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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка текста

Студентка гр. 2384 _____ Валеева А. А.

Преподаватель _____ Гаврилов А. В.

Санкт-Петербург
2022

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Валеева А. А.

Группа 2384

Тема работы: Обработка текста

Исходные данные:

Вариант 6

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. Распечатать каждое слово которое встречается не более одного раза в тексте.
2. Каждую подстроку в тексте имеющую вид “<день> <месяц> <год> г.” заменить на подстроку вида “ДД/ММ/ГГГГ”. Например, подстрока “20 апреля 1889 г.” должна быть заменена на

“20/04/1889”.

3. Отсортировать предложения по произведению длин слов в предложении.
4. Удалить все предложения, которые содержат символ ‘#’ или ‘№’, но не содержат ни одной цифры.

Все сортировки и операции со строками должны осуществляться с использованием функций стандартной библиотеки. Использование собственных функций, при наличии аналога среди функций стандартной библиотеки, запрещается.

Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов (например, функции обработки текста в один, функции ввода/вывода в другой). Также, должен быть написан Makefile.

Содержание пояснительной записки:
«Аннотация», «Содержание», «Введение», «Цель», «Описание выполнения работы», «Программная реализация функционала», «Заключение», «Список использованных источников», «Примеры работы кода», «Исходный код».
Использование программы: текст вводиться в одну строку, слова разделяются сепараторами « » и «,» ;предложения разделены «.». Ввод текста завершается при вводе символа «\n».
Предполагаемый объем пояснительной записки:
Не менее 20 страниц.

Дата выдачи задания: 20.10.2022

Дата сдачи реферата: 22.12.2022

Дата защиты реферата: 24.12.2022

Студентка _____ Валеева А. А.

Преподаватель _____ Гаврилов А. В.

АННОТАЦИЯ

Курсовая работа заключается в реализации программы для обработки текста на языке Си. Для хранения и работы с текстом были использованы структуры и функции стандартных библиотек языка Си.

Сначала программа выводит подсказку о том, что нужно вводить текст, считывает его и удаляет одинаковые, независимо от регистра, предложения. Выводит подсказку, запрашивает у пользователя, что делать дальше, предусмотрена опция выхода из программы, а так же обработка неправильного ввода. Далее программа выполняет функции, которые вызывает пользователь, пока он не введет значение для выхода из программы. При выходе из программы производятся действия по очистке выделенной памяти.

Пример работы программы приведен в приложении А.

Исходный код программы приведен в приложении В.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы:

Написать программу, выполняющую действия по обработке текста, выбранные пользователем. Реализация программы происходит с помощью использования структур (для хранения текста, предложений и некоторых данных о них), стандартных библиотек (в том числе wchar.h и wctype.h для работы с кириллическими буквами) и выделения динамической памяти. Сборка программы происходит с помощью Makefile и утилиты make.

Задачи:

Для достижения поставленной цели требуется решить следующие задачи: изучить синтаксис языка программирования C, разработать код программы, написать Makefile, собрать проект, протестировать работоспособность программы. Программа разработана для операционных систем на базе Linux. Разработка производилась на операционной системе Ubuntu Linux в IDE Clion и редакторе Nano.

ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Создание и объявление структур.

Для работы с текстом созданы структуры *Sentence, Text, Word, Dictionary*:

Переменные структуры *Sentence*:

wchar_t sentence_arr* – указатель на массив предложения;

int count_char – длина предложения;

int multiplication – произведение длин слов в предложении;

Переменные структуры *Text*:

*Sentence_s *text_arr* – указатель на массив предложений;

int count_sentence – количество предложений;

Переменные структуры *Words*:

wchar_t word_arr* – указатель на слово;

int count_symbols – длина слова;

int repeat – количество повторов слова;

Переменные структуры *Dictionary*:

*Word_s *word_arr* – указатель на массив слов;

int count_words – количество слов;

Структуры объявлены в заголовочном файле *structs.h.*

2.2. Считывание текста и обработка ошибок.

Считывание текста осуществляется при помощи функций *read_text()* и *read_sent()*, обработка ошибок происходит в функции *err()*. Функции находятся в заголовочном файле *read.h*.

1) Рассмотрим функцию *read_sent()* :

В качестве аргументов принимает указатель на две переменные структуры предложения: *count_char, multiplication,* а возвращает указатель на

массив символов – предложение. Создается массив типа *wchar_t** под который выделяется динамическая память с помощью функции *malloc*, если выделенной памяти недостаточно, то увеличиваем переменную *size* на размер макроса *MEM_SIZE*. Если памяти не хватает (*NULL*), тогда выводится ошибка №1.

С помощью цикла *do-while* посимвольно считывается предложение до встречи символа окончания строки (в нашем случае это «.»). В массив символов добавляется считанные символ, длина увеличивается на 1. С помощью переменной *tabulation_start* и оператора *if-else* пропускаем табуляцию в начале предложения. Также, используя цикл *if,* мы проверяем случай, когда пользователь ввел единственный символ - «\n». В таком случае возвращается массив из одного символа - «\n». После ввода пользователем «.» цикл *do-while* заканчивается, переменная *count_char* получается значение переменной *len*, считающей количество введенных символов, переменная *multiplication* равна переменной *pr*, которая подсчитывается в операторе *if-else*. Предложение завершается символом конца строки «\0». В конце возвращаем предложение.

2) Функция *read_text()*:

В качестве аргумента принимает указатель на текст. В цикле *while* создается новое предложение, а в его массив символов (т.е. само содержимое предложения) вводятся символы при помощи функции *read_sent()*. Делается проверка, на перенос строки, если он был – считывание текста завершается. Полученное предложение добавляется в массив предложений текста, увеличивается количество предложений *count_sentence* на 1. Если память в массиве предложений закончилась, производится ее перевыделение с бОльшим объемом при помощи функции *realloc(),* если это возможно.

3) Функция *creat_empy_dict()* служит для создания пустого словаря, для которого выделает память с помощью функции *malloc()*. Указатель на массив слов изначально NULL, количество также ноль. Возвращаем пустой словарь.

