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. 2300 _____ Локосов Д.Д.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2022

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Локосов Д.Д.

Группа 2300

Тема работы: Обработка текстовых данных

Исходные данные:
Вариант 13

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 неизвестна.

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. Сделать сдвиг слов в предложении на положительное целое число N. Например, предложение “abc b#c ИЙ два” при N = 2 должно принять вид “ИЙ два abc b#c”.

2. Вывести все уникальные кириллические и латинские символы в тексте.

3. Подсчитать и вывести количество слов (плюс слова в скобках), длина которых равна 1, 2, 3, и т.д.

4. Удалить все слова, которые заканчиваются на заглавный символ.

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.

Предполагаемый объем пояснительной записки:
Не менее 16 страниц.

Дата выдачи задания: 20.10.2022

Дата сдачи реферата: 17.12.2022

Дата защиты реферата: 19.12.2022

Студент _____ Локосов Д.Д.

Преподаватель _____ Чайка К.В.

АННОТАЦИЯ

Курсовая работа представляет собой реализацию программы обработки текстовых данных, вводимых пользователем, на языке программирования C.

Для хранения текста использовались структуры и динамическая память, для работы с ним были задействованы функции стандартной библиотеки языка.

В результате написана программа, считывающая текст из консоли, после чего разбивающая его на предложения. Далее она предлагает пользователю несколько возможных вариантов обработки введённого текста, предварительно напечатав подсказку, также предусмотрена возможность выхода из программы и ввод некорректных данных. Программа выполняет выбранную команду, выводя результат обработки. При выходе вся выделенная динамическая память освобождается.

Для включения программы требуется зайти в корневую папку с проектом, ввести в консоль *make && ./cw_exe*. Далее нужно ввести текст в одну строку и выбрать действие в соответствии с выведенной подсказкой путём ввода числа. Для выхода нужно ввести 0.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является освоение подходов к работе с текстовыми данными в языке С и последующая реализация программы для их обработки. Для достижения поставленной цели требуется решить следующие задачи:

1. Разработать способ представления и хранения текста, предложений и слов в памяти.
2. Реализовать алгоритм считывания текста произвольной длины с клавиатуры и вывода его на экран.
3. Освоить способ выделения из текста отдельных его частей: подстрок, являющихся словами, частями слов, предложениями.
4. Освоить способ модификации символов текста или его частей. Удаление, добавление новых символов.
5. Реализовать задания из курсовой работы на обработку текста в соответствии с условием варианта в виде функций.
6. Сгруппировать написанные функции по файлам в соответствии с выполняемыми ими задачами.
7. Написать Makefile для сборки проекта.

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Создание структур для хранения текста

Для удобного хранения текста были реализованы структуры Sentence и Text для предложений и текста соответственно. С помощью *typedef* для них были даны названия: *SentStruct* и *TextStruct* соответственно.

Структура Sentence включает в себя следующие поля:

- *wchar_t *content* – само предложение как массив широких символов
- *int length* – длина предложения в символах
- *int wordCount* – количество слов в предложении

Структура Text включает в себя следующие поля:

- *SentStruct *sentenceArray* – массив структур Sentence (предложений)
- *int sentenceCount* – количество предложений

Также для быстрой инициализации структур были созданы функции *SentStruct makeSentStruct()* и *TextStruct makeTextStruct()*, которые возвращают структуры с обнуленными целочисленными переменными и выделенной памятью для массивов.

Для объявления структур и функций был создан отдельный заголовочный файл *structs.h*, а для определения функций – *structs.c*.

2.2. Работа с памятью

Для работы с памятью было реализовано 3 функции:

- *void cleanMemory(TextStruct text)* – принимает структуру Text и освобождает память в каждом предложении и в самом массиве предложений при помощи *free()*.
- *int reallocateMemory(void **pointer, unsigned int newBuffer)* – принимает указатель на указатель области, в которой нужно изменить объём выделенной памяти, и размер нового буфера. Далее функция изменяет объём памяти с помощью функции из стандартной библиотеки *stdlib*

realloc() и возвращает результат изменения: 0 – изменение прошло успешно, 1 – *realloc* вернул нулевой указатель. То есть кратко данную функцию можно назвать «безопасным» *realloc*.

- *void deleteSentence(TextStruct *text, int index)* – принимает структуру Text и индекс удаляемого предложения. После функция освобождает память в предложении при помощи *free()* и сдвигает последующие предложения в массиве на один влево.

Для данных функций были созданы отдельные файлы *memory.c* и *memory.h*.

2.3. Ввод текста

Для возможности обработки кириллических символов установим локаль с помощью функции стандартной библиотеки *locale.h setlocale(LC_ALL, "")*.

Для функций чтения был создан отдельный файл read.c.

Стоит отметить, что в данной курсовой работе за конец предложения были приняты «точка» и символ переноса строки, который в то же время считается символом окончания всего текста. А за разделители слов были выбраны «пробел» и «запятая».

Чтение текста можно разделить на две части: чтение предложения и «склейка» прочтённых предложений. Соответственно было написано 2 функции:

- *int getSentence(SentStruct *newSent)* – принимает на вход указатель на структуру Sentence, поля которой впоследствии функция полностью заполнит следующим предложением. В процессе чтения в случае нехватки функция увеличивает объём памяти, выделенный для поля структуры *content*, выделяя его «кусками». В случае ошибки этой операции функция возвращает 2. В случае, если последний символ предложения – перенос строки, функция возвращает 1, как конец текста. В иных случаях возвращается 0. Функция считывает символы при помощи цикла *while* вплоть до конца предложения, притом

символ конца предложения не входит в само предложение. Для подсчёта слов в предложении была написана функция *getWordCount()*.

- *int getText(TextStruct *newText)* – принимает указатель на структуру Text, которую требуется заполнить введённым пользователем текстом. Для этого функция при помощи цикла *do-while* считывает предложения вплоть до момента, когда текущее предложение не вернёт 1 – знак окончания текста. В процессе считывания функция увеличивает объем памяти в случае нехватки, а также пропускает «пустые» предложения, те, в которых нет слов. Функция возвращает 0, если ошибок не было, 1, если текст пустой, 2, если произошла ошибка при добавлении памяти.

2.4. Первичная обработка текста

Первичная обработка текста включает в себя очистку каждого предложения от лишних знаков табуляции и пробелов в начале, а также удаление повторяющихся предложений без учёта регистра.

Все функции обработки текста, а также анализа для дальнейшей обработки, находятся в файле processing.c.

За удаление «пустых» символов в предложении отвечает функция *void deleteTabs(SentStruct *sent)*, которая принимает указатель на структуру с предложением и с помощью цикла while ищет первое вхождение не «пустого» символа с помощью функции *iswblank()* и сдвигает предложение до этого символа функцией *wmemmove()*. Данная функция является основной в функции *void deleteTabsInText(TextStruct *text)*, которая с помощью цикла *for* «очищает» все предложения.

За удаление повторяющихся предложений отвечает *void deleteRepeatedSentence(TextStruct *text)*, которая принимает указатель на структуру Text, после чего, пробегая по каждому предложению циклом *for* и

проверкой на повторение функцией *isRepeated()*, в случае повтора удаляет предложение функцией *deleteSentence()*, притом остаётся первое вхождение.

Функция *int isRepeated(SentStruct *sent, TextStruct *text)* принимает на вход проверяемое предложение в виде указателя на структуру Sentence и указатель на структуру Text. Далее функция циклом for сравнивает предложение с каждым в тексте при помощи *compareSentenceCaseIntensive()*, которая сравнивает их на основе длины, количества слов и собственно посимвольно без учёта регистра. Если предложение было повторено более 1 раза – оно не уникальное, поэтому функция в таком случае возвращает 1, иначе 0.

