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. 9381 _____ Москаленко Е.М.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Москаленко Е.М.

Группа 9381

Тема работы: Обработка текста.

Исходные данные:
Компилятор gcc, язык программирования C.

Содержание пояснительной записки:
1. Введение
2. Формулировка задания
3. Разработка программы
4. Тестирование
5. Заключение
6. Список использованных источников

Предполагаемый объем пояснительной записки:
Не менее 30 страниц.

Дата выдачи задания: 15.10.2019

Дата сдачи реферата: 19.12.2019

Дата защиты реферата: 19.12.2019

Студентка	_____	Москаленко Е.М.
Преподаватель	_____	Жангиров Т.Р.

АННОТАЦИЯ

Разработана программа обработки введенного пользователем текста, состоящего из латинских, кириллических букв, цифр и специальных символов. Предложения разделены точкой, слова - пробелом или запятой. Программа удаляет все повторяющиеся предложения, сравнивая их без учета регистра. Реализованы все подзадачи, обрабатывающие текст. Для сборки файлов проекта написан Makefile. Представлены тестирование и исходный код программы.

SUMMARY

The program of processing of the text entered by the user consisting of Latin, Cyrillic letters, numbers and special symbols is realized. Sentences are separated by a period, words by a space or comma. The program removes all duplicate sentences by comparing them case-insensitive. All subtasks processing text are implemented. Makefile was written for linking of project’s files. Testing and source code of the program are presented.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы.

Разработать программу, способную принимать и обрабатывать введённый текст в зависимости от команды пользователя.

Задачи.

Для достижения поставленных целей требуется:

1. Изучить синтаксис языка программирования C.
2. Реализовать подзадачи, заданные в условии курсовой работы.
3. Написать Makefile.
4. Произвести сборку проекта.
5. Протестировать программу.

1. ФОРМУЛИРОВКА ЗАДАНИЯ

Вариант 5

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 и количество его повторений в тексте.

2. Заменить каждый символ, который не является буквой, на его код.

3.Отсортировать предложения по количеству латинских букв в предложении.

4. Удалить все предложения, которые содержат специальные символы и не содержат заглавные буквы.

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. РАЗРАБОТКА ПРОГРАММЫ.

Подключены заголовочные файлы *stdio.h, wchar.h, locale.h, wctype.h, stdlib.h* в целях использования функций стандартной библиотеки языка С. Для хранения символов латиницы и кириллицы используется тип данных *wchar_t*. Для удобства считывания и хранения отдельного предложения и текста были реализованы структуры *Sentence* и *Text.*

Структура *Sentence* содержит следующие поля:

wchar_t* sent – массив широких символов, в который записывается введенная пользователем строка.

int act_size – счётчик количества символов в строке.

int max_size – возможное максимальное количество символов в предложении.

Структура *Text* содержит два поля:

wchar_t** buf – массив указателей на массив строк, хранящий предложения текста.

int number – количество предложений в введенном тексте.

Объявления обеих структур представлены в заголовочном файле *read.h.*

2.1. Функции считывания

2.1.1. Функция readSentence()

Sentence readSentence();

Функция реализована для считывания предложения с консоли. Инициализируется переменная *sentence* типа *Sentence*. Полю *sentence.max_size* изначально присваивается значение константы *INIT* = 10, определенной через макрос *define.* Счетчик количества символов *sentence.act_size* обнулен, а для массива символов *sentence.sent* с помощью функции стандартной библиотеки *malloc* выделяется количество памяти, равное *sentence.max_size * sizeof(wchar_t).* Считывается первый символ, и пока он равен табуляции или пробелу, продолжается считывание его следующего значения, никуда не записывая. Как только находится начало предложения, символ записывается в

массив по индексу *sentence.act_size*. Счетчик увеличивается на 1. Входим в цикл с постусловием, пока вводимый символ не равен ‘\n’ или ‘.’. Считывается следующий символ и записывается по индексу. Сравнивается текущее количество символов и максимальное возможное количество, и если первое превышает максимум, то максимум увеличивается в два раза и массиву символов выделяется больше памяти через realloc. После выхода из цикла записываем в массив ‘\0’, что означает конец предложения.

Функция возвращает sentence.

2.1.2. Функция readText()

Text readText ();

Функция предназначена для сохранения всех предложений. Инициализируются переменные:

int size = INIT – максимально возможное количество предложений.

int number = 0 – счетчик количества предложений.

Text text – переменная, хранящая сам текст и количество предложений.

