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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Работа со строками в языке С

Студентка гр. 1304 _____ Ярусова Т.В.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2021

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Ярусова Т.В.

Группа 1304

Тема работы : работа со строками в языке С

Исходные данные:
Текст, представляющий собой предложения, разделенные точкой. Предложения – набор слов, разделенные пробелом или запятой, слова – набор латинских букв и цифр.
Технические требования:
Текст должен быть сохранен в динамический массив строк.

Содержание пояснительной записки:
«Содержание», «Введение», «Основные теоретические положения», «Реализация программы», «Тестирование», «Пользовательская инструкция», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 15.10.2021

Дата сдачи реферата: 23.12.2021

Дата защиты реферата: 25.12.2021

Студентка _____ Ярусова Т.В.

Преподаватель _____ Чайка К.В

АННОТАЦИЯ

В курсовой работе была реализована программа, принимающая на вход текст и сохраняющая его в динамический массив строк, а затем предлагающая пользователю выбрать вариант обработки введенного текста. Были реализованы функции считывания текста, функция удаления повторно встречающихся предложений без учета регистра, функция подсчета слов “garbage” в предложениях без учета регистра, функция замены всех цифр на введенную подстроку, функция удаления предложений, в которых есть три подряд идущие буквы в верхнем регистре, функция сортировки по уменьшению количества слов, начинающихся с гласной буквы, функция вывода полученного результата и функция освобождения динамической памяти. Каждая подзадача реализована в виде отдельных функций. Использованы функции стандартных библиотек и написаны собственные.

SUMMARY

In the course work, a program was implemented that accepts text as input and saves it to a dynamic array of strings, and then prompts the user to choose the option for processing the entered text. The functions of reading text, the function of deleting repeated sentences without regard to case, the function of counting words "garbage" in sentences without regard to case, the function of replacing all digits with the entered substring were implemented, a function for deleting sentences in which there

are three consecutive uppercase letters, a function for sorting by reducing the number of words starting with a vowel, a function for outputting the result and a function for freeing dynamic memory. Each subtask is implemented as separate functions. The functions of the standard libraries have been used and our own have been written.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является разработка программы, на вход которой поступает текст. Программа должна сохранить этот текст в динамический массив строк и оперировать только с ним. Программа должна найти и удалить все повторно встречающиеся предложения. Далее, программа должна запрашивать у пользователя одно из следующих доступных действий и, впоследствии, выполнить его.

На основе выше изложенной цели, были сформулированы следующие задачи:

1. Изучение особенностей работы со строками и символами в языке С
2. Изучение особенностей работы с динамической памятью
3. Изучение использования указателей и массивов
4. Использование функций стандартных библиотек
5. Создание диалогового окна

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

1.1. Строки в языке программирования С

Строки в С, как и в большинстве языков программирования высокого уровня рассматриваются как отдельный тип, входящий в систему базовых типов языка. Так как язык С по своему происхождению является языком системного программирования, то строковый тип данных в С как таковой отсутствует, а в качестве строк в С используются обычные массивы символов, заканчивающийся нулевым символом.

1.2. Используемые библиотеки

В данной курсовой работе использовались такие библиотеки, как: stdio.h, stdlib.h, string.h, ctype.c.

Из библиотеки stdio.h были использованы следующие функции:

- getchar() – используется для считывания символа из стандартного потока ввода;
- puts() – используется для вывода строки в стандартный поток вывода;
- printf() – используется для вывода переданных аргументов в виде строки в стандартный поток вывода;

Из библиотеки stdlib.h были использованы следующие функции:

- malloc() – используется для динамического выделения блока памяти;
- realloc() – используется для изменения величины ранее выделенной памяти;
- free() – используется для возвращения памяти в кучу;

Из библиотеки string.h были использованы следующие функции:

- memmove() – используется для копирования n количество байт из одной области памяти в другую;

- strlen() – используется для вычисления длины строки;
- strcpy() – используется для копирования строки в другую строку;

Из библиотеки ctype.h были использованы следующие функции:

- tolower() – используется для замены символа в нижнем регистре на символ в верхнем регистре;
- isdigit() – используется для определения, является ли символ цифрой;
- isupper() – используется для определения, является ли символ буквой верхнего регистра;

2. РЕАЛИЗАЦИЯ ПРОГРАММЫ

2.1. Функция main

int main() – главная функция.

В функции *main()* объявляется переменная *char** text*, которая будет хранить в себе считанный текст, переменная *char sep*, в которой будет храниться знак конца предложения, переменная *int len*, в которой после выполнения функции *read_text()* будет храниться количество предложений в тексте, а после выполнения функции *delete_sentence()* количество предложений, оставшихся после удаления повторно встречающихся предложений (без учета регистра).