4) Обратная ей функция — *free_dict()*, на вход которой передается словарь, с помощью цикла *for* проходим по словам, заводим переменную word

структурного типа, очищаем сначала сами слова, потом предложение, потом указатели.

5) Функция *creat_dictionaru_from_sent()* используется для создания массива слов предложения. В качестве аргументов поступают два указателя на массивы - слов и сепараторов, а также само предложение. Делаем две копии предложения для разрезания на токены (для слов и для сепараторов). Выделаем память, если это возможно. Используется два указателя — *old_ptr* и *ptr.* Пока предложение не закончилось (*word != NULL*), с помощью функции *add_word*() (будет описана далее) добавляем слово в словарь словарь(слово получаем с помощью *wcstok()*), передавая словарь слов, указатель на нулевой элемент слова и его длину (находится следующим образом — указатель на новое слово минус указатель на нулевой символ предыдущего слова и минус 1). Далее если в словаре слов больше 1 слова, то также добавляем сепаратор в словарь сепараторов, предварительно заменяя символ после сепаратора на символ окончания строки «\0».

Не забываем поменять указатель на предыдущее слово (*old_ptr*) на только что использованный (*ptr*) и вырезать новое слово. В конце освобождаем память, выделенную под две копии строк.

6) Функция *add_word*() помогает добавлять слово в словарь. В качестве аргумента получает указатель на словарь, куда добавить слово(сепаратор), указатель на начало слова(сепаратора), а также его длину. Для начала заводим переменную *word_arr* типа *Word_s* *, если строка = NULL, тогда выделяем для нее память, если уже заполнена — то перевыделяем, проверяем на наличие памяти. Далее в предложении берем переменную (перебираем по индексам), типа структуры слова, и заполняем ее поля, для поля слова выделяем память с помощью *calloc().* Функцией *wcsncpy()* в выделенную под слова память копируем полученное на вход слово, длина = *len.* Количество слов увеличиваем.

7) Функция *sent_from_dict()* используется для «склеивания» строки обратно. Если больше одного слова в предложении, тогда вносим в

предложение нулевое слово из словаря. Далее циклом *for и* функцией *wcscat()* склеиваем строку (где уже есть нулевое слово) со сепаратором и новым словом. В конце добавляем в склейку точку, как окончание строки.

7) Функция *err()*:

В качество аргумента принимает целочисленное число и в зависимости от его значения выводит ошибку. Используется оператор *switch*. Обрабатывается два типа ошибок: невозможность выделения памяти и ввод неверного значения при выборе команды пользователем.

Функции объявлены в заголовочном файле *read_and_err.h*.

2.3. Функции обработки и изменения текста.

1. Первая функция — *del_replay_sentence(),* которая удаляет из текста два одинаковых предложения вне зависимости от регистра (для этого используется функция *wcscasecmp из библиотеки wchar.h.* Для ускорения программы два предложения сравниваются по длине. Если предложения совпадают, то для используем функцию *memmove*, которая «сдвигает» предложения, идущие после первого совпавшего (второе пропускаем). Так как количество предложений уменьшилось, уменьшаем *count_sentence* на 1. Переменную *j* так же уменьшаем, чтобы курсор остался на том же индексе и работал с новым предложением на том же месте.

2. Следующая функция — *del_special_sentence()*. В задание №4 нам требуется удалить все предложения, которые содержат символ ‘#’ или ‘№’, но не содержат ни одной цифры. Заводим два флага — *flag_simbol* для нахождения символов ‘#’ или ‘№’, и *flag_digit* для нахождения цифры в строке.

В качестве аргумента передаем указатель на текст, с помощью вложенных циклов проходим посимвольно по каждой строке. Если символ ‘#’ или ‘№’, то *flag_simbol* = 1, аналогично при встрече цифры *flag_digit* = 1. Находим цифры при помощи функции *isdigit()*. После каждого предложения проверяем значения

флагов, если *flag_digit = 0* (нет цифр), а *flag_simbol = 1* (есть символ), то используем функцию *memmove* для «сдвига» предложений на место удаленного. Уменьшаем количество предложений в тексте, а так же переменную *i*, для того, чтобы не «перепрыгнуть» через предложение, вставшее на место удаленного. В конце каждого предложения не забываем обнулять флаги. В конце печатаем подходящие под условия предложения.

Функции объявлены в заголовочном файле *sent_changes.h.*

2.4. Функции заданий курсовой работы.

1. Для начала рассмотрим функцию *creat_dict_from_text()* для выполнения задания под номером 1, в котором требуется вывести слова, встречающиеся ровно один раз в тексте. Аргументы: текст, словарь, в котором будут находиться, встречающиеся ровно один раз в тексте. Для начала с помощью функции *creat_empy_dict()* создаем два пустых словаря — для слов и для сепараторов. Далее перебираем предложения текста и для каждого функцией *creat_dictionaru_from_sent()* создаем словарь и кладем в словари слова и сепараторы (с каждым следующим предложением словари *words* и *seps* будут заполняться). Теперь нам нужно в словарь *once_words()* положить только те слова, которые встречаются 1 раз, для этого заводим флаг, перебираем двумя циклами *for* словарь *words* и словарь *once_words()*, если слова совпали (проверяем функцией *wcscasecmp* без учета регистра), то поле *repeat* увеличиваем на 1, флаг меняем. Если же флаг не изменился, то в таком случае *repeat* слова = 1, добавляем его в словарь *once_words()* с помощью функции *add_word()*. Используем функцию *print_dict()* для печати слов и проверки значения поля .

2. Для 2 задания были созданы следующие функции — *is_number(), number_of_month(), is_correct_data, replace_data()*. Рассмотрим каждую:

1) первая функция *is_number()* принимает на вход указатель на нулевой символ строки и посимвольно проверяет строку — число или нет. Если нет — то возвращаем 1, иначе 0.

