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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: обработка строк на языке Си

Студент гр. 0382 _____ Осинкин Е. А.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Осинкин Е. А.

Группа 0382

Тема работы: обработка строк на языке Си

Исходные данные:

Программе на вход подается текст (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).

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. Удалить все четные по счету предложения в которых четное количество слов.
2. Отсортировать все слова в предложениях по возрастанию количества букв в верхнем регистре в слове.
3. Заменить все слова в тексте длина которых не более 3 символов на подстроку “Less Then 3”.
4. Найти в каждом предложении строку максимальной длины, которая начинается и заканчивается цифрой. Вывести найденные подстроки по убыванию длины подстроки.

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.

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Осинкин Е. А.

Преподаватель _____ Жангиров Т. Р.

АННОТАЦИЯ

В процессе выполнения курсовой работы была реализована программа для обработки текст на языке Си. Для хранения текста использовался двумерный динамический массив. Обработка и вывод текста производились при помощи использования функций следующих стандартных заголовочных файлов: stdio.h, stdlib.h, string.h, ctype.h. Для удобства пользователя реализован вывод на консоль контекстного меню выбора необходимой опции обработки текста.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной курсовой работе производится разработка стабильной консольной программы, производящей обработку введённого пользователем текста в соответствии с выбранной пользователем опцией обработки введённого текста.

Программа реализована при помощи использования двумерного динамического массива (он используются для хранения данных о введённом тексте). Память под текст в программе выделяется динамически при помощи использования стандартных функций выделения памяти из стандартного заголовочного файла stdlib.h. Также использовались функции из заголовочных файлов: stdio.h, string.h, ctype.h для обработки текста.

Программа разработана на языке Си. Разработка велась при помощи интерактивной среды разработки Microsoft Visual Studio 2017.

1. ЦЕЛИ И ЗАДАЧИ

Цель: разработка стабильной программы, способной выполнять считывание текста и его обработку в соответствии с заданием.

Для достижения цели необходимо выполнить следующие задачи:

- Реализовать считывание текста;
- Написать функции обработки текста в соответствии с заданием;
- Произвести отладку программы.

2. ХОД ВЫПОЛНЕНИЯ

2.1 Функции для считывания текста

Для начала пользователю предлагается ввести текст. Для этого реализована одна функция:

Функция *void scan_text(char*** text, int* size)* предназначена для чтения, записи текста и получения количества предложений в тексте. В начале выделяется память для двумерного динамического массива *sents* с помощью *malloc* для одного предложения, если предложений будет больше выделяется дополнительная память с помощью метода *realloc*. Аналогично в цикле выделяется память для одномерного динамического массива *sent*. Также в цикле не записываются все лишние разделители слов в начале и в конце предложения, не записываются разделители, которые идут подряд, если такие есть. Запись предложения заканчивается, когда пользователь вводит точку. Запись текста заканчивается, когда пользователь нажимает Enter. В конце переменной *size* присваивается количество предложений, а в *text* присваивается значение *sents*.

2.2 Первичная обработка

Требуется найти и удалить все повторно встречающиеся предложения (сравнить их посимвольно, но без учета регистра). Для этого реализовано две функции.

Функция *bool is_similar_sents(char* strA, char* strB)* принимает две динамические строки и возвращает *true*, если эти строки одинаковые без учета регистра, и *false*, если это не так. Сравниваем посимвольно, с помощью метода *tolower()* из библиотеки *ctype.h* для того, что бы привести символы к одинаковому регистру.

Функция *void delete_similar_sents(char*** text, int* size)* предназначена для удаления одинаковых предложений в тексте. Предложение сравнивается со всеми предложениями, которые идут дальше, при помощи функции *is_similar_sents().* Если предложения похожи то, предложение с большим порядковым номером приравнивается к “ \0”. Записываем в new_text все предложения не равные “ \0” и с помощью метода *realloc* выделяем нужное количество памяти. В конце переменной *size* присваивается новое количество предложений, а в *text* присваивается значение *new_text*.

