_sentences()`. Суть заключается в том, чтобы при помощи вложенных циклов проверить предложения со 2го предложения до последнего, не встречается ли их эквивалент в предыдущих предложениях. Для сравнения строк сначала вспомогательная функция `wstr_tolower()` приводит нужные нам предложения к нижнему регистру, а затем при помощи функции `wscmp()` сравниваем получившиеся строки, пометая флагом дублирующиеся предложения кроме самого первого из них.

Преобразованный массив `struct Sentences` и его длина записываются в `struct Text`.

1.4. Функции обработки текста

Текст считан, разбит на предложения и записан в структуры. Теперь необходимо обработать текст согласно заданию. Для реализации 4 действий, описанных в задании, мы реализуем соответственно функции `command_1()`, `command_2()`, `command_3()` и `command_4()`.

Функция `command_1()` в каждом предложении заменяет первое слово на второе слово из предыдущего предложения (для первого предложения, второе слово берёт из последнего). Для удобства мы выделили взятие второго слова из нужного предложения в отдельную подзадачу и написали для неё отдельную функцию `get_second_word()`. В функции `get_second_word()` мы при помощи функции `wcstok()` извлекаем из копии строки второе слово, в функции `command_1()` мы также извлекаем срез текущего предложения начиная со второго слова (вместе с первым пробелом, если в предыдущем предложении было лишь одно слово), динамически выделяем память и при помощи `wscat()` объединяем строки.

Функция `command_2()` сортирует предложения по длине третьего слова (если слов меньше трех, то длину третьего слова считаем равной нулю). Для сортировки массива структур воспользуемся функцией `qsort()`, для которой напишем функцию-компаратор `compare()`. Идея последней проста, мы сравниваем структуры, (если количество слов `words_number > 2`) извлекая при помощи функции `wcstok()` третье слово в каждом предложении, получает их длину при помощи функции `wcslen()` (в случае, когда `words_number <= 2`, длина слова по умолчанию становится равной 0) и сравнивает эти длины.

Функция `command_3()` выводит на экран все предложения, в которых в середине слова встречаются цифры. Данные слова выделяются зеленым цветом. Идея заключается в том, чтобы каждое предложение сначала разбить на отдельные элементы (слова и знаки препинания ‘,’, ‘.’ хранятся в виде массива строк), при этом все элементы, отличные от точки и запятой проходят проверку на цифры внутри при помощи специальной функции `numbers_inside()`, возвращающей 1, если у слова внутри есть цифры (мы считаем, что у слова есть

цифры внутри, если в слове есть последовательность цифр, ограниченная с обеих сторон буквами. Такое слово может и начинаться и заканчиваться цифрами. (например: “1a2b3” считаем подходящим, так как цифра 2 ограничена буквами; “1ab3” – не считаем подходящим)) и возвращающей 0, если слово неподходящее. Найдя хотя бы 1 подходящее слово мы меняем значение flag на 1, после того, как мы получили массив из элементов, мы в случае, если flag = 1 выводим это предложение особым способом. Сначала проверяется первое слово и выводится нужным способом (зелёным или не зелёным цветом в зависимости от того подходящее это слово или нет), затем в цикле каждый элемент если он является точкой или запятой, просто выводится на экран, иначе проверяется на цифры внутри и выводится нужным способом с пробелом впереди (все слова начиная со второго мы выводим с впередистоящим пробелом), после вывода всех элементов выводим символ переноса строки. Операция повторяется для всех предложений.

Функция `command_4()` в каждом предложении, в слове, все символы, которые встречаются несколько раз подряд заменяет одним таким символом. Суть заключается в том, что мы проходим каждое предложение посимвольно, начиная со второго символа и, если он не равен пробелу, запятой или точке, и если он равен предыдущему символу, мы удаляем этот символ при помощи функции `wmemmove()`.

1.5. Функция `main()`.