Данные функции объединяет *void cleanText(TextStruct *text)*, которая принимает указатель на структуру Text и очищает текст.

2.5. Запрос текста и команды

Для вывода меню используется функция *void printMenu(int menuType)*, которая принимает код меню (1 – стартовое меню; 2 – подсказка доступных действий; 3 – конечное сообщение), после чего печатает соответствующее меню.

Во избежание ввода пустого текста написана функция *int getTextAnswer(TextStruct *text)*, которая принимает указатель на заполняемую структуру Text, после чего запрашивает ввод текста до тех пор, пока он не заполнится либо не произойдет ошибки изменения памяти (возвращает 1).

Для запроса команды используется *int getAnswer(const wchar_t *question, int maxNumber)*, которая принимает вопрос (строку) и максимально доступное число для ввода (за минимальную шкалу установлен 0). Далее функция запрашивает число до тех пор, пока оно не будет введено корректно и попадёт в доступный диапазон. Для отслеживания корректности ввода используется *checkInput()*, которая проверяет количество верно считанных значений *wscanf()* и очищает входной поток, после чего возвращает либо число, либо -1 как ошибку ввода. Функция возвращает запрашиваемое число.

Данные функции вынесены в файл user_tips.c.

2.6. Обработка ошибок

Для функций обработки ошибок был создан файл error.c. Он содержит две функции:

- *void panic(int errorId)* – принимает код ошибки (1 – ошибка ввода; 2 – ошибка изменения памяти; 3 – ввод пустого текста; 4 – пустой текст). Ошибка ввода имеет место при запросе команды. Ошибка изменения памяти – при считывании предложения или текста в целом. Ошибка ввода пустого текста – при вводе текста без слов. Ошибка пустого текста – при удалении слов, оканчивающихся на заглавную букву, в результате которого в тексте не окажется слов. Функция печатает сообщение соответствующей ошибки.
- *void crash(TextStruct text)* – принимает структуру Text, после чего очищает память в ней с помощью *cleanMemory()* и выводит сообщение об ошибке *panic(2).*

2.7. Обработка текста согласно команде

Для функций обработки создан отдельный файл processing.c.

Первая подзадача:

Первая подзадача предполагает сдвиг в одном предложении, поэтому была учтена возможность использовать её как ко всему тексту, так и к одному предложению по номеру. Для этого написана функция *void moveWords (SentStruct *sent, int shift),* которая принимает сдвиг и указатель на структуру Sentence. Далее функция берёт остаток от деления сдвига на количество слов в предложении и начнёт алгоритм только в том случае, если он не 0. Остаток от деления позволяет упростить сдвиг в предложениях, где слов меньше значения сдвига. Алгоритм представляет собой сдвинутую запись слов и разделителей в предложение, то есть так как слова двигаются независимо от разделителей, начало записи разделителей – первый встретившийся разделитель, а слов – (сдвиг = н) н-ое слово с конца. Далее поочередно записываются разделители и слова, после чего их указатели двигаются на следующий разделитель/слово

соответственно (в случае с словами берется остаток от деления на длину предложения, чтобы предотвратить выход за границы массива). Это продолжается до тех пор, пока общий счётчик не достигнет значения длины предложения.

Функция *void moveWordsInText(TextStruct *text, int shift, int number)* принимает в качестве аргументов указатель на структуру Text, номер предложения (0 – весь текст) и сдвиг. После чего вызывает функцию *moveWordsInSentence()* либо единожды (номер не равен 0), либо для всего текста циклом *for*.

Вторая подзадача:

Для выполнения второй подзадачи требуется получить массив с уникальными кириллическими и латинскими символами. И так как заранее известно их общее количество, можно сразу определить массив данной длины. Для оптимизации проверки на уникальность можно использовать тот факт, что коды символов уникальны, поэтому мы можем однозначно определить их индекс в массиве.

Функция *int *findUniqueCharacters(TextStruct text)* принимает структуру Text и перебирает каждый символ в каждом предложении, проверяя его на букву (*iswalpha()*) и на принадлежность латинице (*isLatinic()*) или кириллице (*isCyrillic()*). Далее функция сопоставляет код символа с индексом и записывает в массив по этому индексу код символа (изначально все элементы массива равны 0). Функция возвращает массив кодов уникальных символов (латинские символы находятся вплоть до 51 индекса).

Третья подзадача:

Для хранения слов одинаковой длины удобно использовать структуру Text, так как длина слов, их количество будут легкодоступны.

Сначала нужно определить длину массива предложений, поскольку индексация будет по длине слов. Для этого была написана функция *int findMaxWordLen(TextStruct text)*, которая по структуре Text ищет максимальную длину слова в тексте.

Далее для каждого предложения создается копия при помощи функции *wchar_t *copyString(SentStruct sent)*, чтобы удобнее использовать функцию *wcstok()*. После чего функция разбивает копию на слова и в соответствии с длиной конкатенирует к соответствующей строке, хранящей слова одинаковой длины с пробелом.

Память для строк выделяется только если найдено слово соответствующей длины. Также память выделяется не по количеству символов, а количеству слов, поскольку длина слов заранее известна и легче отслеживать количество слов.

После заполнения структуры функция возвращает её.

Четвертая подзадача:

Для решения данной подзадачи была реализована функция *int removeWordsWithLastUpperCase(TextStruct *text)*, которая принимает указатель на структуру Text и удаляет в ней слова с заглавной буквой на конце. Стоит отметить, что в рамках данной подзадачи за слово считается само слово и разделители перед ним (если слово крайнее слева, то и разделители справа). То есть удаляется не только слово, но и «закрепленные» за ним разделители.

За собственно удаление слов отвечает *int removeWord(wchar_t *string, const wchar_t *word)*, которая принимает строку, в которой нужно удалить слово, и удаляемое слово. Затем функция ищет это слово в строке. Далее «левая точка» сдвигается до следующего слева слова (либо до начала строки, если слово первое). В случае если слово первое, то двигается и «правая точка». Затем по этим двум точкам подстрока удаляется.

Сама функция схожим с третьей подзадачей способом пробегает по каждому слову в тексте, только с проверкой на заглавную букву в конце с помощью iswupper().

После удаления уменьшается количество слов, длина соответствующей структуры Sentence. Если слов не осталось, предложение удаляется. Если были удалены все предложения, выводится сообщение и программа автоматически завершается.

В конце работы функция возвращает количество удаленных слов для передачи в функции вывода.

2.8. Вывод информации

Для всех функций вывода создан файл print.c.

Основная задача вывода – вывод текста. Для этого написана функция *void printText(TextStruct text)*, которая печатает входную структуру Text, а также номера предложений (для удобства пользования первой подзадачей). Текст выводится после его ввода для проверки корректности чтения и первичной обработки, а также после выполнения первой и четвертой подзадачи.

Для четвертой подзадачи также выводится количество удаленных слов функцией *void printResultsAfterRemoving(int count)*.

Для второй подзадачи используется *void printCharacters(const int *characters)*, которая принимает массив с кодами символов и поочередно выводит их, если значение элемента не 0. Также функция разграничивает вывод латинских и кириллических символов и, если их нет, выводит «отсутствуют».

Вывод для третьей подзадачи схож с выводом всего текста за исключением дополнительной информации о длине слов и их количестве. Функция выводит строку только в том случае, если в ней есть слова.

Также для «красоты» склонения слова «слово» была написана функция *wchar_t endOfWord(int number)*, которая позволяет по числительному перед «словом» определить его окончание (нулевое, “о”, “а”). Она используется при выводе для третьей и четвертой подзадачи.

2.9. Выход из программы

Для отслеживания конца программы цикл while ссылается на переменную *continueStatus*, которая становится 0 при выборе команды «0» (выход из программы).