2.3 Выбор операции

Функция *int menu()* предназначена для информирования пользователя о возможных преобразованиях текста и получения значения переменной *command* для дальнейшего преобразования текса. Переменная command может принимать значение 0 – 4: 0 – выход из программы, 1 – 4 преобразования текста, описанных в задание.

2.4 Первая операция

Требуется удалить все четные по счету предложения, в которых четное количество слов. Для этого реализовано две функции.

Функция *int word_count(char* sent)* принимает предложение и возвращает количество строк. Так как лишних разделителей нет в текст, нужно лишь посчитать количество разделителей, чтобы узнать количество слов.

Функция *void delete_even_sents(char*** text, int* size)* имеет похожую структуру, как и у функции *delete_similar_sents.* Только приравнивать к “ \0” нужно те предложения, которые имеют четное количество слов и четные по счету. Первого условие проверяется при помощи функции word_count. Второе при помощи порядкового номера предложения. Записываем в new_text все предложения не равные “ \0” и с помощью метода *realloc* выделяем нужное количество памяти. В конце переменной *size* присваивается новое количество предложений, а в *text* присваивается значение *new_text*.

2.5 Вторая операция

Требуется отсортировать все слова в предложениях по возрастанию количества букв в верхнем регистре в слове. Для этого реализовано две функции и одна структура.

Структура *typedef struct word,* в которой храниться слово из предложения *wrd* и количество символов в верхнем регистре *count_uppercase*.

Функция *int countcmp(const void* i, const void* j),* которая нужная для функции *qsort* для сортировки слов в предложении по количеству символов в верхнем регистре.

Функция *word_sort(char** text, int size)*, которая обрабатывает данный текст. Для каждого предложения создаем массив из структур *word*, записываем все слова в *wrd* и считаем для них значение *count_uppercase*. После этого пользуемся сортировкой *qsort* из библиотеки *stdlib.h* при помощи функции *countcmp.* Дальше создаем новую строку при помощи функции *strcat* из библиотеки *string.h.* Последовательно прибавляем слова отсортированные по нужному параметру и расставляем разделители слов в нужном порядке. Приравниваем *text[i]* к новой строке. В конце очищаем память при помощи функции *free*.

2.6 Третья операция

Требуется заменить все слова в тексте, длина которых не более 3 символа на подстроку “Less Then 3”. Для этого реализована одна функция.

Функция *void less_then_3(char** text, int size),* предназначена для обработки данного текста. Запускаем цикл для каждого предложения и считаем там количество символов для слова. Если оно больше трех, то записывает это слово в *new_string*. Если оно меньше или равно трем, то записываем в *new_string* строку “Less Then 3”. Разделители слов в новом предложении не меняем. В конце приравниваем *text[i]* к новой строке.

2.7 Четвертая операция

Требуется найти в каждом предложении строку максимальной длины, которая начинается и заканчивается цифрой. Вывести найденные подстроки по убыванию количества строк. Для этого реализовано 4 функции и одна структура.

Функция *bool is_2_number(char* str),* получает на вход строку и должна вернуть *true*, если в строке две или более цифры, и *false*, если наоборот. При помощи цикла считаем количество цифр, если оно меньше или равно двум возвращаем false, иначе возвращаем true.

Функция *substring(char* str)*, получает на вход строку и возвращает подстроку максимальной длины, которая начинается и заканчивается цифрой. Для начала находим первую и последнюю цифру, которая встречается в предложении, и записываем их индексы. Затем создаем новую строку и записывает символы из нужного диапазона индексов старой строки. Возвращаем новую строку.

Структура *typedef struct sub*, хранить в себе подстроку и длину этой подстроки.

Функция *sizecmp(const void* i, constt void* j),* нужна для функции *qsort* для сортировки найденных подстрок в порядке убывания их длины.