В функции `main()` при помощи функции `setlocale()` устанавливается кириллическая локализация. Вызываются функции ввода текста и записывается `struct Text`. Затем при помощи цикла `while(1)` реализуется бесконечная работа программы. Внутри цикла программа выводит подсказку и ждёт пользовательского ввода. При помощи оператора `switch()` мы обрабатываем различные команды пользователя: 1-4 по обработке текста, 5 – для вывода текущего состояния текста, остальные цифры подразумевают завершение программы.

2. СБОРКА ПРОГРАММЫ.

Функции ввода текста, его первичной обработки сгруппированы в отдельном файле `inputtext.c`. Для этого файла написан заголовочный файл `inputtext.h` в котором содержатся объявления необходимых для работы функций из `inputtext.c`, шаблоны функций, а также объявление `struct Sentence` и `struct Text`.

Функции обработки готового текста и некоторые вспомогательные функции сгруппированы в файле `formattext.c`. Этому файлу соответствует заголовочный файл `formattext.h`, в котором содержатся объявления функций стандартной библиотеки, используемых функциями в файле `formattext.c`.

Сборка программы осуществляется при помощи утилиты `make`, для программы написан `Makefile`.

Программный код представлен в приложении А.

3. ТЕСТИРОВАНИЕ.

В таблице 1 представлены результаты тестирования. Для удобства опущены элементы интерфейса, представлен только результат работы программы.

Таблица 1. Результаты тестирования.

№	Ввод	Вывод	Комментарии
Проверка работы функции ввода текста			
1.	1234, 7891 3456 89123 56789. 345, 891 345678 1234, 78912. 567891 34567891234 678912 4567891, 45678.	1234, 7891 3456 89123 56789. words: 5 345, 891 345678 1234, 78912. words: 5 567891 34567891234 678912 4567891, 45678. words: 5	Ввели 99 символов, чтобы попасть в исключение. Программа отработала корректно, количество предложений и слов в них верное.
Проверка работы функции разбиения текста на предложения			
2.	Ммышка аааааа за кошку, кошка за Жучку, Жучка за внучку, 34внучка за 63534абку467, 3456а3456ка3 з5а3 дедку57, дедк876а за репку тянут-потянуттттт — вытянули репку, sgsgdf sdghdsj dfkgsdkjl skdjgdfs, sds dh dsfhs sdhfg, dfhsh, sdfhs, dfh dfsh jg34внучка за 63534абку467, 3456а3456ка3 з5а3 дедку57, дедк876а за репку тянут-потянуттттт	Ммышка аааааа за кошку, кошка за Жучку, Жучка за внучку, 34внучка за 63534абку467, 3456а3456ка3 з5а3 дедку57, дедк876а за репку тянут-потянуттттт — вытянули репку, sgsgdf sdghdsj dfkgsdkjl skdjgdfs, sds dh dsfhs sdhfg, dfhsh, sdfhs, dfh dfsh jg34внучка за 63534абку467, 3456а3456ка3 з5а3 дедку57, дедк876а за репку тянут-потянуттттт — вытянули репку. words: 48 ААААА. words: 1	Ввели более длинное предложение, чтобы проверить корректность конкатинации разных частей предложения. Программа отработала корректно, символы на стыке строк не потеряны.

	— вытянули репку. AAAAA. Sssss11 sssss.	Sssss11 sssss. words: 2	
Проверка удаления повторяющихся строк			
3.	ЭТО тест. Это тест.	ЭТО тест. words: 2	Программа отработала корректно, повторяющихся предложений не замечено.
Проверка работы первой функции			
4.	Посадил дед репку — выросла репка большая, пребольшая. Де2д. Стал д1ед репку из земли тянуть: тянет-потянет, вытянуть не может. AAAAA. Sssss11 sssss.	sssss дед репку — выросла репка большая, пребольшая. words: 8 дед. words: 1 д1ед репку из земли тянуть: тянет-потянет, вытянуть не может. words: 9 д1ед. words: 1 sssss. words: 1	Введём предложения состоящие из 1 и 2 слов, чтобы вызвать исключения. Программа отработала корректно.
Проверка второй функции			
5.	Посадил дед репку — выросла репка большая, пребольшая. Де2д. Стал д1ед репку из земли тянуть: тянет-потянет, вытянуть не может. AAAAA. Sssss11 sssss.	Де2д. words: 1 AAAAA. words: 1 Sssss11 sssss. words: 2 Посадил дед репку — выросла репка большая, пребольшая. words: 8 Стал д1ед репку из земли	Введём предложения состоящие менее чем из 3 слов. Программа отработала корректно.