После этого выводится конечное сообщение и очищается память, отведенная под структуру Text и все его предложения.

2.10. Написание Makefile

В итоге весь проект курсовой работы состоит из 15 файлов:

1. main.c – связывание функций ввода, обработки, вывода и т.д.
2. structs.c – функции инициализации структур
3. structs.h – объявление структур и функций их инициализации
4. memory.c – функции изменения памяти
5. memory.h – объявление функций изменения памяти
6. read.c – функции ввода (текста, чисел)
7. read.h – объявление функций ввода
8. user_tips.c – функции взаимодействия с пользователем
9. user_tips.h – объявление функций взаимодействия с пользователем
10. error.c – функции-обработчики ошибок
11. error.h – объявление функций-обработчиков ошибок
12. processing.c – функции анализа и обработки текста и его частей
13. processing.h – объявление функций анализа и обработки
14. print.c – функции вывода (текста, результата анализа)
15. print.h – объявление функций вывода

Для Makefile было создано две переменных:

- CC – компилятор. В данном случае gcc
- CFLAGS – флаги компилятора. Используются -c (компиляция без линковки) и -Wall (вывод ошибок при их возникновении)

В главную цель all помещено две зависимости:

- cw_exe – исполняемый файл, который получается линковкой всех полученных объектных файлов путём выставления цели для компиляции каждого .c файла и зависимостей для него в виде своего заголовочного файла и заголовочных файлов используемых функций, чтобы в случае их изменения, файл был скомпилирован снова.
- clean – очистка всех объектных файлов после сборки

ЗАКЛЮЧЕНИЕ

Были изучены основные конструкции языка C (циклы, условные операторы, массивы), а также основы работы с ним: выделение и изменение динамической памяти, использование структур, ввод строки неизвестной длины, её обработка при помощи функций стандартной библиотеки языка.

В итоге была написана программа, принимающая от пользователя текст любой длины, содержащий как латиницу, так и кириллицу. После чего она обрабатывает текст по команде пользователя в соответствии с задачами курсовой работы, список которых можно увидеть в контекстном меню программы. Запрос действий продолжается до тех пор, пока пользователь не введёт специальную команду для выхода из программы. Также в программе предусмотрено возникновение ошибок при изменении памяти, вводе некорректных данных, опустошении текста при удалении слов, оканчивающихся на заглавную букву.

Для сборки проекта курсовой работы был написан Makefile.

Примеры работы программы см. в приложении А.

Разработанный программный код см. в приложении Б.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган Б., Ритчи Д. Язык программирования Си. СПб.: "Невский Диалект", 2001. 352 с.

2. Методические указания по выполнению курсовой и лабораторных работ по дисциплине программирование. Первый курс. Осенний семестр / сост.: Кринкин К.В., Берленко Т.А., Заславский М.М., Чайка К.В., Допира В.Е., Гаврилов А.В. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2022. 39 с.

3. Cplusplus: [Электронный ресурс]. URL: https://cplusplus.com/

4. C-cpp: [Электронный ресурс]. URL: https://www.c-cpp.ru/

ПРИЛОЖЕНИЕ А
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Пример 1:

1. Считывание и первичная обработка текста.

```
Введите текст для его дальнейшей обработки
Данный текст не несёт никакого смысла, он лишь показывает возможности, которые может предоставить данная программА. This part
 of text in English, I know, that you know this language and that you can read it. НА самоМ делЕ, тестЫ писатЬ verY harD, bec
ausE бессмыслицА порой несёТ большЕ смыслА thaN смыслицА.               ,,,Данные знаки препинания указывают на ошибочность
предыдущего предложения, ведь ОНО несёт смысл, ПОТОМУ будет удалено.
                ТЕКУЩИЙ ТЕКСТ
1 : Данный текст не несёт никакого смысла, он лишь показывает возможности, которые может предоставить данная программА.
2 : This part of text in English, I know, that you know this language and that you can read it.
3 : НА самоМ делЕ, тестЫ писатЬ verY harD, becausE бессмыслицА порой несёТ большЕ смыслА thaN смыслицА.
4 : ,,,Данные знаки препинания указывают на ошибочность предыдущего предложения, ведь ОНО несёт смысл, ПОТОМУ будет удалено.
```

2. Вывод меню-подсказки

```
С введённым текстом программа может сделать следующее:
1 - Сделать сдвиг слов в предложении на натуральное число;
2 - Вывести все уникальные кириллические и латинские символы в тексте;
3 - Подсчитать и вывести количество слов и сами слова, длина которых равна 1, 2, и т.д.;
4 - Удалить все слова, которые заканчиваются на заглавный символ;
0 - Завершить выполнение программы.
Выберите действие:
```

3. Выполнение второй подзадачи

```
С введённым текстом программа может сделать следующее:
1 - Сделать сдвиг слов в предложении на натуральное число;
2 - Вывести все уникальные кириллические и латинские символы в тексте;
3 - Подсчитать и вывести количество слов и сами слова, длина которых равна 1, 2, и т.д.;
4 - Удалить все слова, которые заканчиваются на заглавный символ;
0 - Завершить выполнение программы.
Выберите действие:2

Уникальные латинские символы: a b c d e f g h i k l n o p r s t u v w x y D E I N T Y
Уникальные кириллические символы: а б в г д е ж з и й к л м н о п р с т у ц ч ш щ ы ь ю я А Д Е Й М Н О П Т У Ы Ь ё
```

4. Выполнение третьей подзадачи

```
Выберите действие:3

С длиной 1  в тексте 1  слово ( I )
С длиной 2  в тексте 7  слов  ( не он of in it НА на )
С длиной 3  в тексте 5  слов  ( you and you can ОНО )
С длиной 4  в тексте 15 слов  ( лишь This part text know that know this that read делЕ verY harD thaN ведь )
С длиной 5  в тексте 11 слов  ( текст несёт может самоМ тестЫ порой несёТ знаки несёт смысл будет )
С длиной 6  в тексте 8  слов  ( Данный смысла данная писатЬ большЕ смыслА Данные ПОТОМУ )
С длиной 7  в тексте 4  слова ( которые English becausE удалено )
С длиной 8  в тексте 3  слова ( никакого language смыслицА )
С длиной 9  в тексте 2  слова ( программА указывают )
С длиной 10 в тексте 2  слова ( показывает препинания )
С длиной 11 в тексте 5  слов  ( возможности бессмыслицА ошибочность предыдущего предложения )
С длиной 12 в тексте 1  слово ( предоставить )
```

5. Выполнение первой подзадачи с одним предложением

```
Выберите действие:1
Введите сдвиг слов в предложении:2
Выберите номер предложения (0 - весь текст):4
                ТЕКУЩИЙ ТЕКСТ
1 : Данный текст не несёт никакого смысла, он лишь показывает возможности, которые может предоставить данная программА.
2 : This part of text in English, I know, that you know this language and that you can read it.
3 : НА самоМ делЕ, тестЫ писатЬ verY harD, becausE бессмыслицА порой несёТ большЕ смыслА thaN смыслицА.
4 : ,,,будет удалено Данные знаки препинания указывают на ошибочность, предыдущего предложения ведь ОНО, несёт смысл ПОТОМУ.
```

6. Выполнение первой подзадачи для всего текста

```
Выберите действие:1
Введите сдвиг слов в предложении:3
Выберите номер предложения (0 - весь текст):0
                ТЕКУЩИЙ ТЕКСТ
1 : предоставить данная программА Данный текст не, несёт никакого смысла он, лишь показывает возможности которые может.
2 : can read it This part of, text in, English I know that you know this language and that you.
3 : смыслА thaN смыслицА, НА самоМ делЕ тесты, писатЬ verY harD becausE бессмыслицА пороЙ несёТ большЕ.
4 : ,,,несёт смысл ПОТОМУ будет удалено Данные знаки препинания, указывают на ошибочность предыдущего, предложения ведь ОНО.
```

