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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка строк на языке С

Студент гр. 3388 _____ Пьянков М.Ф.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2023

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Пьянков М.Ф.

Группа 3388

Тема работы: Обработка строк на языке С

Исходные данные:

Вариант 4.16

Вывод программы должен быть произведен в стандартный поток вывода: stdout.

Ввод данных в программе в стандартный поток ввода: stdin.

В случае использования Makefile название исполняемого файла должно быть: cw.

Важно: первой строкой при запуске программы нужно выводить информацию о варианте курсовой работе и об авторе программы в строго определенном формате:

Course work for option <V>, created by <Name> <Surname>.

Где V – вариант курсовой и Имя и Фамилия, как указано в репозитории группы. Данное предложение должно быть строго первым предложением в выводе программы и является отдельной строкой (заканчивается знаком ‘\n’).

Например:

Course work for option 3.2, created by Ivan Ivanov.

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:

Вывести каждое предложение, и если в нем есть словосочетание “define BLUE”, то все слова после этого словосочетания вывести голубым цветом.

Во всем тексте заменить последовательность подряд идущих букв без учета регистра подстрокой вида “<кол-во подряд идущих букв><буква в верхнем регистре>”. Например, последовательность “AaaA” должна быть заменена строкой “4A”.

Удалить все предложения в которых количество слов кратно трем.

Отсортировать предложения по уменьшению суммарной длины первого и последнего слова. Если в предложении одно слово, то необходимо брать длину этого слова.

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.

Дата выдачи задания: 11.11.2023

Дата сдачи реферата: 6.12.2023

Дата защиты реферата: 8.12.2023

Студент гр. 3388 _____ Пьянков М. Ф.

Преподаватель _____ Заславский М. М.

АННОТАЦИЯ

Курсовая работа заключается в создании программы на языке С для обработки текста. Для хранения и обработки текста были использованы различные функции стандартных библиотек языка С.

После запуска первой строкой программа выводит подсказку о том, что нужно ввести текст. Далее программа считывает текст и удаляет повторяющиеся предложения, без учёта регистра.

После первичной обработки текста программа выводит вторую подсказку пользователю о доступных преобразованиях над текстом.

Далее выполняется обработка текста согласно введённой пользователем командой и его вывод.

SUMMARY

The course work consists of creating a program in C for text processing. Various functions of standard C libraries were used to store and process text.

After starting the first line, the program displays a hint indicating that you need to enter text.

Next, the program reads the text and removes duplicate sentences, without taking into account case.

After the initial processing of the text, the program displays a second hint to the user about the available transformations on the text.

Next, the text is processed according to the command entered by the user and output.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы:

Написать программу, обрабатывающую текст, согласно запросу пользователя.

Подробная характеристика запросов пользователя изложена в разделе ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ (стр. 2)

Основные теоретические положения:

Для хранения текста создаётся двумерный динамический массив типа char**.

1. Библиотеки: stdio.h, stdlib.h, string.h, ctype.h, malloc.h, strings.h.
2. Функции: Обработка текста: strcpy(), malloc(), tolower(), toupper(), strlen(), realloc(), snprintf(), qsort(), strstr(). Ввод-вывод: scanf(), , getchar(), puts(), printf(), Работа с инструкциями: switch().

   Для работы со строками используются функции из библиотеки string.h.

   Для сортировки используется функция qsort().

   Ввод: getchar(), scanf().

   Вывод: printf(), puts().

ХОД ВЫПОЛНЕНИЯ РАБОТЫ

Ввод текста

Ввод реализован через стандартный поток ввода stdin, используемые функции: scanf(), getchar(), см. ПРИЛОЖЕНИЕ Б.

Функция char** text_input(int* text_length)

Функция начинается с выделения памяти для динамического массива text. Далее внутри цикла while() динамически выделяется память для предложения. Внутри вложенного цикла while() считывается символ с потока ввода. Идёт проверка на корректность ввода (исключени ситуаций: «,,,,», « ,» и т. п.) и запись символа в предложение. Когда на вход подаётся символ «.», предложение заканчивается вложенный цикл прерывается и предложение записывается в динамический массив text.