		тянуть: тянет-потянет, вытянуть не может. words: 10	
Проверка третьей функции			
6.	Ммышка аааааа за кошку, кошка за Жучку, Жучка за внучку, 34внучка за 63534абку467, 3456а3456ка3 з5а3 дедку57, дедк876а за репку тянут-потянуттттт — вытянули репку dfh dfsh jg34внучка за 63534абку467.	Ммышка аааааа за кошку, кошка за Жучку, Жучка за внучку, 34внучка за 63534абку467, 3456а3456ка3 з5а3 дедку57, дедк876а за репку тянут-потянуттттт — вытянули репку dfh dfsh jg34внучка за 63534абку467.	Введём предложение с разными видами комбинаций букв и цифр в словах. Программа отработала корректно.
Проверка четвёртой функции			
7.	Ммышка аааааа за кошку, кошка за Жучку, Жучка за внучку, 34внучка за 63534абку467, 3456а3456ка3 з5а3 дедку57, дедк876а за репку тянут-потянуттттт — вытянули репку dfh dfsh jg34внучка за 63534абку467. ААААА. Sssss11 sssss.	Ммышка а за кошку, кошка за Жучку, Жучка за внучку, 34внучка за 63534абку467, 3456а3456ка3 з5а3 дедку57, дедк876а за репку тянут-потянут — вытянули репку dfh dfsh jg34внучка за 63534абку467. words: 28 А. words: 1 Ss1 s. words: 2	Введём разные варианты слов с повторяющимися символами. Программа отработала корректно.

4. ИНСТРУКЦИЯ.

Программа позволяет ввести текст и обработать его одним из 4х способов представленных в подсказке. Вам требуется ввести текст и выбрать одну из команд, узнать о их назначении вам поможет специальная подсказка.

1. Запуск.

Чтобы запустить приложение, откройте папку с файлами приложения. Щелчком правой клавиши мыши откройте диалоговое окно и выберите “открыть в терминале”. Введите команду “make && ./run”.

2. Ввод текста.

Введите текст одной строкой, предложения разделяются точкой с пробелом, слова внутри пробела разделяются пробелом или запятой и пробелом. Для завершения ввода нажмите “Enter”

3. Выбор команды.

Из выведенной подсказки выберите функцию, введите её номер и нажмите “Enter”. Для вывода результата предусмотрена отдельная коанда.

4. Выход.

Работа приложения не завершается после первой введённой команды, для выхода вам необходимо ввести любую цифру отличную от номера команды.

ЗАКЛЮЧЕНИЕ

Выполнив курсовую работу, мы разработали, собрали и протестировали программу, позволяющую обрабатывать введённый текст, согласно техническому заданию. Также мы изучили функции обработки текста в языке Си и закрепили навыки работы с указателями.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Правила оформления пояснительной записки. // se.moevm.info URL:
<http://se.moevm.info/doku.php/courses:programming:report> (дата обращения 12.12.2021)
2. Синтаксис функции сортировки qsort() // www.cplusplus.com URL:
<https://www.cplusplus.com/reference/cstdlib/qsrt/?kw=qsrt> (дата обращения 04.12.2021)
3. Синтаксис функции wcstok() // www.cplusplus.com URL:
<https://www.cplusplus.com/reference/cwchar/wcstok/?kw=wcstok> (дата обращения 08.12.2021)
4. Синтаксис функции поиска вхождения подстроки wcsstr() // www.cplusplus.com URL:
<https://www.cplusplus.com/reference/cwchar/wcsstr/?kw=wcsstr> (дата обращения 05.12.2021)
5. Синтаксис функции конкатинации wcscat() // www.cplusplus.com URL:
<https://www.cplusplus.com/reference/cwchar/wcscat/?kw=wcscat> (дата обращения 05.12.2021)
6. Синтаксис функции копирования wmemcpy() // cppstudio.com URL:
<http://cppstudio.com/post/678/> (дата обращения 07.12.2021)
7. Синтаксис функции wmemmove() // docs.microsoft.com URL:
<https://docs.microsoft.com/ru-ru/cpp/c-runtime-library/reference/memmove-wmemmove?view=msvc-160&viewFallbackFrom=vs-2019> (дата обращения 09.12.2021)