Для *text.buf* выделяется память через *malloc*. В цикле инициализируется переменная *Sentence sentence,* присваивая ей значение считываемого предложения в *readSentence()*. Если первый символ предложения равен ‘\n’, то очищается память, выделенная предложению, и происходит выход из цикла. Иначе предложение копируется в *text.buf* по индексу *number* с помощью функции *wcscpy*. Очищается память *sentence.sent* и *number* увеличивается на 1. Также при необходимости увеличиваем *size* и количество памяти для текста. В конце *text.number* присваивается значение *number* и возвращается *text.*

2.2. Функции печати текста и удаления дубликатов предложений

Объявления обеих функций представлены в заголовочном файле PRandDEl.h.

2.2.1. Функция printTEXT()

void printTEXT(Text text);*

В качестве аргумента функция принимает переменную *text* типа *Text.* В цикле от 0 до *text.number* функция печатает на консоль элемент массива *text.buf* с помощью функции *wprintf.* Все предложения выводятся через пробел.

2.2.2. Функция delete()

void delete(Text text);*

Функция удаляет дубликаты введенных предложений. С помощью вложенного цикла сравнивает без учета регистра два предложения в массиве, используя функцию *wcsasecmp* из файла *wchar.h.* Если предложения равны, то в цикле, начиная с дубликата, сдвигаем предложения влево на один индекс. Количество предложений уменьшаем на 1. После процедуры удаления и выхода из цикла обновляем количество памяти, выделенное для текста, используя *realloc.*

2.3. Функции для подзадачи 1

Объявления всех функций находятся в заголовочном файле TASK1.h. Также объявлена структура для хранения всех слов текста Words. Поля:

*wchar_t** tokes* – массив указателей на массив строк со словами текста.

int amount – общее количество слов.

2.3.1. Функция countWords()

int countWords(Text text);*

Функция нужна исключительно для того, чтобы найти возможный максимум количества слов в тексте. В цикле в каждом предложении посчитывается количество пробелов, табуляций (с помощью *iswblank* из файла *wctype.h*) и запятых. Возвращается это количество.

2.3.2. Функция cmpWords()

int cmpWords(const void str, const void* tok);*

Функция-компаратор, лексикографически сортирующая слова в массиве с помощью функции широкосимвольных строк *wcscmp*. Принимает на вход два указателя на элементы массива.

2.3.3. Функция printWords()

void printWords(Words words);*

Функция, предназначена для сортировки слов в массиве и подсчёта вхождения в него каждого слова.

Сначала происходит сортировка слов в лексикографическом порядке с помощью библиотечной функции быстрой сортировки *qsort*. Функция-компаратор – *cmpWords*(см.2.3.2). Инициализируется переменная count, изначально равная 1 – счётчик вхождения слова. В цикле от 0 до *words->amount* проверяем, совпадает ли этот элемент массива с предыдущим. Если нет – то переменная count не изменяется, переходим к следующей итерации. В противном случае во вложенном цикле начиная с индекса больше на 1 индекса рассматриваемого элемента смотрим, идут ли за ним такие же слова. При каждом совпадении увеличиваем count на 1.

Функция выводит строку:

“<слово>” в тексте <*count*> раз.

Затем *count* вновь присваивается значение 1 и функция переходит к новой итерации цикла.

2.3.4. Функция freeMemoryWORDS()

void freeMemoryWORDS(Words words);*

Функция возвращает память, выделенную для массива слов *words.tokes*, в кучу. Принимает в качестве аргумента указатель на структуру типа *Words* и с помощью *free* очищает память всех элементов поля *words.tokes*.

2.3.5. Функция TASK1()

void TASK1(Text text);*

Функция для реализации 1 задания. Переменной *count* присваиваем возвращаемое значение функции *countWords(*text)*. Так как для доступа к каждому слову необходимо воспользоваться функцией *wcstok*, а после нее невозможно продолжать работу с текстом, то все предложения нужно скопировать в отдельный массив и каждый раз при выполнении 1 задания работать только с ним. Создаём массив *wchar_t** buffer*, выделяя ему память и в цикле копируя в него каждое предложение из *text.buf*. Далее объявляем переменную *words* типа *Words*, выделяя *words.tokes* блок памяти размером *count * sizeof(wchar_t*)*. Инициализируем счетчик количества слов. В цикле каждый элемент *buffer* разбивается на токи, используя в качестве разделителей ‘.’, ’,’ , ‘ ‘, ‘\t’. Все получившиеся слова копируем в массив *words.tokes* и с каждой итерацией увеличиваем счетчик на 1. Полю *words.amount* присваивается получившееся значение счетчика, объём памяти *words.tokes* изменяется с помощью *realloc*. Затем вызываются функции *printWords(&words)* и *freeMemoryWORDS(&words)* и через цикл очищается память вспомогательного массива *buffer*.