2) следующая функция - *number_of_month()* хранит в себе статический двумерный массив месяцев. С помощью цикла for проверяем поступившее на вход слово с каждым месяцем (без учета регистра), если совпало — возвращаем найденный индекс+1 (т. к. индексы с 0, а номера месяца с 1). Иначе возвращаем 0.

3) далее проверим введенную дату на корректность функцией *is_correct_data, replace_data()*, день должен находиться в промежутке от 1 до 31, а год с 1 до 9999. Далее заводим переменную *limits* = 31 (служит для проверки дня определенного месяца), с помощью *switch()* проверяем введенный номер месяца и изменяем переменную *limits* в зависимости от месяца. Если введенная дата меньше или равна верному значению дня месяца (*limits*), то в таком случае возвращается *True*.

4) завершающая функция второго задания — *replace_data()*. В качестве аргумента поступает текст, который перебираем по предложениям. Создаем два пустых словаря функцией *create_empty_dict()* и потом заполняем словарь предложения. Проходимся по четверкам слов — день, месяц, год и буква «г», функцией *wcscmp()* проверяем «г», функцией *is_number()* проверяем число и год, функцией *number_of_month()* получает номер месяца (или 0, если месяц не найден). Далее для преобразования строки в целое длинное число используем функцию *wcstoul()* (для года и числа). Перевыделяем память для даты, если возможно, далее с помощью функции *swprintf()* записываем форматированные данные в строку. Как это работает: "%02d означает форматировать целое число с двумя цифрами, заполнив его слева нулями. Перезаписываем новое количество символов в поле *count_symbols*. Проделываем то же самое с месяцем и годом. Отдельно работаем с буквой «г» - в поле слова записываем «\0», удаляя тем самым «г» и заменяя ее длину на 0. В конце нужно добавить «/»

между датой и месяцем и месяцем с годом, делаем это с помощью функции *wcscopy()*. В конце склеиваем получившееся предложение и очищаем словари.

3. Для выполнения задания №3 написаны функции — *text_multiplication()* и подпрограмма *compare*(), которая используется для *qsort*().

1) *text_multiplication()* — печатает предложения и произведения длин слов в них. Аргументом выступает указатель на начало текста.

2) *compare()* — функция, на вход которой поступает два указателя типа *const void**. Инициализируем переменные (две строки), далее выполняем следующие действия: если у первого предложение произведение длин слов больше, чем у второго, то возвращаем -1, если наоборот, то 1. При равенстве 0.

Для сортировки используем функцию *qsort()*, на вход которой передаются следующие 4 аргумента: указатель на начало массива (текста), его размер (количество предложений), размер элемента и указатель на функцию *compare()*. Сортируются предложения по возрастанию.

2.5. Создание Makefile.

Программы была разделена на следующие файлы:

structs.h – заголовочный файл, в котором содержится объявление структур Text и Struct;

structs.c – файл, в котором содержатся стурктуры Text и Struct;

read_and_err.c — файл, в котором содержаться функции ввода текста, а также функция вызова ошибок;

read_and_err.h — заголовочный файл, в котором содержаться сигнатуры функций ввода текста и вызова ошибок;

sent_chahg.c — файл, в котором содержаться функции обработки и изменения текста;

sent_change.h — файл, содержащий вызовы функций обработки и изменения текста;

cw_all_func.c — файл, содержащий функции, используемые для решений заданий курсовой работы;

cw_all_func.h — файл, содержащий вызовы функций, используемых для выполнения заданий;

main.c — файл, в котором содержатся функции-исполнители.

Сборка программы осуществляется с помощью Makefile, в котором прописаны все необходимые цели и зависимости, и утилиты make.

ЗАКЛЮЧЕНИЕ

Были изучены основы языка программирования Си, применены в ходе работы его составляющие, такие как: функции стандартных библиотек, методы работы с динамической памятью и структуры. Реализовано решение задания к курсовой работе, написана программа, осуществляющая считывание и обработку введенных данных и вывод текста согласно указанным условиям.

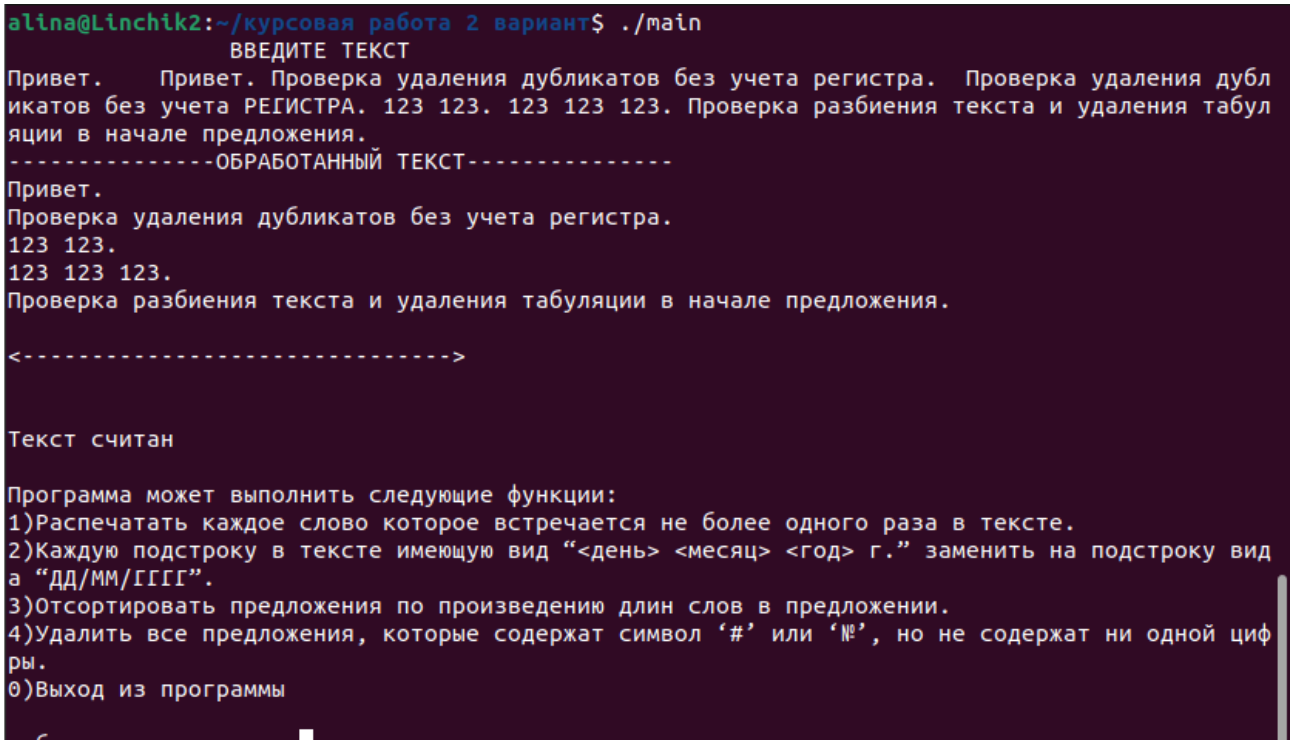
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кринкин К. В., Берленко Т. А., Заславский М. М., Чайка К. В., Допира В. Е., Гаврилов А. В. Методические указания по выполнению курсовой и лабораторных работ по дисциплине программирование. Первый семестр, 2022.