ПРИЛОЖЕНИЕ А

ПРОГРАММНЫЙ КОД

Название файла: main.c

```
#include"inputtext.h"
#include"formattext.h"
#include<stdio.h>
#include<wchar.h>
#include<locale.h>

int main(){
    setlocale(LC_ALL, "");
    wchar_t **some_strings;
    int n = 0, number_sents, command; //n - счётчик количества text_string

    some_strings = input_text(&n);

    struct Sentence *sents;
    sents = make_sentences(n, some_strings, &number_sents);
    number_sents --;

    input_text_free( n, some_strings );
    sents = delete_repeated_sentences(sents, &number_sents);

    struct Text text;
    text.sentences = sents;
    text.sents_number = number_sents;
```

```
while(1){
    wprintf(L"\033[33mДля продолжения введите номер команды от 1
до 5:\n\t1. В каждом предложении заменить первое слово на второе слово из
предыдущего предложения. Для первого предложения, второе слово надо брать
из последнего.\n\t2. Отсортировать предложения по длине третьего слова. Если
слов меньше трех, то длина третьего слова равняется нулю.\n\t3. Вывести на
экран все предложения, в которых в середине слова встречаются цифры.
Данные слова нужно выделить зеленым цветом.\n\t4. В каждом предложении, в
слове, все символы, которые встречаются несколько раз подряд заменить одним
таким символом.\n\t5. Для вывода текущего состояния текста.\nДля завершения
программы введите любую другую цифру\n\033[0m");
```

```
command = 0;
```

```
wscanf(L"%d", &command);
```

```
switch(command){
```

```
    case 1:
```

```
        wprintf(L"\033[31mВыполняю команду 1\n\033[0m");
```

```
        text = command_1(text);
```

```
        break;
```

```
    case 2:
```

```
        wprintf(L"\033[31mВыполняю команду 2\n\033[0m");
```

```
        text = command_2(text);
```

```
        break;
```

```
    case 3:
```

```
        wprintf(L"\033[31mВыполняю команду 3\n\033[0m");
```

```
        command_3(text);
```

```
        break;
```

```
    case 4:
```

```
        wprintf(L"\033[31mВыполняю команду 4\n\033[0m");
```

```

        text = command_4(text);
        break;
    case 5:
        wprintf(L"\n");
        for(int i = 0; i < text.sents_number; i++)
            wprintf(L"%ls\nwords: %d\n\n", text.sentences[i].str,
text.sentences[i].words_number);
        break;
    default:
        wprintf(L"Вы вышли из программы\n");
        return 0;
    }
}
return 0;
}

```

Название файла: formattext.c

```
#include "formattext.h"
```

```

wchar_t* slice(wchar_t* str, int start, int end){
    wchar_t *res, *try;
    try = calloc( (end - start + 1), sizeof(wchar_t) );
    if(try != NULL){
        res = try;
    }else{
        wprintf(L"In func slice. Не удалось выделить память.\n");
        return L"";
    }
    for(int i = start; i < end; i++)

```

```

        res[i - start] = str[i];
    res[end - start] = L'\0';
    return res;
}

```

```

wchar_t* wstr_tolower( wchar_t* first_str){
    wchar_t*res, *try;
    try = (wchar_t*)calloc( wcslen(first_str)+1, sizeof(wchar_t));
    if(try != NULL){
        res = try;
    } else {
        wprintf(L"func wstr_tolower. Не удалось выделить память, возможно
некорректное сравнение.\n");
        return first_str;
    }
    for(int i = 0; i < wcslen(first_str) - 1; i++)
        res[i] = towlower(first_str[i]);
    res[wcslen(first_str)-1] = '.';
    res[wcslen(first_str)] = '\0';
    return res;
}