2.4. Функция TASK2()

void TASK2(Text text);*

Функция для реализации подзадачи №2. Инициализируем строку широких символов *out* длиной 100 – этого будет достаточно, чтобы записывать в неё коды символов. В цикле от первого до последнего предложения пробегаемся по символам каждого. В функции *count* (изначально = 1) – длина кода символа, который не является буквой или разделителем. Каждую итерацию, проходя по конкретному предложению, прибавляем к индексу *count-1*, чтобы перешагнуть ‘код’ уже замененного символа. Если получившийся индекс выходит за границу предложения, то выходим из цикла и приступаем к обработке следующего предложения. Иначе проверяем символ, является ли он цифрой или специальным знаком (не является буквой), и если условие верно, то присваиваем переменной *int k* значение его кода (код находим с помощью приведения символа к типу *int*). Удаляем символ сдвигом. Находим длину кода

символа (1, 2, 3) путем сравнения с 100 и 9. Изменяем длину предложения, прибавляя *count*, а затем увеличиваем блок памяти, выделенный для самой строки.

Используя функцию *swprintf* осуществляем вывод кода *(k)* в строку *out* (так как необходимо перевести целое число в массив символов). Затем двумя вызовами функции копирования *wcscpy* сначала «соединяем» код символа и остаток строки после удалённого символа, после – первую часть предложения до кода и получившуюся строку.

```
wcscpy(out  + count, text->[buf] + i);
wcscpy(text->[buf] + i, out);
```

Так как один символ был удален, а *count* символов вставлены в строку, длина предложения увеличилась на *count-1*, поэтому необходимо изменить значение переменной, хранящей длину строки.

После выхода из цикла вызывается функция *printTEXT* и на консоль выводится отформатированный текст.

2.5. Функции для подзадачи 3

Объявления всех функций представлены в заголовочном файле TASK3.h.

2.5.1. Функция letters()

int letters(wchar_t a);*

В качестве аргумента принимает массив широких символов. Функция ведёт счётчик латинских букв в предложении. Проходя по символам предложения в цикле, проверяем, является ли символ буквой, используя функцию *iswalpha* из файла *wctype.c*. Буква должна быть обязательно латинской, поэтому добавляем условие на проверку кода ASCII – от 65 до 122 включительно. Если символ удовлетворяет условиям, то увеличиваем счетчик на 1. Функция возвращает количество латинских букв той или иной строки.

2.5.2. Функция cmp()

int cmp(const void str, const void*tok);*

Функция-компаратор для сортировки строк по возрастанию количества латинских букв. В качестве аргументов принимаются указатели на два блока

памяти, которые приводятся к строковому типу данных. Инициализируются переменные *countA* и *countB,* значения которых – возвращаемые значения функции *letters()* , аргументы которой – две полученные строки соответственно. В зависимости от разницы *countA* и countB функция возвращает 1, 0 или -1.

2.5.3. Функция TASK3()

void TASK3(Text text);*

Функция сортирует предложения по количеству латинских букв с помощью быстрой сортировки *qsort.* Компаратором является функция *cmp.* После сортировки вызывается *printTEXT* и текст печатается на консоль.

2.6. Функция TASK4()

void TASK4(Text text);*

Функция для реализации подзадачи №4. Проходя в цикле по всем предложениям текста, считает количество специальных символов (с помощью *iswpunct*, с учётом того, что символ не может быть точкой или запятой) и заглавных букв (*iswupper*). Если первое количество больше 0, а второе равно 0, то удаляем предложение сдвигом, количество предложений уменьшаем на 1. Последнее предложение текста необходимо обработать отдельно, так как если оно удовлетворяет условиям, то его нужно заменить на символ ‘\0’, а не сдвигать.

После обработки текста вызывается функция *printTEXT,* и отформатированный текст выводится на консоль.

2.7. Функция freeMemoryTEXT()

void freeMemoryTEXT(Text text);*

Объявление функции представлено в заголовочном файле *FREE.h.* Функция реализована для возврата в кучу количества памяти, выделенной для *text.buf* – массива указателей на предложения текста. В цикле память каждого предложения освобождается с помощью функции стандартной библиотеки *free,* а затем очищается блок, выделенный для всего массива.

2.8. Функция print_menu()

void print_menu();

Объявление функции представлено в заголовочном файле menu.h.

В программе вызывается единственный раз для ознакомления пользователя с командами. Выводит на консоль 6 строк с возможными действиями.

2.9. Функции для обработки и выполнения команды пользователя.

Определения функций находятся в файле main.c.

2.9.1. Функция get()

Функция реализована для обработки действия, выбранного пользователем. Объявляются переменные:

int input - введенное пользователем число.

char ch - введенный пользователем символ (если не было введено число).

Входим в цикл с предусловием, пока input не считано. Если условие истинно, то входим в следующий цикл, пока символ ch не равен символу перевода строки. Очищаем буфер, выводя неверный символ, и просим пользователя выбрать действие заново. Возвращаем целое число, как только оно будет получено на вход.

