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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка текста на языке С

Студент гр. 3384 _____ Рудаков А.Л.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2023

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент: Рудаков А.Л.

Группа: 3384

Тема работы: обработка текста на языке С
Исходные данные:
Вариант 4.18
Вывод программы должен быть произведен в стандартный поток вывода: *stdout*.
Ввод данных в программе в стандартный поток ввода: stdin.

Ввод данных:

После вывода информацию о варианте курсовой работе программа ожидает ввода пользователем числа – номера команды:

0 – вывод текста после первичной обязательной обработки (если предусмотрена заданием данного уровня сложности)

1 – вызов функции под номером 1 из списка задания

2 – вызов функции под номером 2 из списка задания

3 – вызов функции под номером 3 из списка задания

4 – вызов функции под номером 4 из списка задания

5 – вывод справки о функциях, которые реализует программа.

Программа не должна выводить никаких строк, пока пользователь не введет число.

В случае вызова справки (опция 5) текст на вход подаваться не должен, во всех остальных случаях после выбора опции должен быть считан текст.

Признаком конца текста считается два подряд идущих символа переноса строки ‘\n’. После каждой из функций нужно вывести результат работы программы и завершить программу.

В случае ошибки и невозможности выполнить функцию по какой-либо причине, нужно вывести строку:

Error: <причина ошибки>

Каждое предложение должно выводиться в отдельной строке, пустых строк быть не должно. Текст представляет собой предложения, разделенные точкой.

Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских букв и цифр. Длина текста и каждого предложения заранее не известна.

Программа должна сохранить этот текст в динамический массив строк и оперировать далее только с ним.

Программа должна найти и удалить все повторно встречающиеся предложения (сравнивать их следует посимвольно, но без учета регистра).

Программа должна выполнить одно из введенных пользователем действий и завершить работу:

1. Удалить все символы в начале и конце предложения так, чтобы в итоге первый и последний символ предложения были различными (без учета регистра). Например, предложение “abcdba” должно принять вид “cd”.
2. Отсортировать все слова в предложении в лексикографическом порядке.
3. Удалить все предложения, в которых хотя бы одного слово является палиндромом.
4. Вывести все предложения в которых есть слово “HiddenTiger” и которое не является первым словом.

Все сортировки должны осуществляться с использованием функции стандартной библиотеки. Использование собственных функций, при наличии аналога среди функций стандартной библиотеки, запрещается.

Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции.

Водержание пояснительной записки: «Аннотация», «Содержание», «Введение», «Общие сведения», «Функция main()», «Функция ввода», «Функции основной обработки», «Дополнительные функции», «Функция вывода», «Makefile», «Заключение», «Список используемых источников», «Приложение А. Примеры работы программы», «Приложение Б. Исходный код программы».

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 16.10.2023

Дата сдачи реферата: 10.12.2023

Дата защиты реферата: 15.12.2023

Студент гр. 3384 _____ Рудаков А.Л.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Курсовая работа заключается в создании программы для обработки текста на языке С. Для хранения текста была использована структура, представляющая из себя динамический массив строк, расширяющийся или уменьшающийся по мере необходимости.

Программа считывает целоedocx число — команду, далее считывает текст, сразу разделяя его на предложения. После чего удаляет одинаковые предложения и убирает лишние пробелы в конце строк. После чего выполняет команду с ранее введенным номером и выводит результат. После выполнения программы динамически выделенная под текст память освобождается.

Пример работы программы приведен в приложении А.

Исходный код программы приведен в приложении Б.

SUMMARY

The course work is to create a program for text processing in the C language. To store the text, a structure was used, which is a dynamic array of strings, expanding or decreasing as needed.

The program reads an integer command, then reads the text, immediately dividing it into sentences. After that, it deletes the same sentences and removes unnecessary spaces at the end of the lines. After that, it executes the command with the previously entered number and outputs the result. After the program is executed, the dynamically allocated subtext memory is freed.

An example of how the program works is given in Appendix A.

The source code of the program is given in Appendix B.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: написать программу на языке С, которая считывает текст и обрабатывает его определенным образом. Для этого необходимо реализовать:

- Считывание текста;
- Хранение текста в динамическом массиве строк;
- Удаление предложений из текста;
- Замена символов в предложении;
- Построение нового предложения;
- Поиск слова в предложении;
- Вывод текста;

1. ОБЩИЕ СВЕДЕНИЯ

Программа защищена от ошибок ввода. При ошибке во время исполнения программы она вызывает функцию *fatal_error*, выводит ошибку и завершает работу с помощью функции *exit()*. Все функции курсовой работы реализованы с использованием стандартных библиотек языка С. Функции были сгруппированы по их функционалу и разбиты на отдельные файлы, которые имеют заголовочные файлы дли их подключения. Также для предотвращения повторного подключения файлов были использованы директивы условной компиляции. Сборка файлов происходит через Makefile.

2. ФУНКЦИЯ MAIN().

Основным файлом в работе является файл *main.c*. Внутри него реализована функция *int main()*.

В начале программы была подключен заголовочный файл *stdio.h*, в котором объявлены функции стандартного ввода и вывода. Также были подключены заголовочные файлы *stdlib.h*, в котором объявлены функции для работы с памятью, *string.h*, в котором объявлены функции для работы со строками, и *ctype.h*, в котором объявлены функции для преобразования символов. Кроме того были подключены заголовочные файлы *scan_text.h,* содержащий прототип функции ввода текста, *processing_functions.h,* содержащий прототипы функций основных обработок текста, *additional_processing.h*, содержащий прототипы функций дополнительной обработки, *fatal_error.h*, содержащий прототип функции для аварийного выхода, и *output_text.h*, содержащий прототип функции для вывода текста.