Первичная обработка текста

Первичная обработка включает в себя удаление повторяющихся предложений. В данной программе представлена функциями void erase(char** text, int *text_length, int index) и void removing_repetitions(char** text, int* text_length), для подробностей см. ПРИЛОЖЕНИЕ Б.

Функция void erase(char** text, int *text_length, int index)

С помощью этой функции реализовано удаление элемента массива путём сдвига всех элементов массива до удаляемого влево и освобождением памяти под последний элемент.

Функция void removing_repetitions(char** text, int* text_length)

Эта функция позволяет непосредственно удалить повторяющиеся предложения. Предложения перебираются с помощью двух циклов while(), один из которых вложен в другого. Два предложения сравниваются посредством функции strcasecmp(text[i], text[j]), и если они равны, то удаляется предложение, встреченное позже, см. ПРИЛОЖЕНИЕ Б.

Ввод кода операции

Пользователю представлена возможность ввести код необходимой инструкции, которые изложены на 3 стр., ввод кода осуществляется с помощью функции scanf().

Функция void key_input(int* key)

Запись значения из потока ввода в переменную key по указателю.

Выполнение операций

Для каждой операции была написана отдельная функция. Все функции запускаются в функции void switch_func(char** text, int* text_length, int key).

Функция void switch_func(char** text, int* text_length, int key)

Функция представлена конструкцией switch(key), где вызываются отдельные функции для операций, соответствующих своему коду.

Нулевая операция

Вывод предложений текста с помощью функции puts(). Пример работы операции см. в приложении А, Рисунок 2.

Первая операция

Вывод текста синим цветом осуществлен с помощью функции printf() и её параметров. Пример работы операции см. в приложении А, Рисунок 3.

Функция void first_func(char** text, int text_length)

В функции циклично перебираются предложения. Во вложенном цикле с помощью функции strstr(text[i], BLUE) проверяется, является ли подстрокой предложения строка "define BLUE". Если является, то так как функция strstr() возвращает указатель на индекс символа, с которого начинается вхождение подстроки, то это значение записывается в отдельную переменную index, к ней прибавляется длина строки "define BLUE", вычисляемая с помощью

функции strlen(BLUE). Далее первым вложенным циклом в обычном виде выводятся все символы до значения переменной index. Следующим циклом, начинающимся со значения index, выводим символы синим цветом с помощью специального параметра функции printf(): "\e[1;34m%c\e[m", см. ПРИЛОЖЕНИЕ Б.

Вторая операция

Предложения в исходном тексте заменяются на предложения, в которых повторяющиеся символы заменены на следующий вид: <число повторений><символ в верхнем регистре>. Пример работы операции см. в приложении А, Рисунок 4.

Функция void second_func(char** text, int text_length)

Циклично перебираем все предложения исходного текста.

На каждой итерации создаётся новое предложение, путём выделения динамической памяти для строки. Далее берётся первый символ исходного предложения и сравнивается со следующим, если они равны, то счётчик повторений увеличивается. Если счётчик повторений не равен 0, то число повторений записывается в строку с помощью фукции snprintf(str, sizeof(str), "%d", symbol_count+1) и соответственно строка и символ в верхнем регистре записываются в новое предложение с помощью функции strcpy(new_sentence+new_sentence_length, str);. Далее в предложение добавляется символ «.» и «\0» и предложение перезаписывается на место исходного, см. ПРИЛОЖЕНИЕ Б.

Третья операция

Удаление предложений, в которых число слов кратно 3 осуществленно с помощью функций int words_count(char* sentence) и void third_func(char**

text, int* text_length). Пример работы операции см. в приложении А, Рисунок 5.

Функция int words_count(char* sentence)

В Данной функции возвращается количество слов в предложении, с помощью подсчёта количества пробелов. Количество слов равно кол-во пробелов+1.