2.9.2. Функция main()

Функция реализована для считывания текста и возможности выбора пользователем действия. С помощью функции *setlocale* из заголовочного файла *locale.h* подключаем символы других алфавитом, тем самым допуская возможность использования кириллических букв.

```
setlocale(LC_CTYPE,"");
```

Объявляем переменную *Text text*, присваивая ей значения, возвращаемое функцией *readText*. Затем вызываем функцию *delete*, в качестве аргумента передавая адрес *text*. Функция удаляет дубликаты предложений без учета регистра. Далее печатаем меню с помощью *print_menu()*, предлагая пользователю ознакомиться с действиями. Затем входим в бесконечный цикл, в

котором через оператор *switch* проверяем значение вводимого с каждой итерацией действия в функции *get()*.

В случае 1/2/3/4 - вызывается функция *TASK1/2/3/4(&text)*. В случае 5 - *printTEXT(&text)*, которая просто печатает текст на экран. В случае 6 – программа прощается с пользователем, очищает память текста и предварительно выходит из программы. В противном случае предупреждает пользователя о неверно введенных данных и входит в следующую итерацию цикла.

2.10. Makefile

Для сборки файлов проекта компиляции программы был реализован make-файл. По умолчанию выполняется цель *all*, выполняющая полную компиляцию и линковку всех исходных файлов и удаление образовавшихся объектных файлов. Для компиляции программы вызывается цель *prog*, для очистки рабочей директории от объектных файлов – цель *clean*.

3. ТЕСТИРОВАНИЕ ПРОГРАММЫ

3.1. Тестирование работы первой подзадачи

Для выбора действия введите текст:

>> Hello, everyone. Меня зовут Елизавета, но друзья зовут меня Лиза. Я учусь в ЛЭТИ с 2019 года. Goodbye, everyone.

Выберите действие:

1. Распечатать количество повторений каждого слова в тексте.

2. Заменить каждый символ, который не является буквой, на его код.

3. Отсортировать предложения по количеству латинских букв.

4. Удалить все предложения со специальными символами и без заглавных букв.

5. Вывести текст на экран.

6. Выйти из программы.

>> 1

"2019" в тексте 1 раз

"Goodbye" в тексте 1 раз

"Hello" в тексте 1 раз

"everyone" в тексте 2 раз

"Елизавета" в тексте 1 раз

"ЛЭТИ" в тексте 1 раз

"Лиза" в тексте 1 раз

"Меня" в тексте 1 раз

"Я" в тексте 1 раз

"в" в тексте 1 раз

"года" в тексте 1 раз

"друзья" в тексте 1 раз

"зовут" в тексте 2 раз

"меня" в тексте 1 раз

"но" в тексте 1 раз

"с" в тексте 1 раз

"учусь" в тексте 1 раз

Тест пройден.

3.2. Тестирование работы второй подзадачи

Для выбора действия введите текст:

>> Password for this site is 36Wl*&j!s. Пожалуйста, запомните эту комбинацию из 9 символов.

Выберите действие:

1. Распечатать количество повторений каждого слова в тексте.

2. Заменить каждый символ, который не является буквой, на его код.

3. Отсортировать предложения по количеству латинских букв.

4. Удалить все предложения со специальными символами и без заглавных букв.

5. Вывести текст на экран.

6. Выйти из программы.

>> 2

Password for this site is 5154Wl4238j33s. Пожалуйста, запомните эту комбинацию из 57 символов.

Тест пройден.

3.3. Тестирование работы третьей подзадачи

Для выбора действия введите текст:

>> Hello I study at LETI. Привет, я учусь в ЛЭТИ. Hello я учусь в ЛЭТИ. Hello I учусь at LETI. Привет, я учусь в LETI.

Выберите действие:

1. Распечатать количество повторений каждого слова в тексте.

2. Заменить каждый символ, который не является буквой, на его код.

3. Отсортировать предложения по количеству латинских букв.

4. Удалить все предложения со специальными символами и без заглавных букв.

5. Вывести текст на экран.

6. Выйти из программы.

>>3

Привет, я учусь в ЛЭТИ. Привет, я учусь в LETI. Hello я учусь в ЛЭТИ. Hello I учусь at LETI. Hello I study at LETI.

Тест пройден.

3.4. Тестирование работы четвертой подзадачи

Для выбора действия введите текст:

>> Я помню чудное мгновенье. передо мной явилась ты)). Как мимолетное мгновенье. как гений чистой красоты *.

Выберите действие:

1. Распечатать количество повторений каждого слова в тексте.

2. Заменить каждый символ, который не является буквой, на его код.

3. Отсортировать предложения по количеству латинских букв.

4. Удалить все предложения со специальными символами и без заглавных букв.

5. Вывести текст на экран.

6. Выйти из программы.

>> 4

Начальное количество предложений 4.