Далее была написана функция *int main()*. Первым действием внутри неё выполняется вывод информации о варианте курсовой и об авторе работы. Инициализированны *flag* и *flag2*, в которые будет произведен ввод первых двух символов, если символы не являются целыми числами от 0 до 5, то будет вызвана функция *fatal_error* и работа программы будет прекращена. Вся проверка происходит в ветвях условного оператора. Переменная *point* отвечает за введённое число — номер команды, *count_sentecne* — за количество введенных предложений. Далее инициализирован динамический массив строк text, в который соответственно будет записан введенный текст. Далее производится ввод через функцию *scan_text(&text,&count_sentence)* и первичная обработка текста благодаря функции *mandatory_processing(&text,&count_sentence)*. После этого благодаря условному оператору *switch* выполняется обработка текста по нужной команде. После этого текст выводится через функцию *output_text(&text,&count_sentence)* и программа завершает работу.

3. ФУНКЦИЯ ВВОДА

Ввод текста реализован в функции *void scan_text(char ***text, int *count_sentence)*, находящейся в файле *scan_text.c*. Ей на вход подается указатель на динамический массив строк и указатель на переменную, хранящую в себе количество предложений в тексте. Каждое предложение сохраняется с точкой на конце и последним символом идет *\0*, также сразу убираются лишние символы в начале строки.

Ввод происходит внутри цикла, который выполняется до момента, пока не встретится два подряд идущих символа перевода строки. Каждое предложение считывается посимвольно через функцию *getchar()*. Далее в условных операторах идет проверка на то, какой символ был считан. Если был считан символ перевода строки, то переменная *flag* увеличивается на 1 и цикл переходит к следующей итерации благодаря оператору *continue*. Если же символ является символом пробела, табуляции или переноса строки и в текущем предложении пока нет символов, то выполняется переход к следующей итерации цикла, это сделано для того, чтобы убрать лишние символы в начале строки. Если оба условия не подходят, то память текущего предложения перевыделяется благодаря функции *realloc*, становясь на один символ больше. В выделенное место записывается считанный символ. Переменная, отвечающая за количество символов в текущем предложении увеличивается, *flag* обнуляется. После происходит проверка на то, был ли считанный символ точкой, если да, то память текущего предложения перевыделяется благодаря функции *realloc*, становясь на один символ больше. В выделенное место записывается символ окончания строки *\0*. память динамического массива строк перевыделяется. становясь на число байт одного указателя больше, в новом предложении выделяется память под один символ. Счетчик символов текущего предложения обнуляется, счетчик количества предложений увеличивается.

После выполнения цикла происходит проверка того, корректно ли записано последнее предложение, то есть имеет точку и знак окончания строки

на конце. Проверка происходит благодаря счетчику *count_flag*, который в случае, если предложение корректно, должен быть равен счетчику количества предложений. Если это не так, то память текущего предложения перевыделяется благодаря функции *realloc*, становясь на два символа больше. В выделенном месте в предпоследний символ строки записывается точка, в последний символ окончания строки. Счетчик количества предложений увеличивается на 1.

Прототип данной функции находится в файле *scan_text.h*.

4. ФУНКЦИИ ОСНОВНОЙ ОБРАБОТКИ

4.1. Сведения о файлах

Прототипы функций обработки текст хранятся в файле *processing_functions.h*. Внутри находятся такие функции, как: *mandatory_processing()* - функция обязательной обработки текста, *delete_while_the_same()* - функция удаления одинаковыъ элементов начала и конца предложения, *sort_lexicographic_order()* - функция сортировки слов в предложениях в лексикографическом порядке, *remove_palindromes()* - функция удаления предложений, в которых встречаются палиндромы, *find_word()* - функция, в которой выполянется поиск предложений со словом *HiddenTiger*, и функция *function_info()*, которая выводит информацию о программе.

Описания функции находятся в файле *processing_functions.c*.

4.2. Первичная обработка

За первичную обязательную обработку отвечает функция *void mandatory_processing(char ***text, int *count_sentence)*, которая получает на вход указатель на динамический массив строк и указатель на переменную, хранящую количество предложений. Внутри функции происходит поиск повторяющихся предложений с дальнейшим их удалением. Оставшиеся предложения проверяются на лишние разделители слов в конце предложения, если такие имеются, то они удаляются из предложения.

Вначале в функции инициализируется динамический массив чисел *delete,* который будет хранить индексы предложений в тексте, которые нужно удалить. Поиск предложений происходит в цикле который проходит по индексам всех предложений, после этого внутри первого цикла идет второй цикл, который происходит от индекса следующего, после текущего предложения, до индекса последнего предложения. Внутри него изначально сравнивается длина двух предложений, если она не равна, вложенный цикл переходит к следующей

итерации. Если же они равны, то инициализируется новый счетчик count и вызывается еще один цикл, в котором проевряется схожесть предложений посимвольно, вне зависимости от регистра, это сделано благодаря функции *toupper()*, переводящий буквы в верхний регистр. Если какие-нибудь 2 буквы с одинаковыми индексами не равны, то count увеличивается на 1. Если после этого счетчик равен нулю, значит предложения одинаковые, в таком случае выполянется проверка на то, есть ли индекс предложения, которое встретилось во втором цикле, в массиве *delete*. Если его в нем нет, то память массива перевыделяется при помощи функции *realloc()*, становясь при этом на количество байт одного int больше. в текущий элемент записывается данный индекс.

Далее выполняется удаление предложений из текста при помощи функции r*emoves()*.

После этого идет цикл, проходящий по всем предложениям и проверяющий на то, есть ли в конце предложений лишние разделители, если есть, то память текущего предложения перевыделяется, и в предпоследний и последний символы записываются точка и *\0* соответственно.

4.3. Функция 1

За обработку при поступлении команды 1 отвечает функция *void delete_while_the_same(char ***text, int *count_sentence)*, которой на вход подаются указатель на динамический массив строк и указатель на переменную, хранящую в себе количесвто предложений. Внутри функции происходит проверка на идентичность символов начала и конца предложений и удаление одинаковых.