```

```

wchar_t* get_second_word(struct Text text, int index){
    wchar_t *temp, *pc, *res, *str;
    res = malloc( sizeof(wchar_t)*(wcslen(text.sentences[index].str) + 1));
    if(res != NULL){
        str = res;
    } else {

```

```

        wprintf(L"func get_second_word. Не удалось выделить память\n");
        return L"";
    }
    wmemcpy(str, text.sentences[index].str, (wcslen(text.sentences[index].str) +
1));
    if(text.sentences[index].words_number >= 2){
        temp = wcstok(str, L",. ", &pc);
        temp = wcstok(NULL, L",. ", &pc);
        res = slice(temp, 0, wcslen(temp));
        return res;
    } else {
        return L"";
    }
}

```

```

int compare(const void* sent1, const void* sent2){

    struct Sentence* p1 = (struct Sentence*) sent1;
    struct Sentence* p2 = (struct Sentence*) sent2;

    struct Sentence first = *p1;
    struct Sentence second = *p2;

    int len1, len2;
    wchar_t *word1, *word2, *pc, *str1, *str2, *try;

    if(first.words_number < 3){
        len1 = 0;
    } else {

```



```

        try = calloc(wcslen(first.str), sizeof(wchar_t));
    if(try != NULL){
        str1 = try;
    } else {
        wprintf(L"Не удалось выделить память в функции compare\n");
    }
    wmemcpy(str1, first.str, wcslen(first.str));
    word1 = wcstok(str1, L",. ", &pc);
    word1 = wcstok(NULL, L",. ", &pc);
    word1 = wcstok(NULL, L",. ", &pc);
    len1 = wcslen(word1);
    free(str1);
}

if(second.words_number < 3){
    len2 = 0;
} else {
    try = calloc(wcslen(second.str), sizeof(wchar_t));
    if(try != NULL){
        str2 = try;
    } else {
        wprintf(L"Не удалось выделить память в функции compare\n");
    }
    wmemcpy(str2, second.str, wcslen(second.str));
    word2 = wcstok(str2, L",. ", &pc);
    word2 = wcstok(NULL, L",. ", &pc);
    word2 = wcstok(NULL, L",. ", &pc);
    len2 = wcslen(word2);
    free(str2);
}

```

```

        if(len1 > len2)
            return 1;
        if(len1 == len2)
            return 0;
        if(len1 < len2)
            return -1;
    }

```

```

int numbers_inside(wchar_t *str){
    int i = 0, c = 0, start = 0, finish = wcslen(str) - 1;
    wchar_t coma[] = L",", fullstop[] = L".";
    if(isdigit(str[i]))
        while(isdigit(str[i]))
            i++;
    start = i;
    if(str[finish] == L',' || str[finish] == L'.')
        finish--;
    i = finish;
    if(isdigit(str[i]))
        while((isdigit(str[i])) && (i > 0))
            i--;
    finish = i + 1;

    while(start < finish){
        if(c == 0){
            if (isdigit(str[start]))
                return 1;

```

```

        }
        start ++;
    }

    return 0;
}

struct Text command_4(struct Text text){

    for(int i = 0; i < text.sents_number; i++){
        int j = 1;
        while(j < wcslen(text.sentences[i].str)){
            if((text.sentences[i].str[j] != L' ' ) && (text.sentences[i].str[j] !=
L',') && (text.sentences[i].str[j] != L'.')){
                if(text.sentences[i].str[j] == text.sentences[i].str[j-1]){
                    memmove(text.sentences[i].str + j - 1,
text.sentences[i].str + j, (wcslen(text.sentences[i].str) - j + 1) * sizeof(wchar_t));
                } else {
                    j++;
                }
            }else{j++;}
        }
    }
    return text;
}