7. Выполнение четвертой подзадачи

```
Выберите действие:4

В тексте было удалено 19 слов
                ТЕКУЩИЙ ТЕКСТ
1 : предоставить данная Данный текст не, несёт никакого смысла он, лишь показывает возможности которые может.
2 : can read it This part of, text in, English know that you know this language and that you.
3 : ,,,несёт смысл будет удалено Данные знаки препинания, указывают на ошибочность предыдущего, предложения ведь.
```

8. Выполнение второй подзадачи после удаления части слов

```
Выберите действие:2

Уникальные латинские символы: a c d e f g h i k l n o p r s t u w x y E T
Уникальные кириллические символы: а б в г д е ж з и й к л м н о п р с т у ч ш щ ы ь ю я Д ё
```

9. Выполнение третьей подзадачи после удаления части слов

```
Выберите действие:3

С длиной 2  в тексте 6  слов  ( не он it of in на )
С длиной 3  в тексте 4  слова ( can you and you )
С длиной 4  в тексте 11 слов  ( лишь read This part text know that know this that ведь )
С длиной 5  в тексте 7  слов  ( текст несёт может несёт смысл будет знаки )
С длиной 6  в тексте 4  слова ( данная Данный смысла Данные )
С длиной 7  в тексте 3  слова ( которые English удалено )
С длиной 8  в тексте 2  слова ( никакого language )
С длиной 9  в тексте 1  слово ( указывают )
С длиной 10 в тексте 2  слова ( показывает препинания )
С длиной 11 в тексте 4  слова ( возможности ошибочность предыдущего предложения )
С длиной 12 в тексте 1  слово ( предоставить )
```

10. Выход из программы

```
Выберите действие:0
Выполнение программы завершено. Спасибо за пользование!
```

Пример 2:

```
Введите текст для его дальнейшей обработки
ДанныЙ текстТ должеН бытЬ УдалеН. В неМ оченЬ странныЕ окончаниЯ.
                ТЕКУЩИЙ ТЕКСТ
1 : ДанныЙ текстТ должеН бытЬ УдалеН.
2 : В неМ оченЬ странныЕ окончаниЯ.
```

1. Выполнение второй подзадачи

```
Выберите действие:2

Уникальные латинские символы: отсутствуют
Уникальные кириллические символы: а б д е ж и к л н о р с т ч ы В Д Е Й М Н Т У Ь Я
```

2. Выполнение четвертой подзадачи для заведомо удаленного текста

```
Выберите действие:4

В тексте было удалено 10 слов
В результате удаления слов текст оказался пустым. Выполнение программы будет завершено автоматически.
Выполнение программы завершено. Спасибо за пользование!
```

Пример 3 (обработка ошибок):

1. Ввод пустого текста

```
Введите текст для его дальнейшей обработки

Не было введено ни одного непустого предложения. Введите текст ещё раз.
,,,  .  . , ,, .. ,, , ..
Не было введено ни одного непустого предложения. Введите текст ещё раз.
```

2. Ввод некорректных данных при выборе действия

```
Выберите действие:gfdfgd
Ошибка входных данных! Вводите целое число, лежащее в доступном диапазоне.
Выберите действие:5
Ошибка входных данных! Вводите целое число, лежащее в доступном диапазоне.
Выберите действие:-1
Ошибка входных данных! Вводите целое число, лежащее в доступном диапазоне.
Выберите действие:2gfdgf

Уникальные латинские символы: a b d e f h i l o r s t u w x T W
Уникальные кириллические символы: отсутствуют
```

Пример 4:

1. Удаление повторяющихся предложений

```
Введите текст для его дальнейшей обработки
This text. THis TEXT. is repeated a lot of times. THIS text. can annoy you. but. BUT. this text.
               ТЕКУЩИЙ ТЕКСТ
1 : This text.
2 : is repeated a lot of times.
3 : can annoy you.
4 : but.
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <locale.h>
#include <limits.h>
#include "structs.h"
#include "error.h"
#include "user_tips.h"
#include "print.h"
#include "processing.h"

int main()
{
	TextStruct text = makeTextStruct();
	int continueStatus = 1;
	setlocale(LC_ALL, "");

	printMenu(1);
	if (getTextAnswer(&text))
		/* если произошла ошибка при перевыделении памяти */
	{
		crash(text);
		return 1;
	}

	cleanText(&text);
	printText(text);
	while (continueStatus)
		/* пока пользователь не ввёл '0' (завершение программы) */
	{
		printMenu(2);
		switch (getAnswer(L"Выберите действие", OPTION_COUNT))
		{
			case 1:
			{
				moveWordsInText(&text, getAnswer(L"Введите сдвиг
слов в предложении", INT_MAX),
					getAnswer(L"Выберите номер предложения
(0 - весь текст)", text.sentenceCount));
				printText(text);
				break;
			}
			case 2:
			{
				int *uniq = findUniqueCharacters(text);
				printCharacters(uniq);
				free(uniq);
				break;
			}
			case 3:
			{
				TextStruct wordList = getWordsSameLength(text);
				if (wordList.sentenceCount != -1)
```

```
                              /* если при заполнении списка слов не
произошло ошибки перевыделения памяти */
                        {
                              printWordSameLength(wordList);
                              cleanMemory(wordList);
                        }
                        else
                        {
                              crash(wordList);
                        }
                        break;
                  }
                  case 4:
                  {
                        int count = removeWordsWithLastUpperCase(&text);
                        printResultsAfterRemoving(count);
                        if (text.sentenceCount && count)
                        {
                              printText(text);
                        }
                        else if (text.sentenceCount == 0)
                        {
                              panic(4);
                              continueStatus = 0;
                        }
                        break;
                  }
                  case 0:
                  {
                        continueStatus = 0;
                        break;
                  }
                  default:
                        break;
            }
      }
      if (text.sentenceCount)
            /* если под текст была выделена память */
      {
            cleanMemory(text);
      }
      printMenu(3);

      return 0;
}
```

Название файла: structs.h

```
#pragma once

#include <wchar.h>
#include <stdlib.h>

#define TEXT_BUF_SIZE 3
#define SENTENCE_BUF_SIZE 20

typedef struct Sentence {
      wchar_t *content;
```

```
	int length;
	int wordCount;
} SentStruct;

typedef struct Text {
	SentStruct *sentenceArray;
	int sentenceCount;
} TextStruct;

/* возвращает структуру Text с обнуленным количеством предложений и
выделенной динамической памятью */
TextStruct makeTextStruct();

/* возвращает структуру Sentence с обнуленными длиной и количеством слов
и выделенной динамической памятью */
SentStruct makeSentStruct();
```

Название файла: structs.c

```
#include "structs.h"

TextStruct makeTextStruct()
{
	TextStruct newText;

	newText.sentenceCount = 0;
	newText.sentenceArray = (SentStruct *) malloc(TEXT_BUF_SIZE *
sizeof(SentStruct));

	return newText;
}

SentStruct makeSentStruct()
{
	SentStruct newSent;

	newSent.length = 0;
	newSent.wordCount = 0;
	newSent.content = (wchar_t *) malloc(SENTENCE_BUF_SIZE *
sizeof(wchar_t));

	return newSent;
}
```

Название файла: memory.h

```
#pragma once

#include <string.h>
#include "structs.h"

/* принимает структуру Text и очищает выделенную под неё память */
void cleanMemory(TextStruct);

/* "безопасный realloc" принимает указатель на указатель элемента, в
котором нужно изменить буфер, а также размер самого буфера,
```

```
 * возвращает результат выполнения перевыделения памяти (0 - успешно; 1 -
ошибка) и перевыделяет память в случае успеха */
int reallocateMemory(void **, unsigned int);

/* удаляет i-ую структуру Sentence из массива предложений (структур
Sentence) структуры Text */
void deleteSentence(TextStruct *, int);
```