Внутри функции идет цикл, проходящий по индексам всех предложений. В нем иицализируется динамическая строка *new_string* размером текущего предложения. В данную строку копируется содержимое текущего предложения (кроме точки) при помощи функции *strncpy()*. Последнему символу

присваеивается *\0*. Далее инициализируются указатели на символы начала и конца данной строки — *ptr_begin* и *ptr_end* соответсвенно. После идет цикл *while*, происходящий до момента, пока указатель на начало меньше указателя на конец. Внутри идет проверка на идентичность символов, если они идентичны то указатели сдвигаются на 1 ближе друг к другу с каждой стороны, если нет, то происходит выход из цикла при помощи оператора *break*. При помощи функции *free()* очищается память текущей строки, после чего инициализируется заного при помощи функции *malloc()*. Длина нового куска памяти равна разности указателей *+3*. в Данное пространство копируется содержимое строки *new_string*, начиная с символа, на который указывает *ptr_begin*. В предпоследний и последний символы строки записываются точка и *\0* соответсвенно, после чего освобождается память, выделенная строку *new_string*.

4.4. Функция 2

За обработку при поступлении команды 2 отвечает функция *void sort_lexicographic_order(char ***text, int *count_sentence)*, которой на вход подаются указатель на динамический массив строк и указатель на переменную, хранящую в себе количесвто предложений. В функции происходит сортировка слов в предложении в лексикографическом порядке.

Внутри функции используется цикл, проходящий по индексам всех предложений. Внутри него инициализируется динамический массив строк *sentence*, строками которого будут слова из предложения. После этого текущая строка разбивается на слов по разделителям и слова записываются в массив строк sentence, все это происходит благодаря функции *split()*. После этого, если количесвто слов не равно 0, то память начальной строки освобождается, и заново инициализируется с длиной первого слова *+1*. Динамический массив строк в это время сортируется при помощи функции стандатной библиотеки *qsort(),* функцией, при помощи которой выполняется сравнение — *cmp()*. псоле этого первое слово копируется в текущую строку при помощи функции *strncpy()*. Далее

идет цикл, добавляющий пробел и новое следующее словов в предложение, при этом увеличивая объем выделенной под нее памяти. После цикла в конец предложения добавляется точка и символ окончания строки. Далее память динамического массива строк sentence очищается вначале построчно, а потом и по указателям на строки.

4.5. Функция 3

За обработку при поступлении команды 3 отвечает функция *void remove_palindromes(char ***text, int *count_sentence)*, которой на вход подаются указатель на динамический массив строк и указатель на переменную, хранящую в себе количесвто предложений. В функции происходит поиск предложений, вкоторых встречаются палиндромы и удаление таких предложений.

Внутри функции инифциализируется массив целых чисел *delete*, в котором будут храниться индексы предложений, которых нужно будет удалить. Далее используется цикл, проходящий по индексам всех предложений. Внутри него инициализируется динамический массив строк *sentence*, строками которого будут слова из предложения. После этого текущая строка разбивается на слов по разделителям и слова записываются в массив строк sentence, все это происходит благодаря функции *split()*. Инициализируется переменная *big_flag*, благодаря которой можно будет понять, нужно ли удалять предложение или нет. После этого идет цикл, проходящий по индексам слов в массиве sentence, внутри которого инициализируются указатели на символы начала и конца текущего слова. Далее идет цикл *while*, происходящий до момента, пока указатель на начало строки меньше указателя на конец строки. В нем проверяется идентичность символов, если символы неидентичны, то переменной *flag* прибавляется 1, если же нет, то указатели сближаются друг к другу с обеих сторон на 1. Если после цикла значение *flag* равно 0, значит слово — палиндром, значит данное предложение нужно удалить, в связи с этим значению переменной *big_flag* присваивается 1, происходит выход из вложенного цикла. Если *big_flag* равно 1, то память массива

delete перевыделяется, увеличиваясь на один размер *int*, и в выделенное место записывается индекс текущего предложения. После этого память динамического массива строк sentence освобождается вначале построчно, потом по индексам строк.

После основного цикла происходит удаление найденных предложений при помощи функции *removes()*.

4.6. Функция 4

За обработку при поступлении команды 4 отвечает функция *void find_word(char ***text, int *count_sentence)*, которой на вход подаются указатель на динамический массив строк и указатель на переменную, хранящую в себе количесвто предложений. В функции происходит поиск предложений со словом *HiddenTiger*, стоящим не в начале предложения и удаление остальных предложений.

Вначале в функции инициализируется строка *find_str*, содержащая в себе слово *HiddenTiger*. Далее инициализируется массив целых чисел *delete*, в который будут записаны индексы предложений, которые нужно будет удалить. Далее идет цикл, проходящий по индексам всех предложений текста. Внутри него при помощи функции *strstr()* ищется вхождение строки *find_str* в текущее предложение. Если вхождение не было найдено, либо индекс начала слова совпадает с индексом начала предложения, то память массива delete перевыделяется, увеличиваясь на один размер int, и в выделенное место записывается индекс текущего предложения. В противном случае идет проверка на то, что если слово не входит в состав другого слова, а является отдельным, проверяется, являются ли символы слева от слова символами и разделителями и символы справа от слова символами разделителями либо точкой, если хоть один из этих случаев неверен, то память массива *delete* перевыделяется, увеличиваясь на один размер *int*, и в выделенное место записывается индекс текущего предложения.

После цикла выполняется удаление найденных предложений при помощи функции *removes()*.

4.7. Функция 5

За выполнение команды 5 отвечает функция *void function_info()*, которая выводит информацию о программе на экран. Функция реализована при помощи функции выводы *puts()*, внутри которой находться вся информация.

5. ДОПОЛНИТЕЛЬНЫЕ ФУНКЦИИ

5.1. Сведения о файлах

К дополнительным функция относятся функции компараторы *cmp()* и *compare()*, функция разделения слов в предложении *split()*, прототипы которых находятся в файле *additional_processing.h*, функция удаления предложений из текста *removes()*, прототип которой находится в файле *removes.h* и функция вывода ошибки *fatal_error(),* прототип которой находится в файле *fatal_error.h*.