2. Брайан Керниган, Деннис Ритчи. Язык программирования Си.

ПРИЛОЖЕНИЕ А
ПРИМЕР РАБОТЫ ПРОГРАММЫ

1. Пример введенного и обработанного теста (удалены дубликаты и табуляция в начале предложения), вывод меню для пользователя:



2. Выполнение индивидуального задания 1:

3. Выполнение индивидуального задания 2:

```
                ВВЕДИТЕ ТЕКСТ
проверка 2 задания курсовой работы 12 Апреля 1876 г верная дата, а 31 ноября 1876 г неверна
я, 12 ноября 1876 тоже неверна, так как нет буквы г.
---------------ОБРАБОТАННЫЙ ТЕКСТ--------------
проверка 2 задания курсовой работы 12 Апреля 1876 г верная дата, а 31 ноября 1876 г неверна
я, 12 ноября 1876 тоже неверна, так как нет буквы г.

<------------------------------>


Текст считан

Программа может выполнить следующие функции:
1)Распечатать каждое слово которое встречается не более одного раза в тексте.
2)Каждую подстроку в тексте имеющую вид “<день> <месяц> <год> г.” заменить на подстроку вид
а “ДД/ММ/ГГГГ”.
3)Отсортировать предложения по произведению длин слов в предложении.
4)Удалить все предложения, которые содержат символ ‘#’ или ‘№’, но не содержат ни одной циф
ры.
0)Выход из программы

выберете что делать: 2
проверка 2 задания курсовой работы 12/04/1876  верная дата, а 31 ноября 1876 г неверная, 12
 ноября 1876 тоже неверна, так как нет буквы г.

<------------------------------>
alina@Linchik2:~/курсовая работа 2 вариант$
```

4. Выполнение индивидуального задания 3:

```
Проверка 3 задания.
Сортировка по произведения длин слов в предложении.
привет привет.
123 123.
1 1 1 1.
2 2 2.
20 декабря.

<------------------------------>


Текст считан

Программа может выполнить следующие функции:
1)Распечатать каждое слово которое встречается не более одного раза в тексте.
2)не прописана.
3)Отсортировать предложения по произведению длин слов в предложении.
4)Удалить все предложения, которые содержат символ ‘#’ или ‘№’, но не содержат ни одной цифры.
0)Выход из программы

выберете что делать: 3
<------------------------------>
                SORTED

произведение длин слов: <1>      sentence <1 1 1 1.>
произведение длин слов: <1>      sentence <2 2 2.>
произведение длин слов: <9>      sentence <123 123.>
произведение длин слов: <14>     sentence <20 декабря.>
произведение длин слов: <36>     sentence <привет привет.>
произведение длин слов: <56>     sentence <Проверка 3 задания.>
произведение длин слов: <42240>  sentence <Сортировка по произведения длин слов в предложении.>
<------------------------------>

выберете что делать:
```

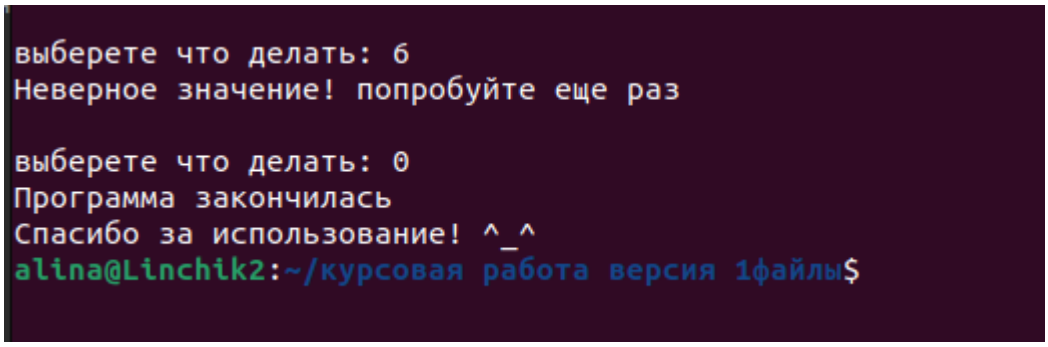
5. Выполнение индивидуального задания 4:

```
                ВВЕДИТЕ ТЕКСТ
Прове№рка за4дания под номером 4. Если есть символ # или №, но нет цифры, то уда
ляю предложение. Если и 4 и #, то оставляю. Если только 444, то оставляю.
---------------ОБРАБОТАННЫЙ ТЕКСТ---------------
Прове№рка за4дания под номером 4.
Если есть символ # или №, но нет цифры, то удаляю предложение.
Если и 4 и #, то оставляю.
Если только 444, то оставляю.

<------------------------------>


Текст считан

Программа может выполнить следующие функции:
1)Распечатать каждое слово которое встречается не более одного раза в тексте.
2)не прописана.
3)Отсортировать предложения по произведению длин слов в предложении.
4)Удалить все предложения, которые содержат символ ‘#’ или ‘№’, но не содержат н
и одной цифры.
0)Выход из программы

выберете что делать: 4
<------------------------------>
                AFTER DELETE

Прове№рка за4дания под номером 4.
Если и 4 и #, то оставляю.
Если только 444, то оставляю.
<------------------------------>

выберете что делать:
```