Функция void third_func(char** text, int* text_length)

Релизован цикличный перебор предложений в тексте и подсчёт количества слов в них. Если количество оказывается кратно 3, то предложение удаляется с помощью функции void erase(char** text, int *text_length, int index), описанной выше, см. ПРИЛОЖЕНИЕ Б.

Четвёртая операция

Для сортировки предложений по суммарному количеству букв в первом и последнем словах предложения использована функция qsort() с функцией-компаратором: int compare_func(const void* first_sentence, const void* second_sentence). Для подсчёта суммарного количества символов в словах используется функция int words_length_counter(char* sentence). Пример работы операции см. в приложении А, Рисунок 6.

Функция int words_length_counter(char* sentence)

Сначала проверяется количество слов в предложении. Если слово одно, то считается только количество символов в одном слове. Если слов оказалось больше, то считается количество символов соответственно в первом и последнем словах с помощью двух циклов, один из которых начинается с 0, а другой — с длины предложения. Также при подсчёте символов не учитываются символы-разделители.

Функция int compare_func(const void* first_sentence, const void* second_sentence)

Функция, необходимая для применения алгоритма быстрой сортировки, реализуемого посредством функции qsort(). Функция возвращает разность между суммарным количеством символов в первом и последнем словах двух предложений.

Функция void fourth_func(char** text, int text_length)

Вызов функции сортировки qsort(text, text_length, sizeof(char*), compare_func);, см. ПРИЛОЖЕНИЕ Б.

Вывод информации о функциях

Вывод информации пользователю о возможных действиях над текстом.

Вывод будет реализован, если пользователь введёт код операции 5. Пример работы операции см. в приложении А, Рисунок 1.

Функции zero_func_info(), first_func_info(), second_func_info(), third_func_info(), fourth_func_info()

С помощью функции puts() реализован вывод описания функций.

Содержания описаний записаны в отдельных макросах: ZERO_FUNC, FIRST_FUNC, SECOND_FUNC, THIRD_FUNC, FOURTH_FUNC. см. ПРИЛОЖЕНИЕ Б.

ЗАКЛЮЧЕНИЕ

В результате написания курсовой работы было создано консольное приложение обрабатывающее текст согласно инструкциям пользователя: вывод информации о возможных операциях программы над текстом, вывод текста после первичной обработки, вывод части предложения, находящейся после сочетания define BLUE синим цветом, замена повторяющихся символов на вид <кол-во повторений><символ в верхнем регистре>, удаление из текста предложений с количеством слов, кратным 3, сортировка предложений по суммарному количеству символов в первом и последнем слове.

Все задачи были успешно выполнены. Предусмотрены случаи ошибочного ввода пользователем. Текст хранится в динамическом двумерном массиве. Все операции проводятся только над этим массивом.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Учебник по языку программирования С Брайан Керниган, Деннис Ритчи: https://www.r-5.org/files/books/computers/languages/c/kr/Brian_Kernighan_Dennis_Ritchie-The_C_Programming_Language-RU.pdf
2. Информация о цветном выводе в функции printf(): https://stackoverflow.com/questions/5412761/using-colors-with-printf