Название файла: memory.c

```
#include "memory.h"

void cleanMemory(TextStruct text)
{
	for (int i = 0; i < text.sentenceCount; i++)
	{
		if (text.sentenceArray[i].length)
			/* если предложение непустое (для него выделена память)
*/
		{
			free(text.sentenceArray[i].content);
		}
	}
	free(text.sentenceArray);
}

int reallocateMemory(void **pointer, unsigned int newBuffer)
{
	void *temp = realloc(*pointer, newBuffer);

	if (temp)
	{
		*pointer = temp;
	}
	else
	{
		return 1;
	}

	return 0;
}

void deleteSentence(TextStruct *text, int index)
{
	text->sentenceCount--;
	free(text->sentenceArray[index].content);
	memmove(text->sentenceArray + index, text->sentenceArray + index +
1,
			(text->sentenceCount - index) * (sizeof(SentStruct)));
	reallocateMemory((void **) &text->sentenceArray,
text->sentenceCount * sizeof(SentStruct));
}
```

Название файла: user_tips.h

```
#pragma once

#include <wchar.h>
```

```
#include "error.h"
#include "read.h"

#define START_TIP L"Введите текст для его дальнейшей обработки"
#define DESCRIPTION L"С введённым текстом программа может сделать
следующее:"
#define EXIT_OPTION L"0 - Завершить выполнение программы."
#define FIRST_OPTION L"1 - Сделать сдвиг слов в предложении на
натуральное число;"
#define SECOND_OPTION L"2 - Вывести все уникальные кириллические и
латинские символы в тексте;"
#define THIRD_OPTION L"3 - Подсчитать и вывести количество слов и сами
слова, длина которых равна 1, 2, и т.д.;"
#define FOURTH_OPTION L"4 - Удалить все слова, которые заканчиваются на
заглавный символ;"
#define OPTION_COUNT 4
#define EXIT_MESSAGE L"Выполнение программы завершено. Спасибо за
пользование!"

/* принимает код меню (1 - стартовое меню; 2 - подсказка доступных
действий; 3 - конечное сообщение)
 * печатает соответствующее меню-подсказку */
void printMenu(int);

/* принимает строку запроса и максимально возможное число (минимальное -
0), возвращает считанное число */
int getAnswer(const wchar_t *, int);

/* принимает указатель на структуру Text и пытается её заполнить, пока
пользователь не введёт непустой текст,
 * возвращает статус работы функции (0 - текст считан успешно; 1 -
произошла ошибка при перевыделении памяти) */
int getTextAnswer(TextStruct *);
```

Название файла: user_tips.c

```
#include "user_tips.h"

void printMenu(int menuType)
{
	switch (menuType)
	{
		case 1:
			wprintf(L"%ls\n", START_TIP);
			break;
		case 2:
			wprintf(L"\n%ls\n%ls\n%ls\n%ls\n%ls\n%ls\n",
DESCRIPTION, FIRST_OPTION, SECOND_OPTION, THIRD_OPTION,
					FOURTH_OPTION, EXIT_OPTION);
			break;
		case 3:
			wprintf(L"%ls", EXIT_MESSAGE);
			break;
		default:
			break; /* тип меню не был найден */
	}
}
```

```
int getAnswer(const wchar_t *question, int maxNumber)
{
	int answer;

	wprintf(L"%ls:", question);
	answer = checkInput(); /* безопасный ввод */
	while (answer > maxNumber || answer < 0)
		/* пока число не лежит в допустимом диапазоне */
	{
		panic(1);
		wprintf(L"%ls:", question);
		answer = checkInput();
	}

	return answer;
}

int getTextAnswer(TextStruct *text)
{
	int option = getText(text);

	while (option)
		/* пока текст незаполненный (пустой(1) или с ошибкой(2)) */
	{
		if (option == 2)
			/* ошибка перевыделения памяти */
		{
			return 1;
		}
		panic(3);
		option = getText(text);
	}

	return 0;
}
```

Название файла: error.h

```
#pragma once

#include <wchar.h>
#include "structs.h"
#include "memory.h"

#define INPUT_ERROR_MESSAGE L"Ошибка входных данных! Вводите целое число,
лежащее в доступном диапазоне."
#define MEMORY_ALLOCATION_ERROR_MESSAGE L"При перевыделении памяти
произошла ошибка. Текущий процесс будет завершён."
#define EMPTY_TEXT_ERROR_MESSAGE L"Не было введено ни одного непустого
предложения. Введите текст ещё раз."
#define EMPTY_TEXT_MESSAGE L"В результате удаления слов текст оказался
пустым. Выполнение программы будет завершено автоматически."

/* принимает код ошибки (1 - ошибка ввода; 2 - ошибка перевыделения
памяти; 3 - ввод пустого текста; 4 - пустой текст)
 * печатает соответствующее сообщение ошибки */
void panic(int);
```

```
/* принимает структуру Text, после чего очищает память в ней и выводит
сообщение об ошибке перевыделения памяти */
void crash(TextStruct);
```

Название файла: error.c

```
#include "error.h"

void panic(int errorId)
{
   switch (errorId)
   {
      case 1:
         wprintf(L"%ls\n", INPUT_ERROR_MESSAGE);
         break;
      case 2:
         wprintf(L"%ls\n", MEMORY_ALLOCATION_ERROR_MESSAGE);
         break;
      case 3:
         wprintf(L"%ls\n", EMPTY_TEXT_ERROR_MESSAGE);
         break;
      case 4:
         wprintf(L"%ls\n", EMPTY_TEXT_MESSAGE);
         break;
      default:
         break; /* код ошибки не был найден */
   }
}

void crash(TextStruct text)
{
   cleanMemory(text);
   panic(2);
}
```

Название файла: read.h

```
#pragma once

#include <wchar.h>
#include <stdio.h>
#include "structs.h"
#include "memory.h"
#include "processing.h"

/* принимает структуру Sentence, возвращает количество слов в предложении
*/
int getWordCount(SentStruct);

/* принимает указатель на структуру Sentence, после чего читает ввод и
заполняет её,
 * возвращает состояние предложения (0 - ошибок нет; 1 - конечное
предложение; 2 - произошла ошибка перевыделения памяти) */
int getSentence(SentStruct *);

/* принимает указатель на структуру Text, после чего заполняет её
предложениями до конца текста,
 * возвращает состояние текста (0 - ошибок нет; 1 - текст пустой; 2 -
```

```
произошла ошибка перевыделения памяти) */
int getText(TextStruct *);

/* очищает стандартный поток ввода */
void cleanStdIn();

/* проверяет ввод на возможность считывание числа,
 * в случае успеха возвращает число, иначе возвращает -1, очищает входной
поток */
int checkInput();
```