6. Неверный ввод символа пользователем:

```
alina@Linchik2:~/курсовая работа версия 1файлы$ ./main
               ВВЕДИТЕ ТЕКСТ
Проверка на неверный ввод символа.
---------------ОБРАБОТАННЫЙ ТЕКСТ---------------
Проверка на неверный ввод символа.

<------------------------------>


Текст считан

Программа может выполнить следующие функции:
1)Распечатать каждое слово которое встречается не более одного раза в тексте.
2)не прописана.
3)Отсортировать предложения по произведению длин слов в предложении.
4)Удалить все предложения, которые содержат символ ‘#’ или ‘№’, но не содержат н
и одной цифры.
0)Выход из программы

выберете что делать: 6
Неверное значение! попробуйте еще раз

выберете что делать:
```

7. Выход:


выберете что делать: 6
Неверное значение! попробуйте еще раз
выберете что делать: 0
Программа закончилась
Спасибо за использование! ^_^
alina@Linchik2:~/курсовая работа версия 1файлы$

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: main.c

```
#include <stdio.h>
#include <locale.h>
#include <wchar.h>
#include <stdlib.h>
#define MEM_SIZE 50
#include "structs.h"
#include "sent_changes.h"
#include "read_and_err.h"
#include "cw_all_func.h"

// основная функция
int main(){
    setlocale(LC_ALL, "");
    Text_s text;
    text.count_sentence = 0;
    wprintf(L"\t\tВВЕДИТЕ ТЕКСТ\n");
    read_text(&text);
    del_replay_sentences(&text);
    wprintf(L"---------------ОБРАБОТАННЫЙ ТЕКСТ---------------\n");
    print_text(&text);
    user_selection(&text);
    for (int i = 0; i < text.count_sentence; i++) {
        Sentence_s  sentence = text.text_arr[i];
        free(sentence.sentence_arr);

    }
```

```
    free(text.text_arr);
    //wprintf(L"Всё удалено");
    return 0;
}
```