Открывается вечный цикл *while()*, в котором запрашивается номер команды, который с помощью функции *scanf()* будет записан в переменную *int value*.

С помощью оператора *switch* по ключу *value* совершается переход в один из *case*, в которых выполняется одна из предложенных команд пользователю. Программа завершится в том случае, когда пользователь введет значение *value* равное 5.

2.2. Функция считывания текста

Считывание текста реализуется с помощью двух функций: *read_text()* и *read_sentence()*.

read_text()

В функцию *int read_text()* подается на вход *char*** *text*- указатель на указатель на массив строк и *char sep* – символ конца предложения. Объявляются переменные *int count_sentence* – количество предложений в считанном тексте, *int memory* – изначальный блок памяти, выделенный для количества предложений текста, *char* sentence* – указатель на считанное предложение. С помощью блока

```
char** t = malloc(memory*sizeof(char*));
if(t != NULL){
     *text = t ;
}
else{
     puts("Ошибка! Недостаточно памяти!");
     return 0;
}
```
проверяется наличие свободной памяти, для ее динамического выделения.

С помощью цикла *do{}while()* происходит считывание предложений текста с использованием функции *read_sentence()*. Цикл завершается тогда, когда встретится символ переноса строки.

Из функции *read_text()* с помощью функции *return* вернется количество предложений текста, которое хранится в переменной *count_sentence*.

read_sentence()

В функцию *char* read_sentence()* подается на вход *char sep*, в которой хранится знак конца предложения. Объявляются переменные *int count_char* – количество символов в считанном предложении, *int memory_for_char* – изначальный блок памяти, выделенный для количества символов предложения, *char* sentence* – указатель на считанное предложение, *char c* – считанный символ из стандартного потока ввода. Также производится проверка на наличие свободной памяти.

С помощью цикла *do{}while()* происходит посимвольное считывание предложения с помощью функции *getchar()*. Цикл завершается тогда, когда встретится символ конца предложения, который хранится в переменной *sep*.

После цикла в конец предложения записывается нулевой символ, означающий конец строки.

Из функции *read_sentence()* с помощью функции *return* возвращается указатель на считанное предложение, которое хранится в переменной *sentence*.

2.3.Функция удаления повторно встречающихся предложений

delete_sentence()

В функцию *int delete_sentence()* подается на вход *char** text*- указатель на массив строк и *int len* – количество предложений в тексте. Объявляются переменные *int sent* – индекс предложения в тексте, *int len_before* – количество предложений в необработанном тексте.

С помощью цикла *while()* будет совершаться проход по каждому предложению и предложения будут сравниваться посимвольно. Если нашлось совпадение, то с помощью функции *memmove()* будет совершаться копирование оставшегося текста на место совпавшего предложения, а потом копирование оставшегося текста на место предложения по которому проверялись все остальные и переменная *len*, отвечающей за количество предложений в тесте, уменьшится на один. Сравнение происходит посимвольно без учета регистра. С помощью функции *tolower()* каждый символ предложений приводится к символу нижнего регистра и символы сравниваются между собой. Индекс, хранящийся в sent увеличится только в том случае, когда за итерацию не встретится ни одного совпавшего предложения.

После завершения цикла будет выведен строка, в которой будет указана разница между количеством предложений необработанного текста и нынешним количеством предложений в тексте после его обработки.

Из функции *delete_sentence()* с помощью функции *return* вернется количество предложений форматированного текста, которое хранится в переменной *len.*

2.4.Функция вывода количества слов «garbage» в каждом предложении без учета регистра

print_garbage()

В функцию *void print_garbage()* подается на вход *char** text*- указатель на массив строк и *int len* – количество предложений в тексте.

С помощью цикла for совершается проход по каждому предложению текста. Объявляется переменная *int count_garbage*, отвечающая за количество подстрок “*garbage*” в данном предложении. Если предложение меньше

подстроки, то сразу выводится строка *“Clean”*, иначе также с помощью цикла for совершается проход по каждому символу в предложении в поисках подстроки *“garbage”* без учета регистра. С помощью функции *tolower()* каждый символ предложения приводится к символу нижнего регистра и сравнивается с одной из букв подстроки. Если подстрока найдена, то переменная *count_garbage* увеличивается на единицу.

После цикла, который совершает проход по символам, выводятся сообщения в зависимости от количества слов *“garbage”*, которое хранится в переменной *count_garbage*.