Название файла: read.c

```
#include "read.h"

int getWordCount(SentStruct sent)
{
	int wordCount = 0;

	if (!isSeparator(sent.content[0]) && sent.content[0] != L'\0')
		/* первый символ - начало слова и не конец строки */
	{
		wordCount++;
	}
	for (int i = 0; i < sent.length - 1; i++)
	{
		if (isSeparator(sent.content[i])
&& !isSeparator(sent.content[i + 1]))
			/* текущий символ разделительный, следующий - нет, то
есть дальше начинается слово */
		{
			wordCount++;
		}
	}

	return wordCount;
}

int getSentence(SentStruct *newSent)
{
	int sentenceCapacity = SENTENCE_BUF_SIZE;
	wchar_t character = getwchar();

	while (character != L'\n' && character != L'.')
	{
		if (newSent->length == sentenceCapacity - 1)
		{
			sentenceCapacity += SENTENCE_BUF_SIZE;
			if (reallocateMemory((void **) &newSent->content,
sentenceCapacity * sizeof(wchar_t)))
				/* произошла ошибка при перевыделении памяти */
			{
				return 2;
			}
		}
		newSent->content[newSent->length++] = character;
		character = getwchar();
	}
```

```
	newSent->content[newSent->length] = L'\0';
	newSent->wordCount = getWordCount(*newSent);
	reallocateMemory((void **) &newSent->content, (newSent->length + 1)
* sizeof(wchar_t));
	if (character == L'\n')
		/* предложение является последним */
	{
		return 1;
	}

	return 0;
}

int getText(TextStruct *newText)
{
	int textCapacity = TEXT_BUF_SIZE;
	SentStruct currentSent;
	int sentOption; /* отслеживание состояния текущего предложения */

	do {
		currentSent = makeSentStruct();
		sentOption = getSentence(&currentSent);
		if (currentSent.wordCount && sentOption != 2)
			/* если предложение непустое и не было ошибок выделения
памяти */
		{
			if (newText->sentenceCount == textCapacity)
			{
				textCapacity += TEXT_BUF_SIZE;
				if (reallocateMemory((void **)
&newText->sentenceArray, textCapacity * sizeof(SentStruct)))
					/* произошла ошибка при перевыделении памяти
*/
				{
					return 2;
				}
			}
			newText->sentenceArray[newText->sentenceCount++] =
currentSent;
		}
		else
		{
			free(currentSent.content);
			if (sentOption == 2)
				/* при перевыделении памяти под предложение
произошла ошибка */
			{
				return 2;
			}
		}
	} while (sentOption != 1);
		/* пока предложение не последнее */
	if (newText->sentenceCount == 0)
		/* текст пустой */
	{
		return 1;
	}
```

```
	reallocateMemory((void **) &newText->sentenceArray,
newText->sentenceCount * sizeof(SentStruct));

	return 0;
}

void cleanStdIn()
{
	while (getwchar() != L'\n');
}

int checkInput()
{
	int input;
	int successReadCount;

	successReadCount = wscanf(L"%d", &input);
	cleanStdIn(); /* очистка потока ввода */
	if (successReadCount)
		/* если значение было успешно считано */
	{
		return input;
	}

	return -1;
}
```

Название файла: processing.h

```
#pragma once

#include <wchar.h>
#include <wctype.h>
#include "structs.h"
#include "memory.h"

#define CHAR_COUNT (2 * (L'я' - L'a' + L'z' - L'a' + 3))
#define LATIN_ONE_CASE_CHAR_COUNT (L'z' - L'a' + 1)
#define CYRILLIC_ONE_CASE_CHAR_COUNT (L'я' - L'a' + 1) /* без учёта 'ё' и
'Ё' */
#define WORD_COUNT_BUFFER 3

/* принимает две структуры Sentence, возвращает результат их сравнения
без учёта регистра (1 - равны, 0 - нет) */
int compareSentenceCaseIntensive(SentStruct, SentStruct);

/* принимает указатели на структуры Sentence и Text, возвращает результат
поиска предложения в тексте (1 - найдено; 0 - нет) */
int isRepeated(SentStruct *, TextStruct *);

/* принимает указатель на структуру Text, после чего удаляет в ней
одинаковые предложения без учёта регистра */
void deleteRepeatedSentence(TextStruct *);

/* принимает указатель на структуру Sentence и удаляет лишние символы
табуляции в начале предложения */
void deleteTabs(SentStruct *);
```

```
/* принимает указатель на структуру Text и удаляет лишние символы
табуляции в начале каждого предложения */
void deleteTabsInText(TextStruct *);

/* принимает указатель на структуру Text и удаляет лишние табуляции и
повторяющиеся предложения */
void cleanText(TextStruct *);

/* принимает символ и проверяет, является ли он латинским (1 - является;
0 - нет) */
int isLatin(wchar_t);

/* принимает символ и проверяет, является ли он кириллическим (1 -
является; 0 - нет) */
int isCyrillic(wchar_t);

/* принимает структуру Text и возвращает массив с кодами найденных
символов (0 - символа с соответствующим индексом нет) */
int *findUniqueCharacters(TextStruct);

/* принимает строку и слово, после чего удаляет слово с левыми
разделителями из строки и возвращает длину удалённой подстроки */
int removeWord(wchar_t *, const wchar_t *);

/* принимает структуру Sentence и возвращает указатель на скопированную
строку этого предложения */
wchar_t *copyString(SentStruct);

/* принимает указатель на структуру Text и удяляет в ней все слова,
оканчивающиеся на заглавную букву
 * возвращает количество удалённых слов */
int removeWordsWithLastUpperCase(TextStruct *);

/* принимает символ и проверяет, является ли он разделителем (1 -
является; 0 - нет) */
int isSeparator(wchar_t);

/* принимает структуру Text, возвращает максимальную длину слова среди
всех предложений структуры */
int findMaxWordLen(TextStruct);

/* принимает структуру Text и возвращает другую структуру Text с
заполненной информацией о словах одной длины */
TextStruct getWordsSameLength(TextStruct);

/* принимает указатель на структуру Sentence и число, на которое нужно
сдвинуть слова в предложении */
void moveWords(SentStruct *, int);

/* принимает указатель на структуру Text, номер сдвигаемого предложения
(0 для всего текста) и сдвиг слов в нём
 * сдвигает слова в предложении на сдвиг вправо */
void moveWordsInText(TextStruct *, int, int);
```