Функция *substrings(char** text, int size)* получает на вход текст и количество предложений. Сначала функция создает двумерный динамический массив строк и записывает в него нужные подстроки с помощью функций *is_2_number* и *substring*. Затем создает структуру *sub* и записывает туда подстроки и их длину. Потом сортирует их в нужном порядке при помощи функции *qsort* и *sizecmp*. Выводит нужные подстроки в нужном порядке. И чистит память при помощи функции *free*.

2.8 Вывод текста

Требуется реализовать вывод текста. Для этого реализована одна функция.

Функция *print_text(char** text, int size)* вывод текст на экран с помощью цикла *for* и метода *printf*.

2.9 Очистка памяти

Требуется очистить память двумерного динамического массива. Для этого реализована одна функция.

Функция *free_text(char** text, int size)* последовательно очищает память при помощи *free*.

2.10 Функция Main

Функция *int main()* сначала создает две переменный: *char** text, int size*. С помощью scan_text() записывает в *text* полученный на вход текст и получает количество предложений в тесте *size*. Делает первичную обработку текста с помощью функции *delete_similar_text* и печатает его с помощью функции *print_text*. Затем с помощью функции *menu()* получает нужную команду и при помощи *switch* определяет какую подзадачу нужно выполнить. С помощью функций *delete_even_sents, word_sort, less_then_3, substrings* выполняет указанные подзадачи и выводит измененный текст при помощи функции print_text, если это нужно. В конце очищает память, выделенную под текст с помощью функции *free_text* и возвращает 0.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было создано предложение для обработки текста согласно запросам пользователя. Все задачи также были успешно выполнены: программа может удалять повторяющиеся предложения; удалять все четные по счету предложения, в которых четное количество слов; отсортировать все слова в предложениях по возрастанию количества букв в верхнем регистре в слове; заменять все слова в тексте длина которых не более 3 символов на подстроку “Less Then 3”; найти в каждом предложении строку максимальной длины, которая начинается и заканчивается цифрой, вывести найденные подстроки по убыванию длины подстроки. Также программа выводит измененный текст на экран.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сайт www.c-cpp.ru

ПРИЛОЖЕНИЕ А
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

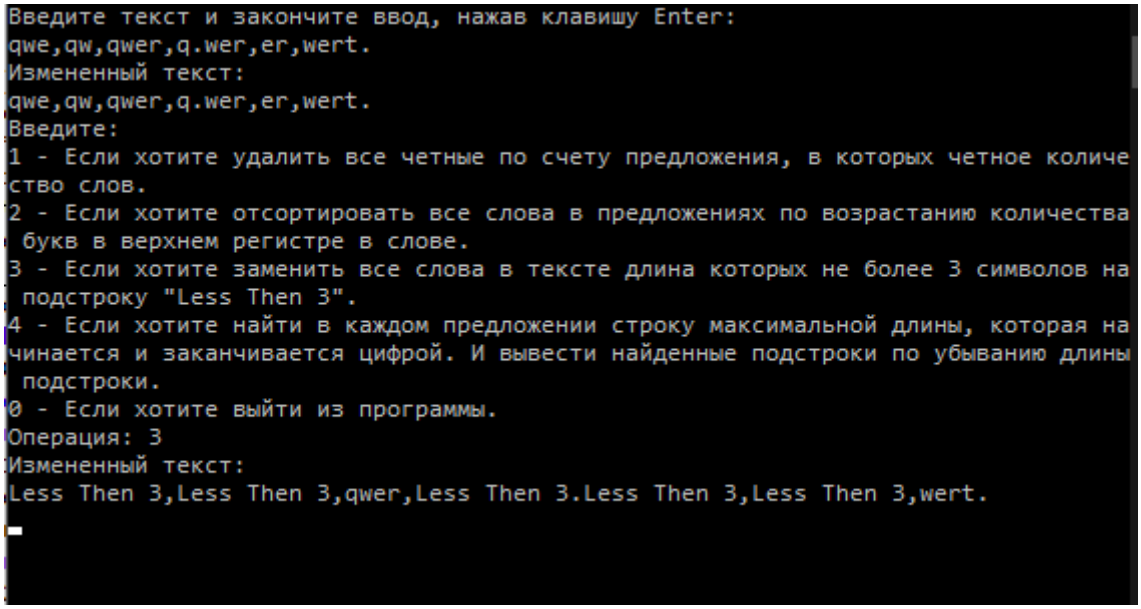
1) Пример

```
Введите текст и закончите ввод, нажав клавишу Enter:
QWE.QWER,qwer.qwe.QWEER,qwe.
Измененный текст:
QWE.QWER,qwer.QWEER,qwe.
Введите:
1 - Если хотите удалить все четные по счету предложения, в которых четное количество с
лов.
2 - Если хотите отсортировать все слова в предложениях по возрастанию количества букв
в верхнем регистре в слове.
3 - Если хотите заменить все слова в тексте длина которых не более 3 символов на подст
року "Less Then 3".
4 - Если хотите найти в каждом предложении строку максимальной длины, которая начинает
ся и заканчивается цифрой. И вывести найденные подстроки по убыванию длины подстроки.
0 - Если хотите выйти из программы.
Операция: 1
Измененный текст:
QWE.QWEER,qwe.
```