Описания функция находятся в файлах *additional_processing.c*, *removes.c* и *fatal_error.c* соответственно.

5.2. Функция cmp().

Функция *int cmp(const void *first, const void *second)* получает на вход указатели на константы *first* и *second* типа *const void* *, и возвращает целочисленное значение. Она отвечает за сравнение двух строк в лексикографическом порядке посимвольно вне зависимости от регистра.

Внутри инициализтированны константы *f* и *s*, при помощи приведения типов к *const char* ** констант *first* и *second* соответственно. Далее идет цикл *while*, выполняющийся до момента, пока длина какой-либо строки не закончится. внутри идет сравнение символов, если символ первой строки по коду больше символа второго, то при помощи оператора return возвращается *1*, если меньше, возвращается *-1*, если они равны, переходит к следующим символам. Если цикл завершился, значит сравненные элементы строк одинаковы, и длина хотя бы одной строки закончилась. Если строка 1 еще не закончилась, то возвращается *1*, в противном случае, если строка 2 еще не закончилась, то возвращается *-1*, если обе строки закончились, возвращается *0*.

5.3. Функция compare().

Функция *int compare(const void *first, const void *second)* получает на вход указатели на константы first и second типа *const void* *, и возвращает целочисленное значение. Она отвечает за сравнение по модулю двух чисел.

Внутри инициализтированны константы *f* и *s*, при помощи приведения типов к *const int* * констант *first* и *second* соответственно. Далее проверяется, если первое число по модулю меньше второго, то возвращается *1*, если больше, возвращается *-1*, если равны, возвращается *0*. Модуль числа берется при помощи функции *abs()*.

5.4. Функция split().

Функция *void split(char ***text, char ***sentence, int *count, int i)* получает на вход указатель на динамический массив строк *text*, указатель на динамический массив строк *sentence*, указатель на переменную, хранящую в себе количество слов и переменную типа *int*, хранящую в себе индекс текущего предложения. Функция разделяет предложение по символам разделителям на слова.

Внутри функции выделяется память под динамическую строку new_string, в которую при помощи функции *strncpy()* копируются символы из текущего предложения, кроме точки. В последний символ записывается *\0*. Далее при помощи функции *strtok()* строка разбивается на слова по разделителям. Далее идет цикл *while*, выполняющийся до момента, пока не закончатся слова. Внутри память sentence перевыделяется и становиться на размер одного указателя больше, выделяется память для новой строки, равной длине слова, при помощи функции *strncpy()* в строку копируется слово, последний символ строки становится *\0*. При помощи функции *strtok()* в строке word оказывается новое слово. Счетчик количества слов увеличивается на 1.

5.5. Функция removes().

Функция *void removes(char ***text, int *count_sentence, int **delete, int len)* получает на вход указатель на динамический массив строк, указатель на пременную, в которой хранится количество предложений, указатель на динамический массив целых чисел и длина данного массива. Функция отвечает за удаление из динамического массива строк строк, с индексами, записанными в динамический массив целых чисел.

Вначале функции происходит сортировка динамического массива чисел *delete* при помощи функции *qsort()*, за сравнение элементов отвечает функция *compare()*. После этого идет цикл, проходящий по массиву *delete*. При помощи функции *free()* очищается память строки с индексом, равным значению текущего элемента массива *delete*, после чего все указатели массива строк сдвигаются на *-1*. Счетчик количества предложений уменьшается на *1*. Память, выделенная под динамический массив строк перевыделяется и становиться на один развер указателя меньше, чем была.

После цикла память, выделенная под массив *delete* очищается при помощи функции *free()*.

5.6. Функция fatal_error().

Функция *void fatal_error()* отвечает за вывод информации об ошибке и завершение программы.

Внутри выводиться информацию об ошибке при помощи функции *puts()*, после чего программа завершается при помощи функции *exit()*.

6. ФУНКЦИЯ ВЫВОДА

Вывод текста реализован в функции *void output_text(char ***text, int *count_sentence)*, описание которой находится в файле *output_text.c*. Ей на вход подается указатель на динамический массив строк и указатель на переменную, хранящую в себе количество предложений. Внутри функции происходит вывод предложений с последующим освобождением памяти.

Внутри идет цикл, который проходит по индексам всех предложений. Внутри цикла происходит вывод текущего предложения при помощи функции *puts()*, после чего освобождение памяти данной строки при помощи функции *free()*. после цикла идет очистка памяти динамического массива строк по индексам.

7. MAKEFILE

Функции в курсовой работе находятся в разных файлах, поэтому для сборки проекта был создан *Makefile*. Он выполняет линковку и компиляцию с помощью компилятора. Каждый заголовочный файл имеет защиту от повторного подключения при помощи директив условной компиляции. Внутри есть независимая цель *all*, которая собирает все объектные файлы в исполняемый файл *cw* и независимая цель *clean*, которая очищает все объектыне файлы и испольняемый файл. Каждая подцель отвечает за свой объектный файл, каждый файл **.o* соответствует файлу **.c*, и зависит от тех же заголовочных файлов, что и файл **.c*.

ЗАКЛЮЧЕНИЕ

При выполнении курсовой работы были изучены методы работы с динамической памятью и обработки текста в языке С. Также были изучены и использованы функции стандартных библиотек. Была создана программа, которая принимает на вход команду и текст, после чего обрабатывает его определенным для каждой команды образом.

В программе были использованы как функции стандартных библиотек, так и функции, написанные собственными руками. Функции были разбиты на разные файлы. Для сборки файлов был написан Makefile.

Программа была протестирована, прошла проверку на курсе.

Примеры работы программы см. в приложении А.