Файл: cw_all_func.c

```
#include <locale.h>
#include <wchar.h>
#include <stdlib.h>
#include "sent_changes.h"
#include "read_and_err.h"
#include "cw_all_func.h"
#include <bits/wctype-wchar.h>
#define BUF_SIZE 100
#define MEM_SIZE 50

// заполнение словаря словами текста
void create_dict_from_text(Text_s *text, Dictionary_s *once_words) {
    Dictionary_s *words = create_empty_dict(); // слова
    Dictionary_s *seps = create_empty_dict(); // сепараторы
    for (int i = 0; i < text->count_sentence; i++){
        add_dictionary_from_sent(words, seps, &text->text_arr[i]); // создаем словари
из слов и сепараторов предложения
    }
    for (int i = 0; i < words->count_words; i++){
        int flag = 1;
        for (int j = 0; j < once_words->count_words; j++){
            if (wcscasecmp(words->words_arr[i].word_arr, once_words-
>words_arr[j].word_arr) == 0) {
                once_words->words_arr[j].repeat++;
                flag = 0;
            }
        }
        if (flag == 1){
            add_word(once_words, words->words_arr[i].word_arr, words-
>words_arr[i].count_symbols);
        }
    }
    free_dict(words);
    free_dict(seps);
}
```

```
void print_dict(Dictionary_s *dict){
   wprintf(L"\t\tWORDS ONCE\n\n");
   for (int i = 0; i < dict->count_words; i++) {
      Word_s word = dict->words_arr[i];
      if (word.repeat == 1 ){
         wprintf(L"%ls\n", word.word_arr);
      }

   }
   wprintf(L"\t\t----------------------\n\n");
}

// функция, печатающая предложения и произведение длин слов в них
void text_multiplication(Text_s *text){
   wprintf(L"<------------------------------>\n\t\tSORTED\n\n");
   for (int i = 0; i < (*text).count_sentence; i++) {
      wprintf(L"произведение длин слов: <%ld>\t sentence <%ls>\n",
(*text).text_arr[i].multiplication,
            (*text).text_arr[i].sentence_arr);
   }
   wprintf(L"<------------------------------>\n");
}

//функция сравнения строк по значению произведения длин слов
int compare(const void *a, const void *b){
   Sentence_s sent_1 = *(Sentence_s *) a; // преобразовываю void * к sentence * и
разыменовываю
   Sentence_s sent_2 = *(Sentence_s *) b;
   if (sent_1.multiplication < sent_2.multiplication) {
      return -1;
   }
   if (sent_1.multiplication > sent_2.multiplication) {
      return 1;
   } else {
      return 0;
   }
}

// проверка на число
int is_number(wchar_t* str){
   for (int i = 0; str[i] != '\0'; i++){
      if (iswdigit(str[i]) == 0)
         return 0;
```

```
    }
    return 1;
}

// поиск месяца
int number_of_month(wchar_t* month){
    wchar_t months[][20] = {
            L"Января",
            L"Февраля",
            L"Марта",
            L"Апреля",
            L"Мая",
            L"Июня",
            L"Июля",
            L"Августа",
            L"Сентября",
            L"Октября",
            L"Ноября",
            L"Декабря"
    };
    for (int i = 0; i < 12; i++){
        if (wcscasecmp(month, months[i]) == 0){
            return i+1;
        }
    }
    return 0;
}

// проверка на корректность введенной даты
int is_correct_data(unsigned long int date_number, int month_number, unsigned long
int year_number){
    if ((1<=date_number <= 31) && (1<=year_number <= 9999)){
        int limit = 31;
        switch (month_number) {
            case 4:
                limit = 30;
                break;
            case 6:
                limit = 30;
                break;
            case 9:
                limit = 30;
                break;
            case 11:
```

```
            limit = 30;
            break;
        case 2:
            limit = 28;
            break;
        default:
            break;
    }
    return (date_number <= limit);
  }
}
// замена даты на требуемую
void replace_data(Text_s* text){
    for (int i = 0; i<text->count_sentence; i++){
        //разбиение
        Sentence_s* sent = &text->text_arr[i];
        Dictionary_s* words = create_empty_dict();
        Dictionary_s* seps = create_empty_dict();
        add_dictionary_from_sent(words, seps, sent);
        //обработка
        for (int j = 0; j < words->count_words-3; j++){
            Word_s date = words->words_arr[j];
            Word_s month = words->words_arr[j+1];
            Word_s year = words->words_arr[j+2];
            Word_s g = words->words_arr[j+3];
            if (wcscmp(g.word_arr, L"г") == 0) {
                if (is_number(date.word_arr) && is_number(year.word_arr)) {
                    int month_number = number_of_month(month.word_arr);
                    if (month_number != 0) {
                        wchar_t* end1 = NULL;
                        unsigned long int date_number = wcstoul(date.word_arr, &end1,
10); // 10 с/сч
                        wchar_t* end2 = NULL;
                        unsigned long int year_number = wcstoul(year.word_arr, &end2,
10);
                        if (is_correct_data(date_number, month_number, year_number)){

                            // дата
                            date.word_arr = realloc(date.word_arr, 3 * sizeof(wchar_t));
                            err(!date.word_arr);
                            swprintf(date.word_arr, 3, L"%02d", date_number);
                            date.count_symbols = 2;

                            // месяц
```

```
                    month.word_arr = realloc(month.word_arr, 3 * sizeof(wchar_t));
                    if (month.word_arr == NULL) {
                        err(1);
                    }
                    swprintf(month.word_arr, 3, L"%02d", month_number);
                    month.count_symbols = 2;

                    // год
                    year.word_arr = realloc(year.word_arr, 5 * sizeof(wchar_t));
                    if (year.word_arr == NULL) {
                        err(1);
                    }
                    swprintf(year.word_arr, 5, L"%04d", year_number);
                    year.count_symbols = 4;

                    // буква г
                    g.word_arr[0] = L'\0';
                    g.count_symbols = 0;

                    // слеши как разделители
                    Word_s sep1 = seps->words_arr[j];
                    Word_s sep2 = seps->words_arr[j+1];
                    Word_s sep3 = seps->words_arr[j+2];
                    wcscpy(sep1.word_arr, L"/");
                    sep1.count_symbols = 1;
                    wcscpy(sep2.word_arr, L"/");
                    sep2.count_symbols = 2;
                    sep3.word_arr[0] = L'\0';
                    sep3.count_symbols = 0;
                }
            }
        }
    }
    //склейка
    sent_from_dict(words,seps, sent);
    //очищаем после склейки
    free_dict(words);
    free_dict(seps);
  }
  print_text(text);
}

    //склейка
```

```
            sent_from_dict(words,seps, sent);
            //очищаем после склейки
            free_dict(words);
            free_dict(seps);
        }
    print_text(text);
}

// функция вывода текста
void print_text(Text_s *text) {
    for (int i = 0; i < text->count_sentence; i++) {
        wprintf(L"%ls\n", text->text_arr[i].sentence_arr);
    }
    wprintf(L"\n<------------------------------>\n");
}

// выбор пользователя (0-4)
void user_selection(Text_s *text) {
    wprintf(L"\n\nТекст считан\n\n");
    wprintf(L"Программа может выполнить следующие функции:\n"
            "1)Распечатать каждое слово которое встречается не более одного раза в
тексте.\n"
            "2)Каждую подстроку в тексте имеющую вид “<день> <месяц> <год> г.”
заменить на подстроку вида “ДД/ММ/ГГГГ”.\n"
            "3)Отсортировать предложения по произведению длин слов в
предложении.\n"
            "4)Удалить все предложения, которые содержат символ ‘#’ или ‘№’, но
не содержат ни одной цифры.\n"
            //"5) напечатать текст\n"
            "0)Выход из программы\n");
    long int way;
    Dictionary_s *dict_w = create_empty_dict();
    Dictionary_s *dict_s = create_empty_dict();
    Dictionary_s *once_words = create_empty_dict();
    wprintf(L"\nвыберете что делать: ");
    wscanf(L"%ld", &way);
        switch (way){
            case 1:
                create_dict_from_text(text, once_words);
                print_dict(once_words);

                break;
            case 2:
                replace_data(text);
```

```
            break;
        case 3:
            qsort((*text).text_arr, (*text).count_sentence, sizeof(Sentence_s),
compare);
            text_multiplication(text);
            break;
        case 4:
            del_special_sentence(text);
            break;
//          case 5:
//              print_text(text);
//              break;
        case 0:
            wprintf(L"Программа закончилась\nСпасибо за использование!\n");
            break;
        default:
            err(2);
            break;
    }
    free(dict_w);
    free(dict_s);
    free(once_words);
}
```

Файл: cw_all_func.h

```
#pragma once
#include "structs.h"

void user_selection(Text_s* text);
void print_text(Text_s* text);
```