Я помню чудное мгновенье. Как мимолетное мгновенье.

Тест пройден.

3.5. Тестирование 5 команды – вывода текста

Для выбора действия введите текст:

>> Главный герой — 27-летний Илья Горюнов, семь лет отсидевший в тюрьме по ложному обвинению в распространении наркотиков. Когда Илья выходит на свободу, он понимает, что прежняя жизнь, по которой он тосковал, разрушена, и вернуться к ней он больше не сможет. Хотя он не собирался мстить человеку, который отправил его в тюрьму, другого выхода теперь нет.

Выберите действие:

1. Распечатать количество повторений каждого слова в тексте.

2. Заменить каждый символ, который не является буквой, на его код.

3. Отсортировать предложения по количеству латинских букв.

4. Удалить все предложения со специальными символами и без заглавных букв.

5. Вывести текст на экран.

6. Выйти из программы.

>> 5

Главный герой — 27-летний Илья Горюнов, семь лет отсидевший в тюрьме по ложному обвинению в распространении наркотиков. Когда Илья выходит на свободу, он понимает, что прежняя жизнь, по которой он тосковал,

разрушена, и вернуться к ней он больше не сможет. Хотя он не собирался мстить человеку, который отправил его в тюрьму, другого выхода теперь нет.

Тест пройден.

3.6. Совместное тестирование всех подзадач с выходом из программы и обработкой исключительных ситуаций

Для выбора действия введите текст:

>> Это последний тест проекта курсовой работы. The number of test is 6. Я студент группы 9381 и защищаю курсовую работу 19 декабря. Это последний тест проекта курсовой работы. do i wanna kmow?. No, I don't. Все будет good. Студенты группы 9381 защищают курсовую.

Выберите действие:

1. Распечатать количество повторений каждого слова в тексте.

2. Заменить каждый символ, который не является буквой, на его код.

3. Отсортировать предложения по количеству латинских букв.

4. Удалить все предложения со специальными символами и без заглавных букв.

5. Вывести текст на экран.

6. Выйти из программы.

>> 5

Это последний тест проекта курсовой работы. The number of test is 6. Я студент группы 9381 и защищаю курсовую работу 19 декабря. do i wanna kmow?. No, I don't. Все будет good. Студенты группы 9381 защищают курсовую.

>> 4

Начальное количество предложений 7.

Это последний тест проекта курсовой работы. The number of test is 6. Я студент группы 9381 и защищаю курсовую работу 19 декабря. No, I don't. Все будет good. Студенты группы 9381 защищают курсовую.

>> 3

Это последний тест проекта курсовой работы. Я студент группы 9381 и защищаю курсовую работу 19 декабря. Студенты группы 9381 защищают курсовую. Все будет good. No, I don't. The number of test is 6.

>> 2

Это последний тест проекта курсовой работы. Я студент группы 57515649 и защищаю курсовую работу 4957 декабря. Студенты группы 57515649 защищают курсовую. Все будет good. No, I don39t. The number of test is 54.

>> 1

"4957" в тексте 1 раз

"54" в тексте 1 раз

"57515649" в тексте 2 раз

"I" в тексте 1 раз

"No" в тексте 1 раз

"The" в тексте 1 раз

"don39t" в тексте 1 раз

"good" в тексте 1 раз

"is" в тексте 1 раз

"number" в тексте 1 раз

"of" в тексте 1 раз

"test" в тексте 1 раз

"Все" в тексте 1 раз

"Студенты" в тексте 1 раз

"Это" в тексте 1 раз

"Я" в тексте 1 раз

"будет" в тексте 1 раз

"группы" в тексте 2 раз

"декабря" в тексте 1 раз

"защищаю" в тексте 1 раз

"защищают" в тексте 1 раз

"и" в тексте 1 раз

"курсовой" в тексте 1 раз

"курсовую" в тексте 2 раз

"последний" в тексте 1 раз

"проекта" в тексте 1 раз

"работу" в тексте 1 раз

"работы" в тексте 1 раз

"студент" в тексте 1 раз

"тест" в тексте 1 раз

>> 9

Данные некорректны. Выберите действие от 1 до 6.

p

p - неверно. Вы должны ввести целое число.

>> 6

До свидания.

Все команды обработаны и выполнены верно. Тест пройден.

ЗАКЛЮЧЕНИЕ

Была реализована программа, считывающая текст с консоли и форматирующая его. Программа удаляет все повторяющиеся предложения и выполняет действие от 1 до 6, введенное пользователем:

1. Распечатать каждое слово и количество его повторений в тексте.
2. Заменить каждый символ, который не является буквой, на его код.
3. Отсортировать предложения по количеству латинских букв в предложении.
4. Удалить все предложения, которые содержат специальные символы и не содержат заглавные буквы.
5. Вывести текст на экран.
6. Выйти из программы.