Название файла: processing.c

```
#include "processing.h"
```

```
int compareSentenceCaseIntensive(SentStruct firstSentence, SentStruct
secondSentence)
{
	if (firstSentence.length == secondSentence.length &&
firstSentence.wordCount == secondSentence.wordCount)
		/* если предложения одной длины и содержат одинаковое
количество слов */
	{
		for (int i = 0; i < firstSentence.length; i++)
		{
			if (towlower(firstSentence.content[i]) !=
towlower(secondSentence.content[i]))
				/* если символы в одном регистре не равны */
			{
				return 0;
			}
		}
	}
	else
	{
		return 0;
	}

	return 1;
}

int isRepeated(SentStruct *sent, TextStruct *text)
{
	int repeatedCount = 0;

	for (int i = 0; i < text->sentenceCount; i++)
	{
		if (compareSentenceCaseIntensive(*sent,
text->sentenceArray[i]))
		{
			repeatedCount++;
			if (repeatedCount > 1)
			{
				return 1;
			}
		}
	}

	return 0;
}

void deleteRepeatedSentence(TextStruct *text)
{
	for (int i = text->sentenceCount - 1; i > 0; i--)
	{
		if (isRepeated(&text->sentenceArray[i], text))
		{
			deleteSentence(text, i);
		}
	}
}
```

```
void deleteTabs(SentStruct *sent)
{
	int firstWordChar = -1;

	while (iswblank(sent->content[++firstWordChar]));
		/* пока текущий символ не станет началом слова */
	if (firstWordChar)
		/* если первый символ - "пустой" */
	{
		wmemmove(sent->content, sent->content + firstWordChar,
sent->length - firstWordChar + 1);
		sent->length -= firstWordChar;
		reallocateMemory((void **) &sent->content, (sent->length + 1)
* sizeof(wchar_t));
	}
}

void deleteTabsInText(TextStruct *text)
{
	for (int i = 0; i < text->sentenceCount; i++)
	{
		deleteTabs(&text->sentenceArray[i]);
	}
}

void cleanText(TextStruct *text)
{
	deleteTabsInText(text);
	deleteRepeatedSentence(text);
}

int isLatin(wchar_t character)
{
	return (character >= L'a' && character <= L'z') || (character >=
L'A' && character <= L'Z');
}

int isCyrillic(wchar_t character)
{
	return (character >= L'a' && character <= L'я') || (character >=
L'A' && character <= L'Я')
			|| character == L'ё' || character == L'Ё';
}

int *findUniqueCharacters(TextStruct text)
{
	int *uniqChars = calloc(CHAR_COUNT, sizeof(int));

	for (int i = 0; i < text.sentenceCount; i++)
	{
		for (int j = 0; j < text.sentenceArray[i].length; j++)
		{
			if (iswalpha(text.sentenceArray[i].content[j]))
			{
				if (isLatin(text.sentenceArray[i].content[j]))
				{
```

```
                              if
(iswlower(text.sentenceArray[i].content[j]))
                              {

     uniqChars[text.sentenceArray[i].content[j] - L'a'] =
text.sentenceArray[i].content[j];
                              }
                              else
                              {

     uniqChars[text.sentenceArray[i].content[j] - L'A' +
LATIN_ONE_CASE_CHAR_COUNT] =

     text.sentenceArray[i].content[j];
                              }
                         }
                         else if
(isCyrillic(text.sentenceArray[i].content[j]))
                         {
                              if (text.sentenceArray[i].content[j] == L'ë')
                              {
                                   uniqChars[CHAR_COUNT - 2] =
text.sentenceArray[i].content[j];
                              }
                              else if (text.sentenceArray[i].content[j] ==
L'Ë')
                              {
                                   uniqChars[CHAR_COUNT - 1] =
text.sentenceArray[i].content[j];
                              }
                              else if
(iswlower(text.sentenceArray[i].content[j]))
                              {

     uniqChars[text.sentenceArray[i].content[j] - L'a' + 2 *
LATIN_ONE_CASE_CHAR_COUNT] =

     text.sentenceArray[i].content[j];
                              }
                              else
                              {

     uniqChars[text.sentenceArray[i].content[j] - L'A' + 2 *
LATIN_ONE_CASE_CHAR_COUNT +

CYRILLIC_ONE_CASE_CHAR_COUNT] = text.sentenceArray[i].content[j];
                              }
                         }
                    }
               }
     }

     return uniqChars;
}

int removeWord(wchar_t *string, const wchar_t *word)
{
```

```
	wchar_t *startPoint;
	wchar_t *endPoint;

	startPoint = wcsstr(string, word);
	endPoint = startPoint + wcslen(word);
	while (startPoint > string && isSeparator(*(startPoint - 1)))
		/* пока стартовая точка в пределах строки и символ за ней -
разделитель */
	{
		startPoint--;
	}
	if (startPoint == string)
		/* если удаляется слово в начале строки */
	{
		while (endPoint < string + wcslen(string) &&
isSeparator(*endPoint))
			/* пока конечная точка в пределах строки и она указывает
на разделитель */
		{
			endPoint++;
		}
	}
	wmemmove(startPoint, endPoint, wcslen(string) - (startPoint -
string));

	return (int) (endPoint - startPoint);
}

wchar_t *copyString(SentStruct sent)
{
	wchar_t *stringCopy = (wchar_t *) malloc((sent.length + 1) *
sizeof(wchar_t));
	wcsncpy(stringCopy, sent.content, sent.length + 1);

	return stringCopy;
}

int removeWordsWithLastUpperCase(TextStruct *text)
{
	int deletedCount = 0;

	for (int i = 0; i < text->sentenceCount; i++)
	{
		wchar_t *pointer;
		wchar_t *stringCopy = copyString(text->sentenceArray[i]);
		wchar_t *word = wcstok(stringCopy, L" ,", &pointer);
		while (word != NULL)
		{
			if (iswupper(word[wcslen(word) - 1]))
				/* если последний символ заглавный */
			{
				text->sentenceArray[i].length -=
removeWord(text->sentenceArray[i].content, word);
				text->sentenceArray[i].wordCount--;
				deletedCount++;
			}
			word = wcstok(NULL, L" ,", &pointer);
```

```
            }
            if (text->sentenceArray[i].wordCount == 0)
                  /* после удалений слов предложение стало пустым */
            {
                  deleteSentence(text, i);
                  i--;
            }
            else
            {
                  reallocateMemory((void **)
&text->sentenceArray[i].content,
                                          (text->sentenceArray[i].length +
1) * sizeof(wchar_t));
            }
            free(stringCopy);
      }

      return deletedCount;
}

int isSeparator(wchar_t character)
{
      return character == L',' || character == L' ';
}

int findMaxWordLen(TextStruct text)
{
      int max = 0;
      int len;

      for (int i = 0; i < text.sentenceCount; i++)
      {
            len = 0;
            for (int j = 0; j < text.sentenceArray[i].length; j++)
            {
                  if (isSeparator(text.sentenceArray[i].content[j]))
                  {
                        if (len > max)
                        {
                              max = len;
                        }
                        len = 0;
                  }
                  else
                  {
                        len++;
                  }
            }
            if (len > max)
            {
                  max = len;
            }
      }

      return max;
}
```

```
TextStruct getWordsSameLength(TextStruct text)
{
	int wordLen;
	TextStruct wordList;
	wordList.sentenceCount = findMaxWordLen(text);
	wordList.sentenceArray = (SentStruct *)
malloc(wordList.sentenceCount * sizeof(SentStruct));
	for (int i = 0; i < wordList.sentenceCount; i++)
	{
		wordList.sentenceArray[i].length = 0;
		wordList.sentenceArray[i].wordCount = 0;
	}

	for (int i = 0; i < text.sentenceCount; i++)
	{
		wchar_t *pointer;
		wchar_t *stringCopy = copyString(text.sentenceArray[i]);
		wchar_t *word = wcstok(stringCopy, L" ,", &pointer);
		while (word != NULL)
		{
			wordLen = (int) wcslen(word);
			if (wordList.sentenceArray[wordLen - 1].wordCount == 0)
				/* '-1' требуется для полного занятия пространства
массива */
			{
				wordList.sentenceArray[wordLen - 1].content =
(wchar_t *) calloc(WORD_COUNT_BUFFER * (wordLen + 1) + 2,

                                        sizeof(wchar_t));
                    /* выделение памяти на несколько слов с
пробелами и '\0' */
                    wordList.sentenceArray[wordLen - 1].length = 1;
                    wordList.sentenceArray[wordLen - 1].content[0] = L'
';
			}
			wordList.sentenceArray[wordLen - 1].wordCount++;
			wordList.sentenceArray[wordLen - 1].length += wordLen +
1;
			if ((wordList.sentenceArray[wordLen - 1].wordCount -
1) % WORD_COUNT_BUFFER == 0 &&
                    wordList.sentenceArray[wordLen - 1].wordCount != 1)
                    /* количество слов до вставки кратно буферу, значит
он переполнен */
			{
                    if (reallocateMemory((void **)
&wordList.sentenceArray[wordLen - 1].content,
                                                    (WORD_COUNT_BUFFER *

(wordList.sentenceArray[wordLen - 1].wordCount / WORD_COUNT_BUFFER + 1) *
                                                     (wordLen + 1) + 2) *
sizeof(wchar_t)))
                        /* выделение памяти на ещё несколько слов с
пробелами */
                    {
                        wordList.sentenceCount = -1;
                        return wordList;
                    }
```

```
                }
                wcsncat(wordList.sentenceArray[wordLen - 1].content,
word, wordLen);
                wcsncat(wordList.sentenceArray[wordLen - 1].content, L"
", 1);
                word = wcstok(NULL, L" ,", &pointer);
           }
           free(stringCopy);
     }
     for (int i = 0; i < wordList.sentenceCount; i++)
     {
           if (wordList.sentenceArray[i].wordCount)
           {
                reallocateMemory((void **)
&wordList.sentenceArray[i].content,
                                      (wordList.sentenceArray[i].length
+ 1) * sizeof(wchar_t));
           }
     }

     return wordList;
}

void moveWords(SentStruct *sent, int shift)
{
     shift %= sent->wordCount;

     if (shift)
     {
           wchar_t *stringCopy = copyString(*sent);
           int generalCounter = 0; /* положение текущего символа в
конечной строке */
           int separatorCounter = 0; /* положение текущего первого
невставленного символа - разделителя */
           int wordCounter = sent->length; /* положение текущего первого
невставленного символа - не разделителя */
           while (shift)
           {
                while (isSeparator(sent->content[--wordCounter])
|| !isSeparator(sent->content[wordCounter - 1]));
                     /* пока предыдущий символ - разделитель или
предыдущий за ним - не разделитель*/
                shift--;
           }
           while (isSeparator(stringCopy[generalCounter]))
                /* пока текущий символ - разделитель */
           {
                sent->content[generalCounter++] =
stringCopy[separatorCounter++];
           }
           do {
                while (!isSeparator(stringCopy[separatorCounter]) &&
separatorCounter < sent->length)
                     /* пока текущий символ - не разделитель и он не
выходит за пределы строки */
                {
                     separatorCounter++;
```

```
                }
                while (isSeparator(stringCopy[wordCounter %
sent->length]))
                      /* пока текущий символ - разделитель */
                {
                      wordCounter++;
                }
                while (!isSeparator(stringCopy[wordCounter %
sent->length]))
                      /* пока текущий символ - не разделитель */
                {
                      sent->content[generalCounter++] =
stringCopy[wordCounter++ % sent->length];
                      if (wordCounter == sent->length)
                            /* началось новое слово с начала строки */
                      {
                            break;
                      }
                }
                while (isSeparator(stringCopy[separatorCounter]) &&
separatorCounter < sent->length)
                      /* пока текущий символ - разделитель и он не
выходит за пределы строки */
                {
                      sent->content[generalCounter++] =
stringCopy[separatorCounter++];
                }
          } while (generalCounter < sent->length);
                /* пока предолжение не заполнено */
     }
}

void moveWordsInText(TextStruct *text, int shift, int number)
{
     if (number)
     {
          moveWords(&text->sentenceArray[number - 1], shift);
     }
     else
     {
          for (int i = 0; i < text->sentenceCount; i++)
          {
                moveWords(&text->sentenceArray[i], shift);
          }
     }
}
```