ПРИЛОЖЕНИЕ А. С ПРИМЕРАМИ РАБОТЫ ПРОГРАММЫ

Пример работы операции №5 представлен на рисунке 1

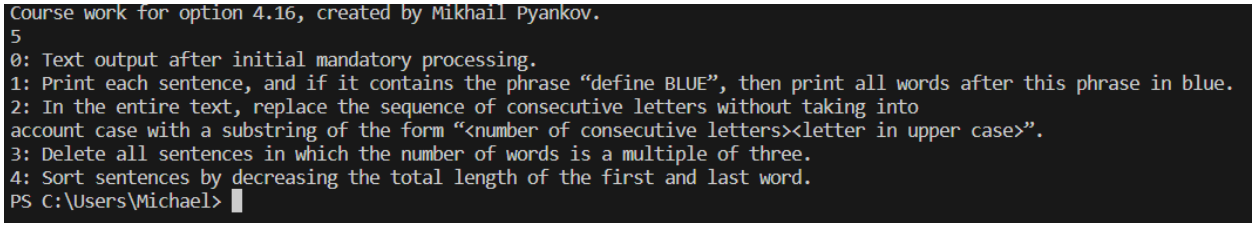

Рисунок 1 - Операция №5

Пример работы операции №0 представлен на рисунке 2

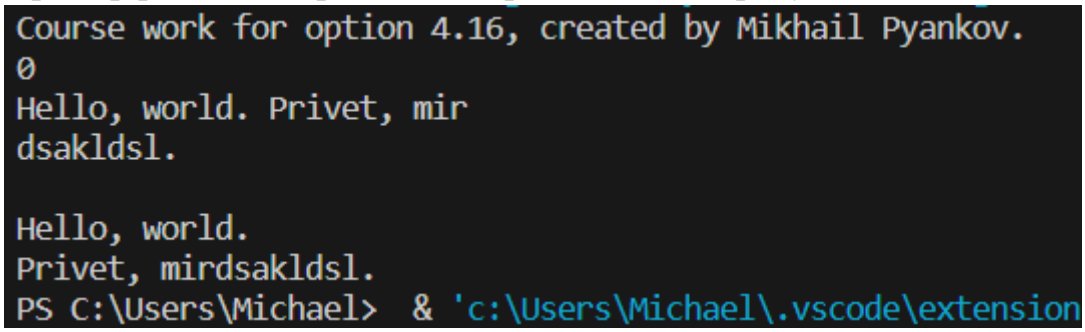

Рисунок 2 - Операция №0

Пример работы операции №1 представлен на рисунке 3

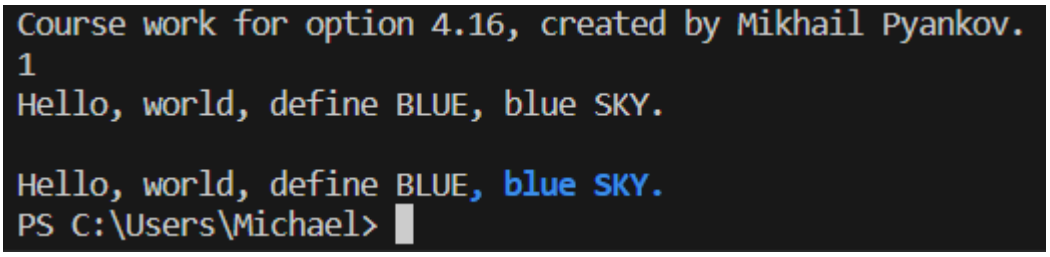

Рисунок 3 - Операция №1

Пример работы операции №2 представлен на рисунке 4

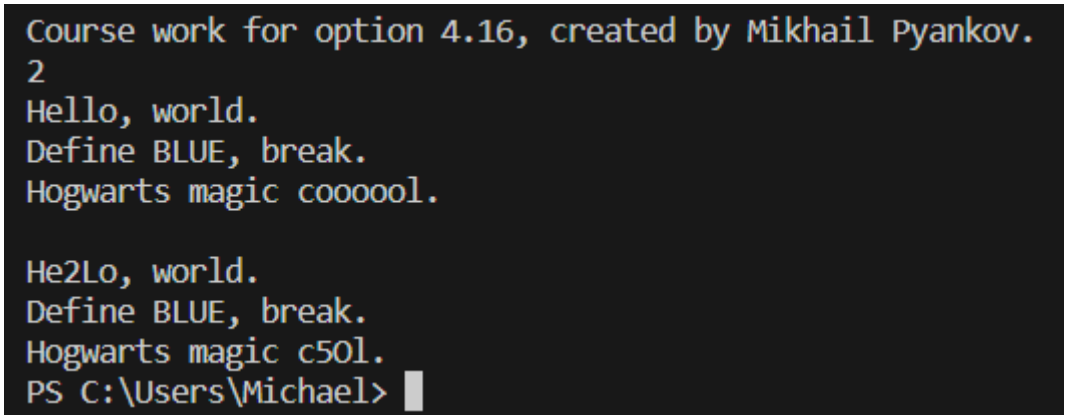

Рисунок 4 - Операция №2

Пример работы операции №3 представлен на рисунке 5

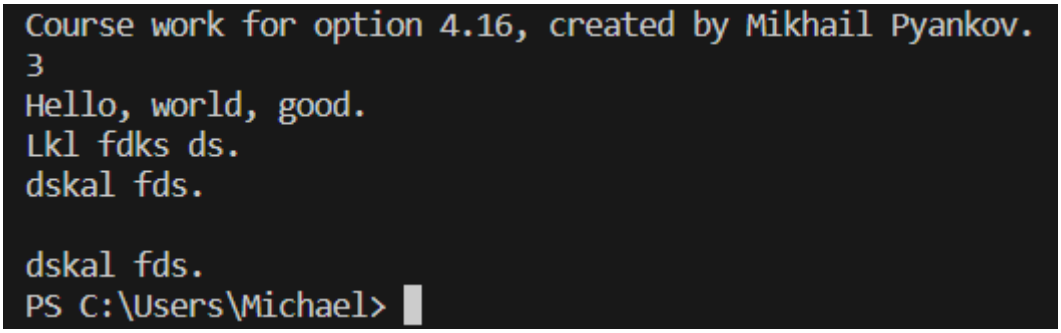

Рисунок 5 - Операция №3

Пример работы операции №4 представлен на рисунке 6

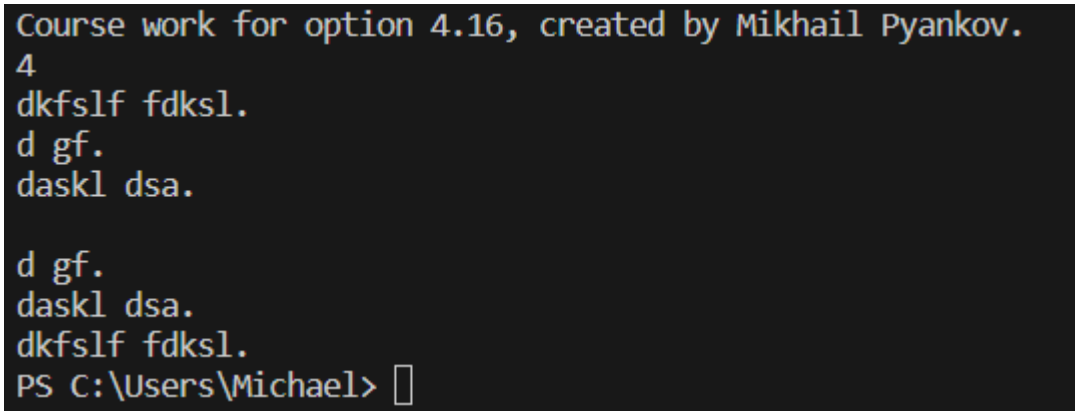

Рисунок 6 - Операция №4

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main_cw.c

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
#include<strings.h>
#include<string.h>
#include<ctype.h>
#define FIRST_MESSAGE "Course work for option 4.16, created by Mikhail
Pyankov.\0"
#define BASE_SIZE 300
#define BLUE "define BLUE"
#define ZERO_FUNC "0: Text output after initial mandatory processing.\
0"