При неверно введенном значении действия программа просит ввести пользователя действие еще раз. Выход из программы происходит при выборе действия 6, до этого пользователь может вызывать какое-либо действие неограниченное количество раз. Был написан Makefile и собран проект при помощи компилятора gcc. Проведено тестирование, обработаны все исключительные случаи. Все поставленные цели реализованы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кенриган Б. И Ритчи Д. Язык программирования Си М.: Вильямс, 1978 288 с.

2. wctype.h // wikipedia.org. URL: https://ru.wikipedia.org/wiki/Wctype.h (дата обращения: 29.11.2019).

3. Справочник по библиотечным функциям языка Си // codenet.ru. URL: http://www.codenet.ru/progr/cpp/sprd/ (дата обращения: 03.12.2019)

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл read.h:

```
#ifndef READ_H
#define READ_H

typedef struct Sentence{
    wchar_t* sent;
    int act_size;
    int max_size;
}Sentence;

typedef struct Text{
    wchar_t** buf;
    int size;
    int number;
}Text;

Sentence readSentence();
Text readText();
#endif
```

Файл PRandDEl.h:

```
#ifndef PRandDEl_H
#define PRandDEl_H
#include "read.h"
void printTEXT(Text* text);
void delete(Text* text);
#endif
```

Файл TASK1.h:

```
#ifndef TASK1_H
#define TASK1_H
#include "read.h"

typedef struct Words{
    wchar_t** tokes;
    int amount;
}Words;

int countWords(Text text);
int cmpWords (const void* str, const void* tok);
void printWords(Words* words);
void freeMemoryWORDS(Words* words);
void TASK1(Text* text);
#endif
```

Файл TASK2.h:

```
#ifndef TASK2_H
#define INDEX_FIRST_ZERO_H
void TASK2(Text* text);
#endif
```

Файл TASK3.h:

```
#ifndef TASK3_H
#define TASK3_H
int letters(wchar_t* a);
wchar_t cmp (const void* str, const void* tok);
void TASK3(Text* text);
#endif
```

Файл TASK4.h:

```
#ifndef TASK4_H
#define TASK4_H
#include "read.h"
void TASK4(Text* text);
#endif
```

Файл FREE.h:

```
#ifndef FREE_H
#define FREE_H
#include "read.h"
void freeMemoryTEXT(Text* text);
#endif
```

Файл menu.h:

```
#ifndef MENU_H
#define MENU_H
void print_menu();
#endif
```

Файл read.c:

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

```
#include "read.h"
#define INIT 10

Sentence readSentence(){
    int max_size_sentence = INIT;
    int act_size_sentence = 0;
    wchar_t* sent = malloc(max_size_sentence * sizeof(wchar_t));
    wchar_t c = getwchar();
    while((c == L'\t') || (c == L' ')){
        c = getwchar();
    }
    sent[act_size_sentence++] = c;
    do{
        c = getwchar();
        sent[act_size_sentence] = c;
        act_size_sentence++;
        if (act_size_sentence == max_size_sentence - 2){
            max_size_sentence += INIT;
            sent = realloc(sent, max_size_sentence * sizeof(wchar_t));
        }
    }while(!wcschr(L".\n", c));
    sent[act_size_sentence] = L'\0';
    Sentence sentence;
    sentence.sent = sent;
    sentence.act_size = act_size_sentence;
    sentence.max_size = max_size_sentence;
    return sentence;
}


Text readText(){
    int size = INIT;
    int number = 0;
    Text text;
    text.buf = malloc(size * sizeof(wchar_t*));
    while(1){
        struct Sentence sentence = readSentence();
        if (sentence.sent[0] == L'\n') {
            free(sentence.sent);
            break;
        }
        else {
            text.buf[number] = malloc(sizeof(wchar_t) *
(sentence.act_size + 10));
            for (int j = 0; j <= sentence.act_size + 1; j++)
                text.buf[number][j] = sentence.sent[j];
            free(sentence.sent);
            number++;
            if (number == size - 2) {
                size += INIT;
                text.buf = realloc(text.buf, size * sizeof(wchar_t *));
            }
        }
    }
    text.size = size;
    text.number = number;
    return text;
```

```
}
```

Файл PRandDEl.c:

```
#include <stdio.h>
#include <wchar.h>
#include <stdlib.h>
#include "PRandDEl.h"
#include "read.h"

void printTEXT(Text* text){
    for (int i = 0; i < text->number; i++)
        wprintf(L"%ls ",text->buf[i]);
    printf("\n");
}

void delete(Text* text) {
    for (int i = 0; i < text->number; i++) {
        for (int j = i + 1; j < text->number; j++) {
            if (wcscasecmp(text->buf[i], text->buf[j]) == 0) {
                for (int k = j; k < text->number - 1; k++)
                    text->buf[k] = text->buf[k + 1];
                (text->number)--;
                j = i;
            }
        }
    }
    text->buf = realloc(text->buf, text->number * sizeof(wchar_t*));
}
```