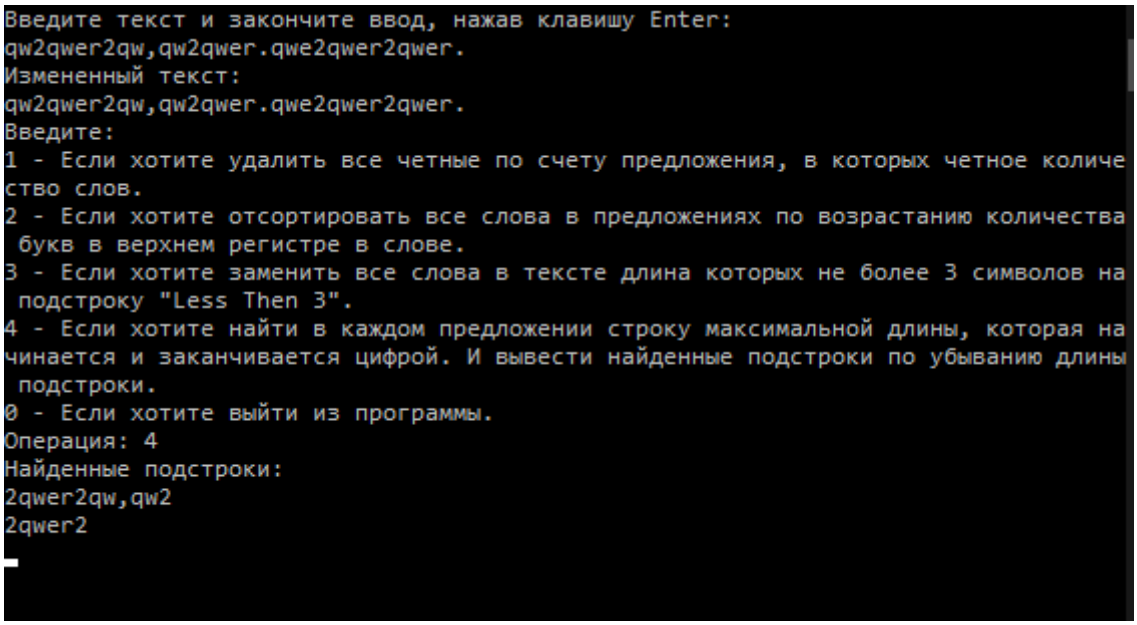
2) Пример

```
Введите текст и закончите ввод, нажав клавишу Enter:
QWE,qwe.qwe,qwe.QWer,qwER.QWEe,qweR.
Измененный текст:
QWE,qwe.QWer,qwER.QWEe,qweR.
Введите:
1 - Если хотите удалить все четные по счету предложения, в которых четное количе
ство слов.
2 - Если хотите отсортировать все слова в предложениях по возрастанию количества
 букв в верхнем регистре в слове.
3 - Если хотите заменить все слова в тексте длина которых не более 3 символов на
 подстроку "Less Then 3".
4 - Если хотите найти в каждом предложении строку максимальной длины, которая на
чинается и заканчивается цифрой. И вывести найденные подстроки по убыванию длины
 подстроки.
0 - Если хотите выйти из программы.
Операция: 2
Измененный текст:
qwe,QWE.qwER,QWer.qweR,QWEe.
```