#define FIRST_FUNC "1: Print each sentence, and if it contains the
phrase “define BLUE”, then print all words after this phrase in blue.\
0"
#define SECOND_FUNC "2: In the entire text, replace the sequence of
consecutive letters without taking into\naccount case with a substring
of the form “<number of consecutive letters><letter in upper case>”.\
0"
#define THIRD_FUNC "3: Delete all sentences in which the number of
words is a multiple of three.\0"
#define FOURTH_FUNC "4: Sort sentences by decreasing the total length
of the first and last word.\0"
#define ERROR_MSG "ERROR: Incorrect instruction.\0"
#define ERROR_DOT_MSG "ERROR: Forgotten dot.\0"
#define SPACE ' '
#define COMMA ','
#define DOT '.'
#define TAB '\t'
#define STR_END '\0'
#define NEXT_LINE '\n'
#define BLUE_CHAR "\e[1;34m%c\e[m"

void erase(char** text, int *text_length, int index)
{
    for(int i = index; i < *text_length-1; i++)
    {
        strcpy(text[i], text[i+1]);
    }
    free(text[*text_length-1]);
    *text_length-=1;
}

int words_count(char* sentence)
{
    int result = 0;
    for(int i = 0; i < strlen(sentence); i++)
    {
        if(sentence[i] == SPACE)
        result++;
    }
```