Из функции ничего не возвращается, потому что функция *print_garbage()* типа *void*.

2.5.Функция замены всех цифр на введенную пользователем строку *number_string()*

В функцию *void number_string()* подается на вход *char** text*- указатель на массив строк и *int len* – количество предложений в тексте. Объявляются переменные *int memory* – блок памяти для переменной *string*, *int count_char* – количество считанных символов, *char* string* – считанная строка.

С помощью цикла *do{}while()* происходит считывание подстроки, которую нужно вставить вместо всех цифр, из стандартного потока ввода.

С помощью цикла *for* совершается проход по каждому предложению в тексте, объявляется переменная *int count_num* – количество цифр в предложении, а с помощью второго цикла *for* совершается проход по каждому символу предложения и с помощью функции *isdigit()* проверяется, является ли символ цифрой. Если условие истинно, то переменная *count_num* увеличивается на единицу.

Если *count_num* не равно нулю, то с помощью функции *strcpy()* создается копия на предложение в переменную *char* sentence*. С помощью двух циклов *for* происходит замена встреченной цифры на введенную подстроку, которая записана в переменную *string*.

```
    if(count_num != 0){
                char* m = malloc(len_sent*sizeof(char));
                char* sentence;
                if(m != NULL){
                    sentence = m;
                }
                else{
                    puts("Ошибка! Недостаточно памяти!");
                }
                strcpy(sentence, text[sent]);
                char*  l  =  realloc(text[sent],((len_sent+count_num  *
(strlen(string)-1))* sizeof(char)));
                if (l != NULL){
                    text[sent] = l;
                }
                else{
                    puts("Ошибка! Недостаточно памяти!");
                }
                int i = 0;
                int n = 0;
                while(i < (len_sent + (strlen(string)-1)*count_num)){
                    if(isdigit(sentence[n])){
                        for(int k = i, j = 0; j <= strlen(string); k++,
j++){
                            text[sent][k] = string[j];

                        }
                        n++;
                        i += strlen(string);
                    }
                    else{
                        text[sent][i++] = sentence[n++];

                    }

                free(sentence);

            }
```

2.6.Функция удаления всех предложений, в которых есть три подряд идущие буквы в верхнем регистре

up()

В функцию *int up()* подается на вход *char** text*- указатель на массив строк и *int len* – количество предложений в тексте. Объявляется переменная *int sentence* – индекс предложения в тексте.

С помощью цикла *while()* совершается проход по каждому предложению в тексте. Объявляется переменная *int flag*, которая отвечает за наличие трех подряд идущих заглавных букв. С помощью цикла *for* совершается проход по

каждому символу предложения. С помощью функции *isupper()* проверяются три рядом стоящие символа. Если условие истинно, то переменная *fla*g принимает значение 1.

Если значение *flag* истинно, то с помощью функции *memmove()* будет совершаться копирование оставшегося текста на место предложения, в котором имеются три подряд идущие буквы в верхнем регистре.

Если значение *flag* будет ложным, то значение переменной sentence будет увеличено на единицу.

Из функции *up()* с помощью функции *return* возвращается количество предложений в тексте после его обработки, которое хранится в переменной *len.*

2.7.Функция сортировки предложений по уменьшению количества слов, начинающихся с гласной

Функция сортировки предложений по уменьшению количества слов, начинающихся с гласной реализуется с помощью функций *vowel()* и *test_vowel().*

vowel()

В функцию *void vowel()* подается на вход *char** *text*- указатель на массив строк и *int len* – количество предложений в тексте. Объявляется массив *int count_vowel[len]* –в данном массиве будет храниться количество слов, начинающихся с гласной, в соответствующем предложении по индексу.

С помощью цикла *for* и функции *int test_vowel*() совершается проход и считается количество слов, начинающихся с гласной и заполняется массив *count_vowel.*

Далее, используя алгоритм сортировки называемый методом «пузырька», производится сравнение значений массива count_vowel и сортировка самого массива *count_vowel* и массива строк *text.*

```
int buff_count;
    char *buff_sent;
    for(int i = len-1; i > 0; i--){
```

```
        for(int j = 0; j < i; j++){
            if(count_vowel[j] < count_vowel[j+1]){
                buff_sent = text[j];
                text[j] = text[j+1];
                text[j+1] = buff_sent;

                buff_count = count_vowel[j];
                count_vowel[j] = count_vowel[j+1];
                count_vowel[j+1] = buff_count;

            }
        }
    }
```

Из функции ничего не возвращается, потому что функция *vowel()* типа *void.*

test_vowel()

В функцию *int test_vowel()* подается на вход *char symbol* – символ, который нужно проверить, является ли он гласной.