3) Пример



4) Пример



ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

//___SCAN
void scan_text(char*** text, int* size) {
     printf("Введите текст и закончите ввод, нажав клавишу Enter:\n");
     char chr = '\0';
     int size_sent = 1;
     int count_sents = 1;
     char** sents = (char**)malloc(count_sents * sizeof(char*));
     while (chr != '\n') {
          scanf("%c", &chr);
          if  ((chr != '\n') && (chr != ' ') && (chr != ',')) {
               char* sent = (char*)malloc(size_sent *
sizeof(char));
               sent[size_sent - 1] = chr;
               while (chr != '.') {
                    scanf("%c", &chr);
                    if (!(((chr == ' ') || (chr == ',')) &&
((sent[size_sent - 1] == ' ') || (sent[size_sent - 1] == ',')))) {
                         if ((chr == '.') && ((sent[size_sent - 1]
== ' ') || (sent[size_sent - 1] == ','))) {
                              printf("%c\n", sent[size_sent - 1]);
                              sent[size_sent - 1] = '.';
                              break;
                         }
                         size_sent++;
```

```
                    sent = (char*)realloc(sent, size_sent *
sizeof(char));
                    sent[size_sent - 1] = chr;
                }
            }
            sent = (char*)realloc(sent, (size_sent + 1) *
sizeof(char));
            sent[size_sent] = '\0';
            size_sent = 1;
            if (count_sents != 1)
                sents = (char**)realloc(sents, (count_sents) *
sizeof(char*));
            sents[count_sents - 1] = sent;
            count_sents++;
        }
    }
    count_sents--;
    *size = count_sents;
    *text = sents;
}

//___DELETE_SIMILAR
bool is_similar_sents(char* strA, char* strB) {
    int i = 0;
    while ((strA[i] != '.') && (strB[i] != '.')) {
        if ((char)tolower(strA[i]) != (char)tolower(strB[i])) {
            return 0;
        }
        i++;
    }
    if (strA[i] != strB[i]) {
        return 0;
    }
    else {
        return 1;
```

```
     }
}

void delete_similar_sents(char*** text, int* size) {
     char** tmp_text = *text;
     char** new_text = *text;
     int new_size = 0;
     for (int i = 0; i < *size - 1; i++) {
          for (int j = i + 1; j < *size; j++) {
               char* strA = tmp_text[i];
               char* strB = tmp_text[j];
               if (is_similar_sents(strA, strB)) {
                    tmp_text[j] = " \0";
               }
          }
     }
     for (int i = 0; i < *size; i++) {
          if (tmp_text[i] != " \0") {
               new_text[new_size] = tmp_text[i];
               new_size++;
          }
     }
     new_text = (char**)realloc(new_text, new_size *
sizeof(char*));
     *text = new_text;
     *size = new_size;
}