```
    return result+1;
}

int words_length_counter(char* sentence)
{
    if(words_count(sentence)==1)
    return strlen(sentence)-1;
    int result = 0;
    for(int i = 0 ; i < strlen(sentence); i++)
    {
        if(sentence[i] != COMMA && sentence[i] != SPACE)
        result++;
        if(sentence[i] == SPACE)
        break;
    }
    for(int i = strlen(sentence)-1; i >= 0; i--)
    {
        if(sentence[i] != COMMA && sentence[i] != SPACE)
        result++;
        if(sentence[i] == SPACE)
        break;
    }
    return result;
}

int    compare_func(const    void*    first_sentence,    const    void*
second_sentence)
{
          return     words_length_counter(*(char**)first_sentence)   -
words_length_counter(*(char**)second_sentence);
}

void removing_repetitions(char** text, int* text_length)
{
    int j;
    int i = 0;
    while(i < *text_length)
    {
        j = i + 1;
        while(j < *text_length)
        {
            if(strcasecmp(text[i], text[j])==0)
            erase(text, *&text_length, j);
            j++;
        }
        i++;
    }
}

void key_input(int* key)
{
    scanf("%d", key);
}

char** text_input(int* text_length)
```

```
{
    char** text = malloc(BASE_SIZE*sizeof(char*));
    char* sentence;
    int sentence_length;
    int sentence_size_count;
    int text_size_count = 1;
    char c;
    char previos_c;
    while(1)
    {
        sentence = malloc(BASE_SIZE*sizeof(char));
        sentence_length = 0;
        sentence_size_count = 1;
        while(1)
        {
            c = getchar();
            if(sentence_length >= BASE_SIZE*sentence_size_count)
            {
                sentence_size_count+=1;
                                        sentence  =  realloc(sentence,
(BASE_SIZE*sentence_size_count)*sizeof(char));
            }
              if(c != NEXT_LINE && !(sentence_length==0 && (c==SPACE||
c==TAB)))
            {
                sentence[sentence_length] = c;
                sentence_length+=1;
            }
            if(c == NEXT_LINE && previos_c == NEXT_LINE)
            {
                if(sentence_length!=0)
                {
                    sentence[sentence_length] = STR_END;
                    text[*text_length] = sentence;
                    *text_length+=1;
                }
                return text;
            }
            if(c == DOT)
            {
                sentence[sentence_length] = STR_END;
                break;
            }
            previos_c = c;
        }
        if(*text_length >= text_size_count)
        {
            text_size_count+=1;
                                          text    =    realloc(text,
(BASE_SIZE*text_size_count)*sizeof(char*));
        }
        text[*text_length] = sentence;
        *text_length+=1;
    }
}
```

```
void first_func(char** text, int text_length)
{
    int index;
    for(int i = 0; i < text_length; i++)
    {
        if(strstr(text[i], BLUE)!=NULL)
        {
                    index = (int)(strstr(text[i], BLUE) - text[i]) +
strlen(BLUE);
            for(int j = 0; j < index; j++)
            printf("%c", text[i][j]);
            for(int j = index; j < strlen(text[i]); j++)
            printf(BLUE_CHAR, text[i][j]);
            printf("\n");
        }
        else
        puts(text[i]);
    }
}

void second_func(char** text, int text_length)
{
    int new_sentence_length = 0;
    char* new_sentence;
    for(int i = 0; i < text_length; i++)
    {
        new_sentence = malloc(strlen(text[i]));
        new_sentence_length = 0;
        char symbol = text[i][0];
        int symbol_count;
        int j = 1;
        while(j < strlen(text[i])+1)
        {
            symbol_count = 0;
            while(tolower(text[i][j]) == tolower(symbol))
            {
                symbol_count +=1;
                j+=1;
            }
            if(symbol_count)
            {
                char str[100];
                snprintf(str, sizeof(str), "%d", symbol_count+1);
                strcpy(new_sentence+new_sentence_length, str);
                new_sentence_length+=strlen(str);
                new_sentence[new_sentence_length] = toupper(symbol);
                new_sentence_length+=1;