Название файла: print.h

```
#pragma once

#include "structs.h"

#define CHAR_COUNT (2 * (L'я' - L'a' + L'z' - L'a' + 3))
#define LATIN_ONE_CASE_CHAR_COUNT (L'z' - L'a' + 1)
#define CYRILLIC_ONE_CASE_CHAR_COUNT (L'я' - L'a' + 1) /* без учёта 'ё' и
'Ё' */
```

```
/* принимает структуру Text и построчно выводит все предложения,
хранящиеся в ней */
void printText(TextStruct);

/* принимает массив с кодами уникальных символов, после чего выводит
символы, чьё значение не равно 0 */
void printCharacters(const int *);

/* принимает количество "слов", возвращает последнюю букву в зависимости
от падежа */
wchar_t endOfWord(int);

/* принимает количество удаленных слов, после чего печатает их количество
*/
void printResultsAfterRemoving(int);

/* принимает структуру Text с данными о словах одной длины, после
печатает их с указанием количества и длины */
void printWordSameLength(TextStruct);
```

Название файла: print.c

```
#include "print.h"

void printText(TextStruct text)
{
   wprintf(L"\t\tТЕКУЩИЙ ТЕКСТ\n");
   for (int i = 0; i < text.sentenceCount; i++)
   {
      wprintf(L"%-2d: %ls.\n", i + 1, text.sentenceArray[i].content);
   }
}

void printCharacters(const int *characters)
{
   int count = 0;

   wprintf(L"\nУникальные латинские символы:");
   for (int i = 0; i < LATIN_ONE_CASE_CHAR_COUNT * 2; i++)
      /* перебор всех элементов массива, отведенных под латинские символы
*/
   {
      if (characters[i])
         /* если значение элемента массива не равно 0 */
      {
         count++;
         wprintf(L" %lc", characters[i]);
      }
   }
   if (!count)
      /* не было встречено ни одного латинского символа */
   {
      wprintf(L" отсутствуют");
   }
   count = 0;
   wprintf(L"\nУникальные кириллические символы:");
   for (int i = LATIN_ONE_CASE_CHAR_COUNT * 2; i < CHAR_COUNT; i++)
      /* перебор всех элементов массива, отведенных под кириллические
```

```
символы */
   {
      if (characters[i])
         /* если значение элемента массива не равно 0 */
      {
         count++;
         wprintf(L" %lc", characters[i]);
      }
   }
   if (!count)
      /* не было встречено ни одного кириллического символа */
   {
      wprintf(L" отсутствуют");
   }
   wprintf(L"\n");
}

wchar_t endOfWord(int number)
{
   if (number % 100 < 5 || number % 100 > 20)
   {
      if (number % 10 == 1)
      {
         return L'о';
      }
      else if (number % 10 >= 2 && number % 10 <= 4)
      {
         return L'а';
      }
   }

   return L' ';
}

void printResultsAfterRemoving(int count)
{
   wprintf(L"\n");
   if (count)
   {
      wprintf(L"В тексте было удалено %d слов%lc\n", count,
endOfWord(count));
   }
   else
   {
      wprintf(L"В тексте не оказалось слов, оканчивающихся на заглавную
букву.\n");
   }
}

void printWordSameLength(TextStruct wordList)
{
   wprintf(L"\n");
   for (int i = 0; i < wordList.sentenceCount; i++)
   {
      if (wordList.sentenceArray[i].wordCount)
         /* если в строке есть слова */
      {
```

```
         if (endOfWord(wordList.sentenceArray[i].wordCount) == L' ')
         {
            wprintf(L"С длиной %-2d в тексте %-2d слов  (%ls)\n", i + 1,
                  wordList.sentenceArray[i].wordCount,
wordList.sentenceArray[i].content);
         }
         else
         {
            wprintf(L"С длиной %-2d в тексте %-2d слов%lc (%ls)\n", i +
1, wordList.sentenceArray[i].wordCount,
                  endOfWord(wordList.sentenceArray[i].wordCount),
wordList.sentenceArray[i].content);
         }
      }
   }
}
```

Название файла: Makefile

```
CC=gcc
CFLAGS=-c -Wall

all: cw_exe clean

cw_exe: main.o error.o memory.o print.o processing.o read.o structs.o
user_tips.o
     $(CC) main.o error.o memory.o print.o processing.o read.o structs.o
user_tips.o -o cw_exe

main.o: main.c print.h user_tips.h structs.h error.h processing.h
     $(CC) $(CFLAGS) main.c

error.o: error.c error.h memory.h structs.h
     $(CC) $(CFLAGS) error.c

memory.o: memory.c memory.h structs.h
     $(CC) $(CFLAGS) memory.c

print.o: print.c print.h structs.h
     $(CC) $(CFLAGS) print.c

read.o: read.c read.h structs.h memory.h processing.h
     $(CC) $(CFLAGS) read.c

structs.o: structs.c structs.h
     $(CC) $(CFLAGS) structs.c

user_tips.o: user_tips.c user_tips.h read.h error.h
     $(CC) $(CFLAGS) user_tips.c

processing.o: processing.c processing.h structs.h memory.h
     $(CC) $(CFLAGS) processing.c

clean:
     rm *.o
```