Файл TASK1.c:

```
#include <stdio.h>
#include <wctype.h>
#include <wchar.h>
#include <stdlib.h>
#include "TASK1.h"
#include "read.h"

int cmpWords (const void* str, const void* tok)
{
    return wcscmp(*(const wchar_t**)str, *(const wchar_t**)tok);
}

int countWords(Text text){
    int count = 0;
    for( int index  = 0; index < text.number; index++) {
        for (int i = 0; i < wcslen(text.buf[index]); i++) {
            if (iswblank(text.buf[index][i]) || text.buf[index][i] ==
L',') {
                count++;
            }
```

```
        }
        count++;
    }
    return count;
}

void printWords(Words* words) {
    qsort(words->tokes, words->amount, sizeof(wchar_t *), cmpWords);
    int count = 1;
    for (int j = 0; j < words->amount; j++) {
        if (j > 0 && wcscmp(words->tokes[j],words->tokes[j - 1]) == 0){
            continue;}
        else {
            for (int k = j + 1; k < words->amount; k++) {
                if (wcscmp(words->tokes[j], words->tokes[k]) == 0) {
                    count++;
                }
            }
            wprintf(L" \"%ls\"  в тексте %d раз \n", words->tokes[j] ,
count);
            count = 1;
        }
    }
}

void freeMemoryWORDS(Words* words){
    for (int i = 0; i < words->amount; i++)
        free(words->tokes[i]);
    free(words->tokes);
}

void TASK1(Text* text) {
    int count = countWords(*text);
    wchar_t** buffer = malloc(text->number * sizeof(wchar_t*));
    for(int i = 0; i < text->number; i++){
        int k = wcslen(text->buf[i]);
        buffer[i] = malloc((k + 100)* sizeof(wchar_t));
       text->buf[i][k] = L'\0';
       wcscpy(buffer[i], text->buf[i]);
    }
    Words words;
    words.tokes = malloc(count * sizeof(wchar_t *));
    int n = 0;
    wchar_t *tmp;
    for (int i = 0; i < text->number; i++) {
        wchar_t *tok = wcstok(buffer[i], L" \t.,", &tmp);
        while (tok != NULL) {
            int h = wcslen(tok);
            words.tokes[n] = malloc((h + 100) * sizeof(wchar_t));
            wcscpy(words.tokes[n], tok);
            n++;
            tok = wcstok(NULL, L" .,\t", &tmp);
        }
    }
    words.amount = n;
    printWords(&words);
    freeMemoryWORDS(&words);
```

```
    for (int i = 0; i < text->number; i++)
        free(buffer[i]);
    free(buffer);
}
```

Файл TASK2.c

```
#include <stdio.h>
#include <wchar.h>
#include <stdlib.h>
#include "PRandDEl.h"
#include "TASK2.h"
#include "read.h"

void TASK2(Text* text){
    wchar_t out[100];
    int k;
    for (int f = 0; f < text->number; f++) {
        int m = wcslen(text->buf[f]);
        int count = 1;
        for (int i = 0; i < m; i++) {
            i = i - 1 + count;
            if (i >= wcslen(text->buf[f]))
                break;
            count = 1;
            if (!iswalpha(text->buf[f][i]) && !iswspace(text->buf[f][i])
&&
                text->buf[f][i] != '.' && text->buf[f][i] != ',') {
                k = (int) text->buf[f][i];
                for (int j = i; j < wcslen(text->buf[f]); j++)
                    text->buf[f][j] = text->buf[f][j + 1];
                if (k >= 100) {
                    count = 3;
                } else {
                    if (k < 100 && k > 9)
                        count = 2;
                    text->buf[f] = realloc(text->buf[f],
sizeof(wchar_t)*(m+count));
                    swprintf(out, 100, L"%d", k);
                    wcscpy(out + count, text->buf[f] + i);
                    wcscpy(text->buf[f] + i, out);
                    m += count - 1;
                }
            }
        }
    }
     printTEXT(text);
}
```

Файл TASK3.c:

```
#include <wchar.h>
#include <stdlib.h>
#include "PRandDEl.h"
```

```
#include "TASK3.h"
#include "read.h"

int letters(wchar_t* a){
    int count = 0;
    for (int i = 0; i <= wcslen(a); i++){
         if (iswalpha(a[i]) && (int)a[i] >= 65 && (int)a[i] <= 122)
            count++;
    }
    return count;
}

int cmp (const void* str, const void* tok){
    int countA = letters(*(wchar_t **)str);
    int countB = letters(*(wchar_t **)tok);
    if (countA - countB > 0)
        return 1;
    if (countA - countB == 0)
        return 0;
    if (countA - countB < 0)
        return -1;
}


void TASK3(Text* text){
    qsort(text->buf, text->number, sizeof(wchar_t *), cmp);
    printTEXT(text);
}
```