            }
            else
            {
                new_sentence[new_sentence_length] = symbol;
                new_sentence_length+=1;
            }
```

```
            symbol = text[i][j];
            j+=1;
        }
        new_sentence[new_sentence_length] = STR_END;
        text[i] = new_sentence;
    }
}

void third_func(char** text, int* text_length)
{
    int i = 0;
    while( i < *text_length)
    {
        if(words_count(text[i])%3 == 0)
        {
            erase(text, text_length, i);
            i--;
        }
        i++;
    }
}

void fourth_func(char** text, int text_length)
{
    qsort(text, text_length, sizeof(char*), compare_func);
}

int word_sum_digits(char* string)
{
    int sum = 0;
    for(int i = 0; i< strlen(string); i++)
    {
        if(string[i] - '0' >= 0 && string[i] - '0' < 10)
        {
            sum+=string[i] - '0';
        }
    }
    return sum;
}

int cmp_func(const void* first_string, const void* second_string)
{
                return    (word_sum_digits(*(char**)first_string)    -
word_sum_digits(*(char**)second_string));
}

char* split(char* string)
{
    char * pch = strtok (string," ");
    int i = 0;
    char** new_cifar_string = malloc(strlen(string)*sizeof(char*));
    while(pch !=NULL)
    {
        new_cifar_string[i] = pch;
        i++;
```

```
        pch = strtok(NULL, " ,.");

    }
    char* new_string = malloc(strlen(string)*sizeof(char)+10);
    new_string[0]='\0';
    qsort(new_cifar_string, i, sizeof(char*), cmp_func);
    for(int k = 0; k < i; k++)
    {
        strcat(new_string, new_cifar_string[k]);
        strcat(new_string, " ");
    }
    strcat(new_string, "\0");
    return new_string;
}

void nineth_func(char** text, int text_length)
{
    for(int i = 0; i < text_length; i++)
    {
        text[i] = split(text[i]);
    }
}

void switch_func(char** text, int* text_length, int key)
{
    switch(key)
    {
        case 0:
            for(int i = 0; i < *text_length; i++)
            puts(text[i]);
            break;
        case 1:
            first_func(text, *text_length);
            break;
        case 2:
            second_func(text, *text_length);
            break;
        case 3:
            third_func(text, text_length);
            break;
        case 4:
            fourth_func(text, *text_length);
            break;
        case 9:
            nineth_func(text, *text_length);
            break;
        default:
            puts(ERROR_MSG);
    }

}

void zero_func_info()
{
    puts(ZERO_FUNC);
```

```
}

void first_func_info()
{
    puts(FIRST_FUNC);
}

void second_func_info()
{
    puts(SECOND_FUNC);
}

void third_func_info()
{
    puts(THIRD_FUNC);
}

void fourth_func_info()
{
    puts(FOURTH_FUNC);
}

int main()
{
    puts(FIRST_MESSAGE);
    int key;
    key_input(&key);
    if(key == 5)
    {
        zero_func_info();
        first_func_info();
        second_func_info();
        third_func_info();
        fourth_func_info();
        return 0;
    }
    int text_length = 0;
    char** text = text_input(&text_length);
    if(text[text_length-1][strlen(text[text_length-1])-1]!=DOT)
    {
        puts(ERROR_DOT_MSG);
    }
    else
    {
        removing_repetitions(text, &text_length);
        switch_func(text, &text_length, key);
        if((key>1 && key<5) || key==9)
        switch_func(text, &text_length, 0);
    }
    return 0;
}
```