Исходный код программы см. в приложении Б.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

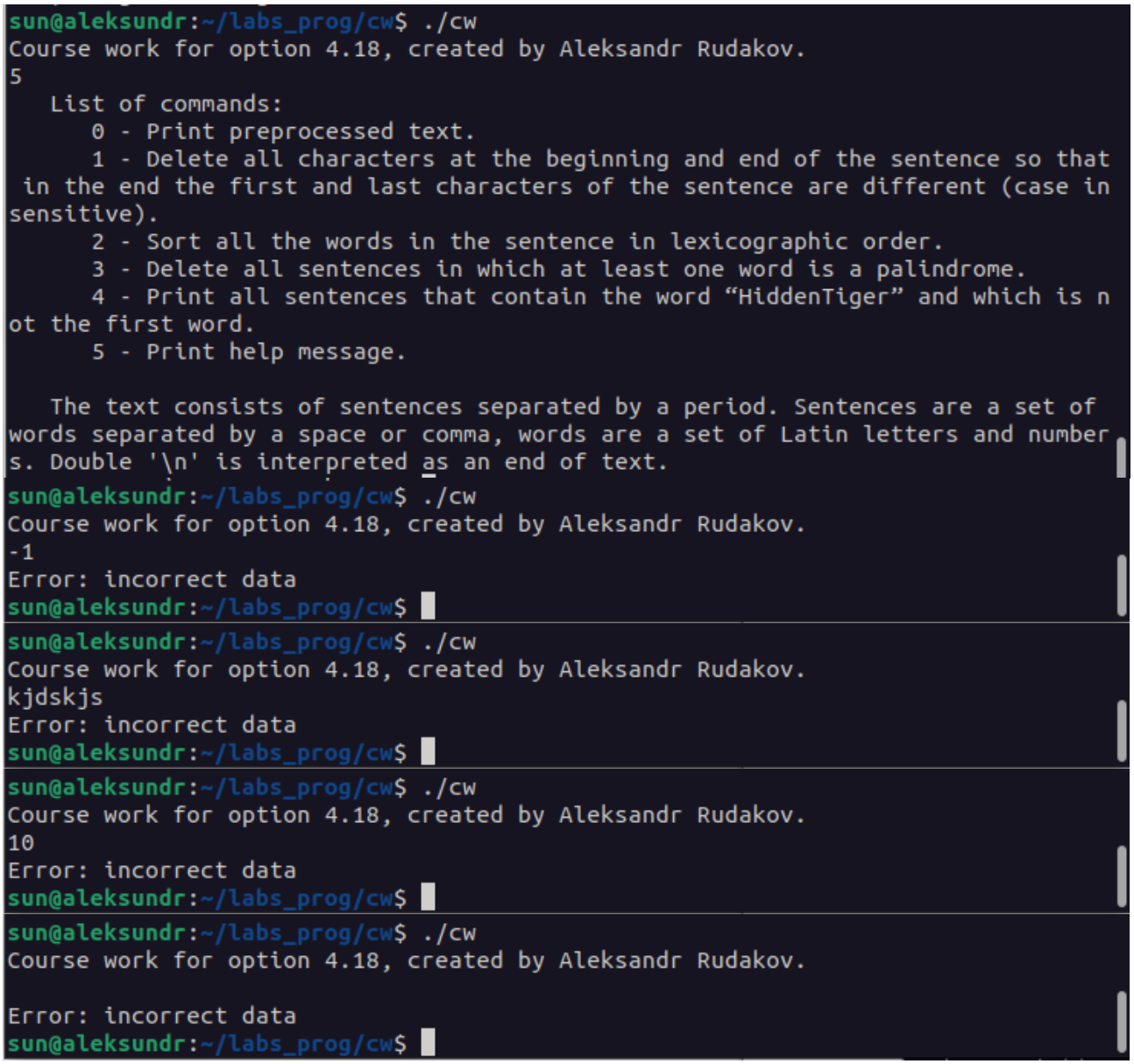
1. *Язык программирования Си / Б. Керниган, Д. Ритчи. Пер. с англ., 3-е изд., испр. – СПб.: «Невский диалект», 2001. 352 с: ил.*
2. *Описание синтаксиса некоторых функция и примеры их использования //* http://cppstudio.com/
3. *Решение некоторых ошибок //* https://stackoverflow.com/

ПРИЛОЖЕНИЕ А

ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ ПРОГРАММЫ

```
sun@aleksundr:~/labs_prog/cw$ ./cw
Course work for option 4.18, created by Aleksandr Rudakov.
0
Pam switch case 1. Qwe. Fre Fred FRue ofnf. roch 123 321 jdor.
Dragon flew away. HiddenTiger free sumh. UwU hash kit.
123, king HiddenTiger. I'm came feed will calli.
DOG,GREEN,SOOS. 42 no 24. Feet growHiddenTiger, grouf. QWE.

Pam switch case 1.
Qwe.
Fre Fred FRue ofnf.
roch 123 321 jdor.
Dragon flew away.
HiddenTiger free sumh.
UwU hash kit.
123, king HiddenTiger.
I'm came feed will calli.
DOG,GREEN,SOOS.
42 no 24.
Feet growHiddenTiger, grouf.
sun@aleksundr:~/labs_prog/cw$ ./cw
Course work for option 4.18, created by Aleksandr Rudakov.
1
Pam switch case 1. Qwe. Fre Fred FRue ofnf. roch 123 321 jdor.
Dragon flew away. HiddenTiger free sumh. UwU hash kit.
123, king HiddenTiger. I'm came feed will calli.
DOG,GREEN,SOOS. 42 no 24. Feet growHiddenTiger, grouf. QWE.

Pam switch case 1.
Qwe.
re Fred FRue ofn.
ch 123 321 jd.
Dragon flew away.
iddenTiger free sum.
UwU hash kit.
123, king HiddenTiger.
'm came feed will call.
DOG,GREEN,SOOS.
no.
eet growHiddenTiger, grou.
```

```
sun@aleksundr:~/labs_prog/cw$ ./cw
Course work for option 4.18, created by Aleksandr Rudakov.
2
Pam switch case 1. Qwe. Fre Fred FRue ofnf. roch 123 321 jdor.
Dragon flew away. HiddenTiger free sumh. UwU hash kit.
123, king HiddenTiger. I'm came feed will calli.
DOG,GREEN,SOOS. 42 no 24. Feet growHiddenTiger, grouf. QWE.

1 case Pam switch.
Qwe.
Fre Fred FRue ofnf.
123 321 jdor roch.
away Dragon flew.
free HiddenTiger sumh.
hash kit UwU.
123 HiddenTiger king.
calli came feed I'm will.
DOG GREEN SOOS.
24 42 no.
Feet grouf growHiddenTiger.
sun@aleksundr:~/labs_prog/cw$ ./cw
Course work for option 4.18, created by Aleksandr Rudakov.
3
Pam switch case 1. Qwe. Fre Fred FRue ofnf. roch 123 321 jdor.
Dragon flew away. HiddenTiger free sumh. UwU hash kit.
123, king HiddenTiger. I'm came feed will calli.
DOG,GREEN,SOOS. 42 no 24. Feet growHiddenTiger, grouf. QWE.

Qwe.
Fre Fred FRue ofnf.
roch 123 321 jdor.
Dragon flew away.
HiddenTiger free sumh.
123, king HiddenTiger.
I'm came feed will calli.
42 no 24.
Feet growHiddenTiger, grouf.
sun@aleksundr:~/labs_prog/cw$ ./cw
Course work for option 4.18, created by Aleksandr Rudakov.
4
Pam switch case 1. Qwe. Fre Fred FRue ofnf. roch 123 321 jdor.
Dragon flew away. HiddenTiger free sumh. UwU hash kit.
123, king HiddenTiger. I'm came feed will calli.
DOG,GREEN,SOOS. 42 no 24. Feet growHiddenTiger, grouf. QWE.

123, king HiddenTiger.
```