Файл TASK4.c:

```
#include <stdio.h>
#include <wchar.h>
#include <wctype.h>
#include <ctype.h>
#include <stdlib.h>
#include "TASK4.h"
#include "read.h"
#include "PRandDEl.h"

void TASK4(Text* text) {
    printf("Начальное количество предложений %d.\n", text->number);
    for (int j = 0; j < text->number; j++) { //проходимся по
предложениям
        int countN = 0;
        int Nlast = 0;
        int countB = 0;
        int Blast = 0;
        for (int i = 0; i < wcslen(text->buf[j]); i++) {

            if (ispunct(text->buf[j][i]) && text->buf[j][i] != '.' &&
text->buf[j][i] != ',') {
                if (j == text->number - 1) {
                    Nlast++;
                } else
```

```
                    countN++;
            }

            if (iswupper(text->buf[j][i])) {
                if (j == text->number - 1) {
                    Blast++;
                } else
                    countB++;
            }

        }
        if (countN != 0 && countB == 0) {
            for (int k = j; k < (text->number) - 1; k++) {
                text->buf[k] = text->buf[k + 1];
            }
            (text->number)--;
            j--;
        }
        if (Nlast != 0 && Blast == 0) {
            text->buf[text->number - 1] = L"\0";
        }
    }
    printTEXT(text);
}
```

Файл menu.c:

```
#include <stdio.h>
#include "menu.h"

void print_menu() {
    printf("\nВыберите действие:\n");
    printf("1. Распечатать количество повторений каждого слова в
тексте.\n");
    printf("2. Заменить каждый символ, который не является буквой, на
его код.\n");
    printf("3. Отсортировать предложения по количеству латинских
букв.\n");
    printf("4. Удалить все предложения со специальными символами и без
заглавных букв.\n");
    printf("5. Вывести текст на экран.\n");
    printf("6. Выйти из программы.\n\n");
}
```

Файл FREE.c:

```
#include <stdlib.h>
#include "FREE.h"
#include "read.h"

void freeMemoryTEXT(Text* text){
    for (int i = 0; i < text->number; i++)
        free(text->buf[i]);
```

```
    free(text->buf);
}
```

Файл main.c:

```
#include <stdio.h>
#include <wchar.h>
#include <locale.h>
#include <ctype.h>
#include <wctype.h>
#include <stdlib.h>
#include "TASK1.h"
#include "TASK2.h"
#include "TASK3.h"
#include "TASK4.h"
#include "TASK1.h"
#include "FREE.h"
#include "PRandDEl.h"
#include "read.h"
#include "menu.h"

int get()
{
 int input;
 char ch;
 while (scanf("%d", &input) != 1)
 {
 while ((ch = getchar()) != '\n')
         putchar(ch); // отбросить неправильный ввод
         printf(" - неверно. Вы должны ввести целое число.\n ");
 }
 return input;
}

int main() {
    setlocale(LC_CTYPE, "");  //rus_rus.866
    printf("Для выбора действия введите текст:\n");
    Text text = readText();
    delete(&text);
    print_menu();
    while (1){
        switch (get()) {
           case 1:
                TASK1(&text);
                break;
            case 2:
                TASK2(&text);
                break;
            case 3:
                TASK3(&text);
                break;
            case 4:
                TASK4(&text);
                break;
            case 5:
                printTEXT(&text);
```

```
                break;
            case 6:
                printf("До свидания.");
                freeMemoryTEXT(&text);
                return 0;
            default:
                printf("Данные некорректны. Выберите действие от 1 до
6.\n");
                break;
        }
    }
    return 0;
}
```

Файл Makefile:

```
all: prog
prog: main.o FREE.o PRandDEl.o TASK2.o TASK1.o TASK3.o TASK4.o menu.o
read.o
        gcc -o cw main.o FREE.o PRandDEl.o TASK1.o TASK2.o TASK3.o
TASK4.o menu.o read.o
main.o: main.c
        gcc -c main.c
read.o: read.c
        gcc -c read.c
PRandDEL.o: PRandDEl.c
        gcc -c PRandDEl.c
menu.o: menu.c
        gcc -c menu.c
FREE.o: FREE.c
        gcc -c FREE.c
TASK1.o: TASK1.c
        gcc -c TASK1.c
TASK2.o: TASK2.c
        gcc -c TASK2.c
TASK3.o: TASK3.c
        gcc -c TASK3.c
TASK4.o: TASK4.c
        gcc -c TASK4.c
clean:
        rm -rf *.o
```