```

```

void command_3 (struct Text text){

```

```

wchar_t *str, *try, **tryr, *pc, *temp, **words, coma[] = L",", fullstop[] =
L".";

int c, counter, flag, sum;
for(int i = 0; i < text.sents_number; i++){
    c = 0;
    try = calloc(wcslen(text.sentences[i].str) + 1, sizeof(wchar_t));
    if(try != NULL){
        str = try;
    } else {
        wprintf(L"Не удалось выделить память в функции
command_3\n");
        continue;
    }
    wmemcpy(str, text.sentences[i].str, wcslen(text.sentences[i].str) + 1);
    temp = str;
    do{
        temp = wcsstr(temp + 1, L",");
        c++; // количество запятых + 1 точка
    }while(temp != NULL);
    sum = text.sentences[i].words_number + c;
    tryr = calloc(sum, sizeof(wchar_t*));
    if(tryr != NULL){
        words = tryr;
    } else {
        free(str);
        continue;
    }

    c = 0;
    flag = 0;

```

```

temp = wctok(str, L" ", &pc);
do{

    words[c] = temp;
    if(wcsstr(temp, L",") != NULL){
        words[c] = slice(temp, 0, wcslen(temp) - 1);
        c++;
        words[c] = coma;
    }
    if(wcsstr(temp, L".") != NULL){
        words[c] = slice(temp, 0, wcslen(temp) - 1);
        c++;
        words[c] = fullstop;
    }
    c++;

    if(numbers_inside(temp))
        flag = 1;

    temp = wctok(NULL, L" ", &pc);
}while(temp != NULL);

counter = 1;
if(flag){
    if(wcslen(words[0]) > 2){
        if(numbers_inside(words[0])){
            wprintf(L"\033[32m%ls\033[0m", words[0]); //green
        }else{
            wprintf(L"%ls", words[0]);
        }
    }
}

```

```

        }else{
            wprintf(L"%ls", words[0]);
        }
        while(counter < c){
            if(wcsstr(words[counter], L",") != NULL){
                wprintf(L",");
            }else if(wcsstr(words[counter], L".") != NULL){
                wprintf(L".");
            }else{
                if(numbers_inside(words[counter])){
                    wprintf(L"\033[32m          %ls\033[0m",
words[counter]);//green
                }else{
                    wprintf(L" %ls", words[counter]);
                }
            }
            counter ++;
        }
        wprintf(L"\n");
        free(str);
        free(words);
    }
}

```

```

struct Text command_2 (struct Text text){
    qsort(text.sentences, text.sents_number, sizeof(struct Sentence), compare);
    return text;
}

```



```

struct Text command_1 (struct Text text){
    wchar_t *str, *second_word, *try, fullstop[] = L".\0";
    int start, last = text.sents_number - 1, end, m, ch, size;
    struct Text result;
    struct Sentence *tryr;
    result.sents_number = text.sents_number;
    tryr = calloc(text.sents_number, sizeof(struct Sentence));
    if(tryr != NULL)
        result.sentences = tryr;

    for(int i = 0; i < text.sents_number; i++){
        if(i == 0){
            second_word = get_second_word(text, last);
            if(text.sentences[i].words_number != 1){
                start = (int)(wcsstr(text.sentences[i].str,L"
text.sentences[i].str);
                if(second_word != L"") {
                    str = slice(text.sentences[i].str,start, wcslen(text.sentences[i].str));
                } else {
                    str
=slice(text.sentences[i].str,start+1, wcslen(text.sentences[i].str));
                }

            } else {
                str = L"";
            }
            m = wcslen(second_word) + wcslen(str);
            try = (wchar_t *)calloc(m + 1, sizeof(wchar_t));

```

```

        if(try != NULL)
            result.sentences[i].str = try;
        wscat( result.sentences[i].str, second_word); //second_word
        wscat( result.sentences[i].str, str); // + str
        result.sentences[i].words_number =
text.sentences[i].words_number;
        if(wcslen(second_word) == 0)
            result.sentences[i].words_number --;
    } else {
        second_word = get_second_word(text, i-1);
        free(result.sentences[i].str );
        result.sentences[i].str = malloc( 1*sizeof(wchar_t));
        if(text.sentences[i].words_number != 1){
            start = (int)(wcsstr(text.sentences[i].str,L"
"-
text.sentences[i].str);
            if(second_word != L"") {
                str = slice(text.sentences[i].str,start,
wcslen(text.sentences[i].str));
            } else {
                str
=slice(text.sentences[i].str,start+1,wcslen(text.sentences[i].str));
            }
        } else {
            str = L"";
        }
        m = wcslen(second_word) + wcslen(str) + 1;
        try = realloc(result.sentences[i].str, (m)*sizeof(wchar_t));
        if(try != NULL)
            result.sentences[i].str = try;
    }
}