sun@aleksundr:~/labs_prog/cw$ ./cw
Course work for option 4.18, created by Aleksandr Rudakov.
5
List of commands:
0 - Print preprocessed text.
1 - Delete all characters at the beginning and end of the sentence so that in the end the first and last characters of the sentence are different (case in sensitive).
2 - Sort all the words in the sentence in lexicographic order.
3 - Delete all sentences in which at least one word is a palindrome.
4 - Print all sentences that contain the word “HiddenTiger” and which is not the first word.
5 - Print help message.
The text consists of sentences separated by a period. Sentences are a set of words separated by a space or comma, words are a set of Latin letters and numbers. Double '\n' is interpreted as an end of text.
sun@aleksundr:~/labs_prog/cw$ ./cw
Course work for option 4.18, created by Aleksandr Rudakov.
-1
Error: incorrect data
sun@aleksundr:~/labs_prog/cw$
sun@aleksundr:~/labs_prog/cw$ ./cw
Course work for option 4.18, created by Aleksandr Rudakov.
kjdskjs
Error: incorrect data
sun@aleksundr:~/labs_prog/cw$
sun@aleksundr:~/labs_prog/cw$ ./cw
Course work for option 4.18, created by Aleksandr Rudakov.
10
Error: incorrect data
sun@aleksundr:~/labs_prog/cw$
sun@aleksundr:~/labs_prog/cw$ ./cw
Course work for option 4.18, created by Aleksandr Rudakov.
Error: incorrect data
sun@aleksundr:~/labs_prog/cw$

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "scan_text.h"
#include "processing_functions.h"
#include "additional_processing.h"
#include "fatal_error.h"
#include "output_text.h"

int main(){
     printf("Course work for option 4.18, created by Aleksandr
Rudakov.\n");
     char flag, flag2;
     int point;
     flag=getchar();
     if (flag=='\n'){
          fatal_error();
     }else if (isdigit(flag)==0){
          fatal_error();
     }else{
          flag2=getchar();
          if (flag2!='\n'){
               fatal_error();
          }else {
               int f=flag-'0';
               if ((f>5) || (f<0)){
                    fatal_error();
               }
          }
     }
     point=flag-'0';
     int count_sentence=0;
        char **text=(char **)malloc(sizeof(char*));
     if (point!=5){
          scan_text(&text,&count_sentence);
          mandatory_processing(&text,&count_sentence);
     }
     switch (point){
          case 0:
               break;
          case 1:
               delete_while_the_same(&text,&count_sentence);
               break;
          case 2:
               sort_lexicographic_order(&text,&count_sentence);
               break;
          case 3:
               remove_palindromes(&text,&count_sentence);
               break;
```

```
            case 4:
                  find_word(&text,&count_sentence);
                  break;
            case 5:
                  function_info();
                  break;
            default:
                  fatal_error;
      }
      if (point!=5){
            output_text(&text,&count_sentence);
      }
      return 0;
}
```

Файл: scan_text.h

```
#ifndef SCAN_TEXT
#define SCAN_TEXT

void scan_text(char ***text, int *count_sentence);

#endif
```

Файл: scan_text.c

```
#include <stdio.h>
#include <stdlib.h>
#include "scan_text.h"

void scan_text(char ***text, int *count_sentence){
      int flag=0;
      int count_flag=0;
      char symbol;
      int count_symbol=0;
      (*text)[0]=(char *)malloc(sizeof(char));
      while (flag<2){
            symbol=getchar();
            if (symbol=='\n'){
                  flag++;
                  continue;
            } else if (count_symbol==0  && (symbol==' ' || symbol=='\n' ||
symbol=='\t')){
                        continue;
            }else{
                  (*text)[*count_sentence]=(char
*)realloc((*text)[*count_sentence],sizeof(char)*(count_symbol+1));
                  (*text)[*count_sentence][count_symbol]=symbol;
                  count_symbol++;
                  count_flag=(*count_sentence)+1;
                  flag=0;
            }
            if (symbol=='.'){
                  (*text)[*count_sentence]=(char
*)realloc((*text)[*count_sentence],sizeof(char)*(count_symbol+1));
                        (*text)[*count_sentence][count_symbol]='\0';
                  count_symbol=0;
                  (*count_sentence)++;
```