Файл: read_and_err.c

```
#include <stdio.h>
#include <locale.h>
#include <wchar.h>
#include <stdlib.h>
#define BUF_SIZE 100
#define MEM_SIZE 50
#include "structs.h"
#include "read_and_err.h"

// функция ошибок
void err(int a) {
```

```
    switch (a) {
        case 1:
            wprintf(L"Ошибка! Выделение памяти невозможно!\n"); // если
невозможно выделить память
            break;
        case 2:
            wprintf(L"Неверное значение! попробуйте еще раз\n");// не та цифра
            break;
        default:
            break;
    }
}

//чтение предложения
wchar_t *read_sent(int *count_char, int *multiplication) {
    int len = 0; // количество введенных символов
    int size = BUF_SIZE; // буфер
    unsigned int c;
    long int pr = 1;
    int len_word = 0;
    int tabulation_start  = 1;
    wchar_t *str = malloc(size * sizeof(wchar_t));
    if (str == NULL) { // если 0, то ошибка переполнения памяти
        err(1);
    }
    do {
        c = getwchar();
        if (tabulation_start  == 1) {
            if (c == ' ') {
                continue;
            }else {
                tabulation_start  = 0;
            }
        }
        if (c != ',' && c != ' ' && c != '.') {
            len_word++;
        } else {
            if (len_word != 0) {
                pr = pr * len_word;
                len_word = 0;
            }
        }
        if (c == L'\n' && len == 0) {
            str[0] = c;
```

```
            *count_char = 1;
            return str;
        }
        //непосредственно считывание в массив
        str[len] = c;
        len++;
        if (len == size) {
            size += MEM_SIZE;
            str = (wchar_t *) realloc(str, size * sizeof(wchar_t));
            if (str == NULL) {
                err(1);
            }
        }
    } while (c != '.');
    *count_char = len; // длина предложения = количество введенных символов
    *multiplication = pr;
    str[len] = '\0';
    return str;
}

// чтение текста
void read_text(Text_s *text) {
    int real_size_text_arr = BUF_SIZE;
    text->text_arr = (Sentence_s *) malloc(real_size_text_arr * sizeof(Sentence_s));
    while (1) {
        Sentence_s Sentence;
        Sentence.sentence_arr = read_sent(&Sentence.count_char,
&Sentence.multiplication);
        if (Sentence.sentence_arr[0] == L'\n') { пользователь нажал \n
            free(Sentence.sentence_arr);
            break;
        }
        //добавили в массив предложений нашу структуру предложения
        (*text).text_arr[(*text).count_sentence] = Sentence; // так как переменная
структуры text с указателем
        (*text).count_sentence++;
        //расширение памяти для хранения структур предложений
        if ((*text).count_sentence == real_size_text_arr) {
            (*text).text_arr = realloc((*text).text_arr, (real_size_text_arr + MEM_SIZE) *
sizeof(Sentence_s));
            if ((*text).text_arr == NULL) {
                err(1);
            }
            real_size_text_arr += MEM_SIZE;
```

```
        }
    }
}

// создание пустого словаря
Dictionary_s *create_empty_dict(){
    Dictionary_s *empty_dict = malloc(sizeof(Dictionary_s));
    empty_dict->words_arr = NULL;
    empty_dict->count_words = 0;
    return empty_dict;
}

// очищение использованного словаря
void free_dict(Dictionary_s* dict){
    for (int i = 0; i < dict->count_words; i++) {
        // Word_s word = dict->words_arr[i];
        //free(word.word_arr);
        Word_s word = dict->words_arr[i];
         free(word.word_arr);
    }
    free(dict->words_arr);
    free(dict);
}

// функция добавления слов в словарь
void add_word(Dictionary_s *dict, wchar_t *word, int len) {
    Word_s *words_arr;
    if (dict->words_arr == NULL) {
        words_arr = malloc(sizeof(Word_s));
        dict->words_arr = words_arr;

    }else {
        //fwprintf(stderr, L"%d", my_dict->count_words+1);
        words_arr = realloc(dict->words_arr, (dict->count_words + 1) *
sizeof(Word_s));
        if (words_arr != NULL) {
            dict->words_arr = words_arr;
        }
        else{
            err(1);
        }
    }
    dict->words_arr[dict->count_words].count_symbols = len;
    dict->words_arr[dict->count_words].repeat = 1;
```

```
    wchar_t *str;
    str = calloc((len + 1), sizeof(wchar_t));
    dict->words_arr[dict->count_words].word_arr = str;
    if (dict->words_arr[dict->count_words].word_arr == NULL) {
        err(1);
    }
    //fwprintf(stderr, L"%d %d\n", wcslen(word), len+1);
    wcsncpy((dict->words_arr)[dict->count_words].word_arr, word, len);
    dict->count_words++;
}

// заполнение словаря словами предложения
void add_dictionary_from_sent(Dictionary_s *my_dict, Dictionary_s *dict_seps,
Sentence_s *sent) {
    wchar_t *sent_copy_w = malloc(sizeof(wchar_t) * (sent->count_char + 1)); //
копия для нарезки слов
    wchar_t *sent_copy_s = malloc(sizeof(wchar_t) * (sent->count_char + 1)); // дл
сепараторов
    wcscpy(sent_copy_w, sent->sentence_arr);
    wcscpy(sent_copy_s, sent->sentence_arr);
    if (sent_copy_w == NULL) {
        err(1);
    }
    if (sent_copy_s == NULL) {
        err(1);
    }
    wchar_t *ptr = NULL;
    wchar_t *old_ptr = NULL;
    wchar_t *word = wcstok(sent_copy_w, L" ,.\n\t", &ptr);
    while (word != NULL) {
        add_word(my_dict, word, ptr - word - 1/*wcslen(word)*/);
        // сохранение разделителей между словами
        if ((my_dict->count_words > 1) && old_ptr) {
            sent_copy_s[word - sent_copy_w] = '\0';
            add_word(dict_seps, (old_ptr - sent_copy_w + sent_copy_s - 1), (word -
old_ptr) + 1);
        }
        old_ptr = ptr;
        word = wcstok(NULL, L" ,.\n\t", &ptr); // обновляем слово
    }
    free(sent_copy_w);
    free(sent_copy_s);
}
```

```
// склеивание строки обратно
void sent_from_dict(Dictionary_s *dict_words, Dictionary_s *dict_seps, Sentence_s
*sent) {
   wcscpy(sent->sentence_arr, dict_words->words_arr[0].word_arr); // нулевое
слово
   if (dict_words->count_words > 1) {
      for (int i = 0; i < dict_seps->count_words; i++) {
         wcscat(sent->sentence_arr, dict_seps->words_arr[i].word_arr); // сепаратор
после нулевого слова
         wcscat(sent->sentence_arr, dict_words->words_arr[i + 1].word_arr); // слово
      }
   }
   wcscat(sent->sentence_arr, L"."); // последний сепаратор - точка
}
```