```

```

        wmemcpy(result.sentences[i].str,                second_word,
wcslen(second_word));
        wcscat( result.sentences[i].str, str);
        result.sentences[i].words_number                =
text.sentences[i].words_number;
        if(wcslen(second_word) == 0)
            result.sentences[i].words_number --;
    }

    if(result.sentences[i].str[wcslen(result.sentences[i].str) - 1] != L'.')
        wcscat(result.sentences[i].str, fullstop);
}
for(int i = 0; i < text.sents_number; i++)
    free(text.sentences[i].str);
free(text.sentences);

return result;
}

```

```

struct Sentence *delete_repeated_sentences(struct Sentence *sentences, int *p){
    int i = 1, c = 0, j = 0, res_counter = 1, sents_num = *p;
    struct Sentence* result, *reserv_result;
    wchar_t* first, *second, *wstr;
    reserv_result = calloc( 1, sizeof(struct Sentence));
    if(reserv_result != NULL)
        result = reserv_result;
    result[0].str = sentences[0].str;
    result[0].words_number = sentences[0].words_number;

```

```

while( i < sents_num){ /* 1 --> sents_num - 1 */
    first = wstr_tolower(sentences[i].str);
    j = 0;
    c = 0;
    while( j < i){
        second = wstr_tolower(sentences[j].str);
        if(wscmp(first, second) == 0)
            c = 1;
        j++;
    }
    if(c == 0){
        res_counter ++;
        reserv_result = realloc( result, res_counter*sizeof(struct
Sentence));

        if(reserv_result != NULL)
            result = reserv_result;
        result[res_counter-1].str = sentences[i].str;
        result[res_counter-1].words_number =
sentences[i].words_number;
    }else{
        free(sentences[i].str);
    }
    i ++;
}
free(sentences);
*p = res_counter;

return result;
}

```

```

void input_text_free(int n, wchar_t** text_string){
    for(int i = 0; i < n; i++){
        free(text_string[i]);
    }
    free(text_string);
}

```

Название файла: formattext.h

```

#pragma once

```

```

#include<stdio.h>

```

```

#include<stdlib.h>

```

```

#include<string.h>

```

```

#include<wchar.h>

```

```

#include<wctype.h>

```

```

#include<ctype.h>

```

```

#ifndef __Structs__

```

```

#define __Structs__

```

```

struct Sentence{
    wchar_t* str;
    int words_number;
};

```

```

struct Text{
    struct Sentence *sentences;
    int sents_number;
};

```

```
#endif
```

```
struct Text command_1 (struct Text text);
```

```
struct Text command_2 (struct Text text);
```

```
void command_3 (struct Text text);
```

```
struct Text command_4(struct Text text);
```

```
wchar_t* slice(wchar_t* str, int start, int end);
```

```
wchar_t* wstr_tolower( wchar_t* first_str);
```

```
wchar_t* get_second_word(struct Text text, int index);
```

```
int compare(const void* sent1, const void* sent2);
```

```
int numbers_inside(wchar_t *str);
```

```
struct Sentence *delete_repeated_sentences(struct Sentence *sentences, int *p);
```

```
void input_text_free(int n, wchar_t** text_string);
```