```
                (*text)=(char **)realloc((*text),sizeof(char
*)*((*count_sentence)+1));
                (*text)[*count_sentence]=(char *)malloc(sizeof(char));
          }
     }
     if (count_flag>(*count_sentence)){
          if ((*text)[*count_sentence][count_symbol-1]!='.'){
                (*text)[*count_sentence]=(char
*)realloc((*text)[*count_sentence], sizeof(char)*(count_symbol+2));
                (*text)[*count_sentence][count_symbol]='.';
                (*text)[*count_sentence][count_symbol+1]='\0';
                (*count_sentence)++;
          }
     }
}
```

Файл: processing_functions.h

```
#ifndef PROCESSING_FUNCTIONS
#define PROCESSING_FUNCTIONS

void mandatory_processing(char ***text, int *count_sentence);

void delete_while_the_same(char ***text, int *count_sentence);

void sort_lexicographic_order(char ***text, int *count_sentence);

void remove_palindromes(char ***text, int *count_sentence);

void find_word(char ***text, int *count_sentence);

void function_info();

#endif
```

Файл: processing_functions.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "processing_functions.h"
#include "additional_processing.h"
#include "removes.h"

void mandatory_processing(char ***text, int *count_sentence){
     int *delete=(int *)malloc(sizeof(int));
     int len=0;
     for (int i=0; i<(*count_sentence)-1; i++){
          for (int j=i+1; j<(*count_sentence); j++){
               if (strlen((*text)[i])!=strlen((*text)[j])){
                    continue;
               } else{
                    int count=0;
                    for (int k=0; k<strlen((*text)[i]);k++){
                         if
(toupper((*text)[i][k])!=toupper((*text)[j][k])) count++;
                    }
                    if (count==0){
```

```
                              int j1=0;
                              for (int l=0; l<len; l++){
                                    if (delete[l]==j){
                                          j1=1;
                                    }
                              }
                              if (j1==0){
                                    delete=(int
*)realloc(delete,sizeof(int)*(len+1));
                                    delete[len]=j;
                                    len++;
                              }
                        }
                  }
            }
      }
      removes(&(*text),&(*count_sentence),&delete,len);
      for (int i=0; i<(*count_sentence); i++){
            int point=strlen((*text)[i])-2;
            if (point>0){
                  char c=(*text)[i][point];
                  while ((c==' ' || c==',') && point>0){
                        point--;
                        c=(*text)[i][point];
                  }
                  (*text)[i]=(char
*)realloc((*text)[i],sizeof(char)*(point+3));
                  (*text)[i][point+1]='.';
                  (*text)[i][point+2]='\0';
            }
      }
}

void delete_while_the_same(char ***text, int *count_sentence){
      for (int i=0; i<(*count_sentence); i++){
            char *new_string=(char
*)(malloc(sizeof(char)*strlen((*text)[i])));
            strncpy(new_string,(*text)[i],strlen((*text)[i])-1);
            new_string[strlen((*text)[i])-1]='\0';
            char *ptr_begin=new_string;
            char *ptr_end=new_string+strlen(new_string)-1;
            while (ptr_begin<ptr_end){
                  if (toupper(*ptr_begin)==toupper(*ptr_end)){
                        ptr_begin++;
                        ptr_end--;
                  } else break;
            }
            free((*text)[i]);
            int len=ptr_end-ptr_begin+1;
            (*text)[i]=(char *)malloc(sizeof(char)*(len+2));
            strncpy((*text)[i],ptr_begin,len);
            (*text)[i][len]='.';
            (*text)[i][len+1]='\0';
            free(new_string);
      }
}
```

```
void sort_lexicographic_order(char ***text, int *count_sentence){
      for (int i=0; i<(*count_sentence); i++){
            char **sentence=(char **)malloc(sizeof(char *));
            int count=0;
            split(&(*text),&sentence,&count,i);
            if (count!=0){
                  free((*text)[i]);
                  qsort(sentence,count,sizeof(char *),cmp);
                  int new_len=strlen(sentence[0])+1;
                  (*text)[i]=(char *) malloc(sizeof(char)*new_len);
                  strncpy((*text)[i],sentence[0],new_len);
                  for (int j=1; j<count; j++){
                        (*text)[i][new_len-1]=' ';
                        (*text)[i][new_len]='\0';
                        new_len+=strlen(sentence[j])+1;
                        (*text)[i]=(char
*)realloc((*text)[i],sizeof(char)*(new_len+1));
                        strcat((*text)[i],sentence[j]);
                  }
                  (*text)[i][new_len-1]='.';
                  (*text)[i][new_len]='\0';
                  for (int j=0; j<count; j++){
                        free(sentence[j]);
                  }
                  free(sentence);
            }
      }
}

void remove_palindromes(char ***text, int *count_sentence){
      int *delete=(int *)malloc(sizeof(int));
      int len=0;
      for (int i=0; i<(*count_sentence); i++){
            char **sentence=(char **)malloc(sizeof(char*));
            int count=0;
            split(&(*text),&sentence,&count,i);
            int big_flag=0;
            for (int j=0; j<count; j++){
                  int flag=0;
                  char *ptr_begin=sentence[j];
                  char *ptr_end=sentence[j]+strlen(sentence[j])-1;
                  while (ptr_begin<ptr_end){
                        if (toupper(*ptr_begin)!=toupper(*ptr_end)) {
                              flag++;
                        }
                        ptr_begin++;
                        ptr_end--;
                  }
                  if (flag==0){
                        big_flag=1;
                        break;
                  }
            }
            if (big_flag==1){
                  delete=(int *)realloc(delete,sizeof(int)*(len+1));
                  delete[len]=i;
```

```
                len++;
            }
            for (int j=0; j<count; j++){
                free(sentence[j]);
            }
            free(sentence);
        }
        removes(&(*text),&(*count_sentence),&delete,len);
}

void find_word(char ***text, int *count_sentence){
        char find_str[]="HiddenTiger";
        int *delete=(int *)malloc(sizeof(int));
        int len=0;
        for (int i=0; i<*count_sentence; i++){
            char *ptr = strstr((*text)[i],find_str);
            if (ptr==NULL || ptr==(*text)[i]){
                delete=(int*)realloc(delete,sizeof(int)*(len+1));
                delete[len]=i;
                len++;
            } else {
                if (*(ptr-1)!=' ' && *(ptr-1)!=','){
                    delete=(int*)realloc(delete,sizeof(int)*(len+1));
                        delete[len]=i;
                           len++;
                } else if (*(ptr+11)!=' ' && *(ptr+11)!=',' &&
*(ptr+11)!='.'){
                        delete=(int*)realloc(delete,sizeof(int)*(len+1));
                            delete[len]=i;
                            len++;
                }
            }
        }
        removes(&(*text),&(*count_sentence),&delete,len);
}

void function_info(){
        puts("   List of commands:\n"
           "      0 - Print preprocessed text.\n"
           "      1 - Delete all characters at the beginning and end of the
sentence so that in the end the first and last characters of the sentence
are different (case insensitive).\n"
           "      2 - Sort all the words in the sentence in lexicographic
order.\n"
           "      3 - Delete all sentences in which at least one word is a
palindrome.\n"
           "      4 - Print all sentences that contain the word
“HiddenTiger” and which is not the first word.\n"
           "      5 - Print help message.\n"
           "\n"
           "   The text consists of sentences separated by a period.
Sentences are a set of words separated by a space or comma, words are a
set of Latin letters and numbers. Double \'\\n\' is interpreted as an end
of text.");
}
```