//___1
int word_count(char* sent) {
     int i = 0;
     int count = 0;
     while (sent[i] != '.') {
          if ((sent[i] == ' ') || (sent[i] == ',')) {
```

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

void delete_even_sents(char*** text, int* size) {
        char** tmp_text = *text;
        int number_sent = 1;
        for (int i = 0; i < *size; i++) {
            if ((number_sent % 2 == 0) && (word_count(tmp_text[i]) %
2 == 0)) {
                tmp_text[i] = " \0";
            }
            number_sent++;
        }
        char** new_text = *text;
        int new_size = 0;
        for (int i = 0; i < *size; i++) {
            if (tmp_text[i] != " \0") {
                new_text[new_size] = tmp_text[i];
                new_size++;
            }
        }
        new_text = (char**)realloc(new_text, new_size *
sizeof(char*));
        *text = new_text;
        *size = new_size;
}


//___2
typedef struct word {
```

```
     char* wrd;
     int count_uppercase;
};

int countcmp(const void* i, const void* j) {
     word* new_i = (word*)i;
     word* new_j = (word*)j;
     return new_i->count_uppercase - new_j->count_uppercase;
}

void word_sort(char** text, int size) {
     for (int i = 0; i < size; i++) {
          char* tmp_string = text[i];
          word* words = (word*)malloc(word_count(tmp_string) *
sizeof(word));
          for (int j = 0; j < word_count(tmp_string); j++) {
               words[j].wrd = (char*)malloc(strlen(tmp_string) *
sizeof(char));
               words[j].count_uppercase = 0;
          }
          char* separators = (char*)malloc(word_count(tmp_string) *
sizeof(char));
          int size_word = -1;
          int count_word = 0;
          for (int j = 0; j < strlen(tmp_string); j++) {
               size_word++;
               words[count_word].wrd[size_word] = tmp_string[j];
               if ((words[count_word].wrd[size_word] >= 65) &&
(words[count_word].wrd[size_word] <= 90)) {
                    words[count_word].count_uppercase++;
               }
               if ((words[count_word].wrd[size_word] == ' ') ||
(words[count_word].wrd[size_word] == ',') ||
(words[count_word].wrd[size_word] == '.')) {
```

```
                    separators[count_word] =
words[count_word].wrd[size_word];
                    words[count_word].wrd[size_word] = '\0';
                    size_word = -1;
                    count_word++;
               }
          }
          qsort(words, count_word, sizeof(word), countcmp);
          char* new_string = (char*)malloc(1 * sizeof(char));
          new_string[0] = '\0';
          for (int j = 0; j < count_word; j++) {
               new_string = (char*)realloc(new_string,
strlen(new_string) + strlen(words[j].wrd) + 2 * sizeof(char));
               strcat(new_string, words[j].wrd);
               new_string[strlen(new_string) + 1] = '\0';
               new_string[strlen(new_string)] = separators[j];
          }
          free(text[i]);
          text[i] = new_string;
          count_word = 0;
          for (int j = 0; j < count_word; j++) {
               free(words[j].wrd);
          }
          free(words);
     }
}

//___3
void less_then_3(char** text, int size) {
     char** new_text = text;
     for (int i = 0; i < size; i++) {
          char* tmp_string = text[i];
          char* new_string = (char*)malloc(1 * sizeof(char));
          new_string[0] = '\0';
          int size_string = 0;
```

```
          int size_word = 0;
          int size_new_string = 0;
          while (tmp_string[size_string] != '\0') {
               if ((tmp_string[size_string] == ',') ||
(tmp_string[size_string] == '.') || (tmp_string[size_string] == '
')) {
                    if (size_word <= 3) {
                         new_string = (char*)realloc(new_string,
strlen(new_string) + 13 * sizeof(char));
                         strcat(new_string, "Less Then 3");
                         new_string[strlen(new_string) + 1] = '\0';
                         new_string[strlen(new_string)] =
tmp_string[size_string];
                         size_new_string += 12;
                    }
                    else {
                         new_string = (char*)realloc(new_string,
strlen(new_string) + size_word + 2 * sizeof(char));