Название файла: inputtext.c

```
#include"inputtext.h"
```

```
wchar_t** input_text(int* c){
```

```
    int n = 0;
```

```

wchar_t **text_string, **reserv_text_string, *reserv, ch = ' ';
reserv_text_string = malloc( sizeof(wchar_t)*1 );
if(reserv_text_string != NULL)
    text_string = reserv_text_string;
while(ch != '\0'){
    n ++;
    reserv_text_string = (wchar_t **)realloc( text_string, n*sizeof(wchar_t*)
);

    if(reserv_text_string != NULL){
        text_string = reserv_text_string;
        reserv = malloc( 100*sizeof(wchar_t) );
        if(reserv != NULL)
            text_string[n - 1] = reserv;
        text_string[n - 1][98] = '\0';
        fgetws(text_string[n - 1], 100, stdin);
        ch = text_string[n - 1][98];

        if(wcslen(text_string[n-1]) == 1){
            free(text_string[n - 1]);
            n --;
        }
    }
}
*c = n;

return text_string;
}

```

```

struct Sentence *make_sentences(int n, wchar_t **text_string, int* p){

```

```

struct Sentence *sentences = malloc(sizeof(struct Sentence)), *try;
int i = 0, start = 0, sentence_counter = 1, hun = 1, r = 0, numb;
wchar_t *end, *sentence, *try_to_realloc, fullstop[] = L".\0";

try_to_realloc = calloc( 100, sizeof(wchar_t) );//сборка предложения, будет
записано в структуру
if(try_to_realloc != NULL){
    sentence = try_to_realloc;
} else{
    wprintf(L"Не удалось выделить память для сборки
предложений.\n");
    return sentences;
}
while( i < n ){
    end = wcsstr( text_string[i] + r + 1, L"." );
    if(end == NULL){
        wscat(sentence, slice(text_string[i], start, wcslen(text_string[i])));
        hun ++;
        i ++;
        r = 0;
        start = 0;
        try_to_realloc = realloc( sentence, hun * 100 * sizeof(wchar_t) );
        if(try_to_realloc != NULL){
            sentence = try_to_realloc;
        } else {
            wprintf(L"Не удалось выделить память для текущего
предложения.\n");
            free(sentence);
            try_to_realloc = calloc(100, sizeof(wchar_t));
            if(try_to_realloc != NULL){

```



```

        sentence = try_to_realloc;
    } else {
        return sentences;
    }
}
} else {
    wscat(sentence, slice(text_string[i],start, (int)(end -
text_string[i])));
    wscat(sentence, fullstop);
    r = (int)(end - text_string[i]);

    try = realloc( sentences, sentence_counter*sizeof(struct
Sentence));

    if(try != NULL){
        sentences = try;
    } else {
        return sentences;
    }

    sentences[sentence_counter-1].str =
(wchar_t*)malloc((wcslen(sentence) + 1)*sizeof(wchar_t));
    wmemcpy(sentences[sentence_counter-1].str, sentence,
wcslen(sentence) + 1);

    numb = 0;
    for(int j = 0; j < wcslen(sentence); j++)
        if(iswspace(sentence[j]))
            numb ++;

    sentences[sentence_counter-1].words_number = numb + 1;
    sentence_counter ++;

```

```

        if(i != n)
            start = ((int)(end - text_string[i]) + 2) % 99;
        free(sentence);
        sentence = calloc( 100, sizeof(wchar_t) );
        hun = 1;
        if((int)(end - text_string[i]) == 98 || (int)(end - text_string[i]) ==
99){
            r = 0;
            i ++;
        }
    }
}

*p = sentence_counter;

return sentences;
}

```

Название файла: inputtext.h

```

#include<stdio.h>
#include<stdlib.h>
#include<wchar.h>
#include"formattext.h"

#ifndef __Structs__
#define __Structs__
struct Sentence{
    wchar_t* str;
    int words_number;
};

```

```
    struct Text{
        struct Sentence *sentences;
        int sents_number;
    };
#endif

wchar_t** input_text(int *c);

struct Sentence* make_sentences(int n, wchar_t **text_string, int* p);
```

Название файла: Makefile

```
all: main.o inputtext.o formattext.o
    gcc main.o inputtext.o formattext.o -o run

main.o: main.c inputtext.h formattext.h
    gcc -c main.c

inputtext.o: inputtext.c inputtext.h
    gcc -c inputtext.c

formattext.o: formattext.c formattext.h
    gcc -c formattext.c

clean:
    rm -rf *.o run
```