Файл: additional_processing.h

```
#ifndef ADDITIONAL_PROCESSING
#define ADDITIONAL_PROCESSING

int cmp(const void *first, const void *second);

int compare(const void *first, const void *second);

void split(char ***text, char ***sentence, int *count, int i);

#endif
```

Файл: additional_processing.c

```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "additional_processing.h"

int cmp(const void *first, const void *second){
        const char **f=(const char **)first;
        const char **s=(const char **)second;
        int i=0, j=0;
        while (i<strlen(*f) && j<strlen(*s)){
                if (toupper((*f)[i])>toupper((*s)[i])){
                        return 1;
                } else if (toupper((*f)[i])<toupper((*s)[i])) return -1;
                i++;
                j++;
        }
     if (i<strlen(*f)){
          return 1;
     } else if (j<strlen(*s)){
          return -1;
     }else return 0;
}

int compare(const void *first, const void *second){
        const int *f = (const int*) first;
        const int *s = (const int*) second;
        if (abs(*f)<abs(*s)){
                return 1;
        } else if (abs(*f)<abs(*s)){
                return -1;
        }
        return 0;
}

void split(char ***text, char ***sentence, int *count, int i){
     int len=strlen((*text)[i]);
     char *new_string=malloc(sizeof(char)*len);
     strncpy(new_string,(*text)[i],len-1);
     new_string[len-1]='\0';
     char *word=strtok(new_string," ,");
     while (word!=NULL){
          (*sentence)=realloc((*sentence),sizeof(char*)*((*count)+1));
          (*sentence)[*count]=malloc(sizeof(char)*(strlen(word)+1));
          strncpy((*sentence)[*count],word,strlen(word));
          (*sentence)[*count][strlen(word)]='\0';
```

```
            word=strtok(NULL," ,");
            (*count)++;
      }
}
```

Файл: removes.h

```
#ifndef REMOVE
#define REMOVE

void removes(char ***text, int *count_sentence, int **delete, int len);

#endif
```

Файл: removes.c

```
#include <stdlib.h>
#include "removes.h"
#include "additional_processing.h"

void removes(char ***text, int *count_sentence, int **delete, int len){
      qsort(*delete,len,sizeof(int),compare);
      for (int i=0; i< len; i++){
            free((*text)[(*delete)[i]]);
            for (int j=(*delete)[i]+1; j<*count_sentence; j++){
                  (*text)[j-1]=(*text)[j];
            }
            (*count_sentence)--;
            (*text)=(char
**)realloc((*text),sizeof(char*)*(*count_sentence));
      }
      free(*delete);
}
```

Файл: fatal_error.h

```
#ifndef FATAL_ERROR
#define FATAL_ERROR

void fatal_error();

#endif
```

Файл: fatal_error.c

```
#include <stdlib.h>
#include <stdio.h>
#include "fatal_error.h"

void fatal_error(){
      puts("Error: incorrect data");
      exit(0);
}
```

Файл: output_text.h

```
#ifndef OUTPUT_TEXT
#define OUTPUT_TEXT

void output_text(char ***text, int *count_sentence);
```

```
#endif
```

Файл: output_text.c

```
#include <stdio.h>
#include <stdlib.h>
#include "output_text.h"

void output_text(char ***text, int *count_sentence){
        for (int i=0; i<*count_sentence; i++){
                        puts((*text)[i]);
                        free((*text)[i]);
                }
        free(*text);
}
```

Файл: Makefile

```
all: main.o scan_text.o processing_functions.o additional_processing.o
removes.o output_text.o fatal_error.o
     gcc main.o scan_text.o processing_functions.o
additional_processing.o removes.o output_text.o fatal_error.o -o cw
scan_text.o: scan_text.c scan_text.h
     gcc -c scan_text.c
processing_functions.o: processing_functions.c processing_functions.h
additional_processing.h
     gcc -c processing_functions.c
additional_processing.o: additional_processing.c additional_processing.h
     gcc -c additional_processing.c
removes.o: removes.c removes.h additional_processing.h
     gcc -c removes.c
output_text.o: output_text.c output_text.h
     gcc -c output_text.c
fatal_error.o: fatal_error.c fatal_error.h
     gcc -c fatal_error.c
main.o: main.c processing_functions.h additional_processing.h scan_text.h
output_text.h fatal_error.h
     gcc -c main.c
clean:
     rm cw *.o
```