Файл: read_and_err.h

```
#pragma once
#include "structs.h"

void err(int a);
wchar_t* read_sent(int *count_char, int *multiplication);
void read_text (Text_s* text);
Dictionary_s *create_empty_dict();
void free_dict(Dictionary_s* my_dict);
void add_word(Dictionary_s * my_dict, wchar_t* word, int len);
void add_dictionary_from_sent(Dictionary_s* my_dict,  Dictionary_s *dict_seps,
Sentence_s* sent);
void sent_from_dict(Dictionary_s* my_dict,  Dictionary_s *dict_seps, Sentence_s*
sent);
```

Файл*: sent_changes.c*

```
#include <string.h>
#include <wchar.h>
#include <ctype.h>
#include "stdlib.h"
#define BUF_SIZE 100
#define MEM_SIZE 50
#include "structs.h"
#include "sent_changes.h"

// функция удаления повторяющихся строк
```

```
void del_replay_sentences(Text_s* text){
   for (int i = 0; i < (*text).count_sentence; i++){
      for (int j = i+1; j < (*text).count_sentence; j++){
         // ускорения программы, предложения будут сравниваться только при
одинаковом количестве символов
         if ((*text).text_arr[i].count_char == (*text).text_arr[j].count_char) {
            if (wcscasecmp((*text).text_arr[i].sentence_arr,
(*text).text_arr[j].sentence_arr) == 0) { // проверка на совпадение без учета
регистра
               free(text->text_arr[j].sentence_arr);
               memmove(((*text).text_arr) + j, ((*text).text_arr) + j + 1,
(((*text).count_sentence) - j) * sizeof(Sentence_s));
               (*text).count_sentence--;
               // т.к. предложение удалено, курсор остается на том же индексе
               j--;
            }
         }
      }
   }
}
// функция удаления предложений, содержащих ‘#’ или ‘№’, но не содержащих
цифр
void del_special_sentence(Text_s *text){
   int flag_symbol = 0;
   int flag_digit = 0;
   for (int i = 0; i < (*text).count_sentence; i++) {
      for (int j = 0; j < (*text).text_arr[i].count_char; j++) {
         if ((*text).text_arr[i].sentence_arr[j] == '#' || (*text).text_arr[i].sentence_arr[j]
== L'№') {
            flag_symbol = 1;
         }
         if (isdigit((*text).text_arr[i].sentence_arr[j])) {
            flag_digit = 1;
         }
      }
      if (flag_digit == 0 && flag_symbol == 1) {
         free(text->text_arr[i].sentence_arr);
         memmove((*text).text_arr + i, (*text).text_arr + i + 1,
(((*text).count_sentence) - i) * sizeof(Sentence_s));
         (*text).count_sentence--;
         i--;
      }
      flag_symbol = 0;
      flag_digit = 0;
```

```
    }
    wprintf(L"<------------------------------>\n");
    wprintf(L"\t\tAFTER DELETE\n\n");
    for (int i = 0; i < (*text).count_sentence; i++) {
        wprintf(L"%ls\n", (*text).text_arr[i].sentence_arr);
    }
    wprintf(L"<------------------------------>\n");
}
```

Файл: sent_changes.h

```
#pragma once
#include "structs.h"

void del_replay_sentences(Text_s* text);
void del_special_sentence(Text_s *text);
```

Файл: *structs.c*

```
#include <locale.h>
#include <wchar.h>
// структура для предложений
typedef struct Sentence{
    wchar_t* sentence_arr; // предложение
    int count_char; // длина предложения
    int multiplication; // произведение длин слов
}Sentence_s;

// структура для текста (массива предложений)
typedef struct Text{
    Sentence_s *text_arr; // указатели на начала предложений
    int count_sentence; // количество предложений в тексте
}Text_s;

typedef struct Word{
    wchar_t *word_arr; // указатели на начало слово
    int count_symbols; // длина слова
    int repeat; // количество повторений слова
} Word_s;

typedef struct Dictionary{
    Word_s *words_arr; // указатель на первое слово
    int count_words; // количество слов
```

```
}Dictionary_s;
```

Файл: structs.h

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

#pragma once
// структура для предложений
typedef struct Sentence {
    wchar_t *sentence_arr; // предложение
    int count_char; // длина предложения
    int multiplication; // произведение длин слов
} Sentence_s;

// структура для текста (массива предложений)
typedef struct Text {
    Sentence_s *text_arr; // указатели на начала предложений
    int count_sentence; // количество предложений в тексте
} Text_s;

typedef struct Word {
    wchar_t *word_arr; // указатели на начало слово
    int count_symbols; // длина слова
    int repeat; // мать повторений
} Word_s;

typedef struct Dictionary {
    Word_s *words_arr; // указатель на первое слово
    int count_words; // количество слов
} Dictionary_s;
```

Makefile:

```
all: cw

cw: main.o cw_all_func.o  read_and_err.o sent_changes.o structs.o
        gcc main.o cw_all_func.o read_and_err.o sent_changes.o structs.o -o main
main.o: main.c structs.h sent_changes.h read_and_err.h cw_all_func.h
        gcc -c main.c
structs.o: structs.c structs.h
        gcc -c  structs.c
sent_changes.o: sent_changes.c sent_changes.h
        gcc -c sent_changes.c
```

```
read_and_err.o: read_and_err.c read_and_err.h
	gcc -c  read_and_err.c
cw_all_func.o: cw_all_func.c cw_all_func.h
	gcc -c cw_all_func.c
remove:
	rm -f *.o main
```