                         for (int j = 0; j <= size_word; j++) {
                              new_string[size_new_string] =
tmp_string[size_string - size_word + j];
                              size_new_string++;
                         }
                         new_string[size_new_string] = '\0';
                    }
                    size_word = -1;
               }
               size_word++;
               size_string++;
          }
          text[i] = new_string;
     }
}

//___4
```

```
bool is_2_number(char* str) {
	int i = 0;
	int count = 0;
	while (str[i] != '.') {
		if ((str[i] >= '0') && (str[i] <= '9')) {
			count++;
		}
		i++;
	}
	if (count >= 2) {
		return 1;
	}
	else {
		return 0;
	}
}

char* substring(char* str) {
	int i = 0;
	int bgn_str = 0;
	int end_str = 0;
	while (str[i] != '.') {
		if ((str[i] >= '0') && (str[i] <= '9')) {
			bgn_str = i;
			break;
		}
		i++;
	}
	i++;
	while (str[i] != '.') {
		if ((str[i] >= '0') && (str[i] <= '9')) {
			end_str = i;
		}
		i++;
	}
```

```
     i = 0;
     char* substr = (char*)malloc((end_str - bgn_str + 2) *
sizeof(char));
     int subsize = 0;
     while (str[i] != '.') {
          if ((i >= bgn_str) && (i <= end_str)) {
               substr[subsize] = str[i];
               subsize++;
          }
          i++;
     }
     substr[subsize] = '\0';
     return substr;
}

typedef struct sub {
     char* str;
     int size;
} sub;

int sizecmp(const void* i, const void* j) {
     sub* new_i = (sub*)i;
     sub* new_j = (sub*)j;
     return new_j->size - new_i->size;
}

void substrings(char** text, int size) {
     int new_size = 0;
     char** new_text = (char**)malloc((new_size + 1) *
sizeof(char*));
     for (int i = 0; i < size; i++) {
          if (is_2_number(text[i])) {
               new_text[new_size] = substring(text[i]);
               new_size++;
```

```
                new_text = (char**)realloc(new_text, (new_size + 1)
* sizeof(char*));
            }
        }
        sub* subs = (sub*)malloc(new_size * sizeof(sub));
        for (int i = 0; i < new_size; i++) {
            subs[i].str = new_text[i];
            subs[i].size = strlen(new_text[i]);
        }
        qsort(subs, new_size, sizeof(sub), sizecmp);
        printf("Найденные подстроки:\n");
        for (int i = 0; i < new_size; i++) {
            printf("%s\n", subs[i].str);
        }
        for (int i = 0; i < new_size; i++) {
            free(new_text[i]);
        }
        free(new_text);
}

//___MENU
int menu() {
        printf("Введите:\n1 - Если хотите удалить все четные по счету
предложения, в которых четное количество слов.\n");
        printf("2 - Если хотите отсортировать все слова в предложениях по
возрастанию количества букв в верхнем регистре в слове.\n");
        printf("3 - Если хотите заменить все слова в тексте длина которых не
более 3 символов на подстроку “Less Then 3”.\n");
        printf("4 - Если хотите найти в каждом предложении строку
максимальной длины, которая начинается и заканчивается цифрой. И вывести
найденные подстроки по убыванию длины подстроки.\n");
        printf("0 - Если хотите выйти из программы.\n");
        printf("Операция: ");
        int command = 0;
```

```
    scanf_s("%d", &command);
    return command;
}

//___PRINT
void print_text(char** text, int size) {
    printf("Измененный текст:\n");
    for (int i = 0; i < size; i++) {
        printf("%s", text[i]);
    }
    printf("\n");
}

//___FREE
void free_text(char** text, int size) {
    for (int i = 0; i < size; i++) {
        free(text[i]);
    }
    free(text);
}

int main() {
    char** text;
    int size;
    scan_text(&text, &size);
    delete_similar_sents(&text, &size);
    print_text(text, size);
    int command = menu();
    switch (command) {
    case 0:
        return 0;
        break;
    case 1:
        delete_even_sents(&text, &size);
        print_text(text, size);
```

```
            break;
        case 2:
            word_sort(text, size);
            print_text(text, size);
            break;
        case 3:
            less_then_3(text, size);
            print_text(text, size);
            break;
        case 4:
            substrings(text, size);
            break;
        default:
            break;
        }
        free_text(text, size);
        return 0;
}
```