С помощью условного оператора *if* и функции *tolower()* совершается проверка, является ли символ гласной без учета регистра.

Если условие истинно, из функции *test_vowel()* с помощью функции *return* возвращается 1, иначе возвращается 0.

2.8.Функция вывода

print_text()

В функцию *void print_text()* подается на вход *char** *text*- указатель на массив строк и *int len* – количество строк в массиве *text*. С помощью цикла *for* совершается проход по каждой строке массива и с помощью функции *printf(),* содержащей спецификатор *%s* для вывода строки, происходит вывод текста на экран.

Из функции ничего не возвращается, потому что функция *print_text()* типа *void.*

2.9.Функция освобождения памяти

free_memory()

В функцию *void free_memory()* подается на вход *char*** *text*- указатель на массив строк *и int len* – количество строк в массиве *tex*t. С помощью цикла *for* совершается проход по каждой строке массива и с помощью функции *free()* происходит освобождение памяти, выделенной на каждую строку.

После цикла с помощью функции *free()* происходит очищение памяти, выделенной под массивы строк.

Из функции ничего не возвращается, потому что функция *free_memory()* типа *void*.

3. ТЕСТИРОВАНИЕ

1) Количество слов “garbage” без учета регистра

№ Теста	Ввод	Вывод
1	Garbage. GaRbAGe. Hello.	Must be washed Must be washed Clean
2	GarBage garbage GARBAGE garbage garbage GaRbAgE. Cat dog. GARBAGE.	It is dump Clean Must be washed
3	Hello. My. Friend.	Clean Clean Clean

2) Замена всех цифр на введенную строку.

№ Теста	Ввод	Вывод
1	2. 3. 4. 5. Hello	Hello. Hello. Hello. Hello.
2	For4. How5. N7. ever	Forever. However. Never.
3	78 loves you. 21 misses you. ma	Mama loves you. Mama misses you.

3) Удаление всех предложений, в которых есть три подряд идущие буквы в верхнем регистре.

№ Теста	Ввод	Вывод
1	DELETE me. Hello, my friend.	Hello, my friend.
2	DelETE me. Hello, my friend.	Hello, my friend.
3	DELete me. Hello, my friend.	

4) Сортировка по уменьшению количества слов начинающихся с гласной буквы.

№ Теста	Ввод	Вывод
1	Apple. Orange orange. Apple APPLE apple.	Apple APPLE apple. Orange orange. Apple.
2	Happy New Year. Tangerine. I wanna see New Year.	I wanna see New Year. Happy New Year. Tangerine.
3	AAAA. Ee EE ee. Ii ii iiii ii ii ii. Oo oo.	Ii ii iiii ii ii ii. Ee EE ee. Oo oo. AAAA.

4.ПОЛЬЗОВАТЕЛЬСКАЯ ИНСТРУКЦИЯ

1. Откройте терминал и запустите программу из папки с помощью команды gcc Yarusova_Tatyana_cw.c && ./a.out.

2. Программа встретит вас словами «Добро пожаловать!»

3. После требуется ввести текст, который вы хотите обработать.

   Текст должен представлять предложения, разделенные точкой. Предложения – набор слов, разделенные пробелом или запятой, слова – набор латинских букв и цифр.

4. Далее необходимо выбрать одну из предоставленных команд.

   На экране отобразится результат.

   Если введенная вами команда не будет соответствовать ни одной из предложенных команд, то программа выведет ошибку вида «Вы ввели некорректное значение. Попробуйте еще раз!»

5. После выполнения команды программа предложит повторно выбрать команду, если вы захотите завершите работу, выберите команду номер 5 и программа завершит свою работу и выведет на экран сообщение «Всего доброго! До свидания!»

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы были изучены особенности работы со строками и символами в языке программирования C, наработаны навыки работы с динамической памятью, ее выделение и освобождение, была освоена работа с указателями.

Была написана программа, считывающая текст из стандартного потока ввода и выполняющая его форматирование исходя из выбранной команды пользователем. Были использованы функции стандартных библиотек и собственные функции. Также в программе разработано диалоговое окно для общения программы с пользователем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Правила оформления пояснительной записки к курсовой работе// se.moevm.info URL: http://se.moevm.info/doku.php/courses:programming:report
2. Теоретические сведение о строках в языке С ://server.179.ru URL: https://server.179.ru/tasks/cpp/total/051.html
3. Библиотеки и функции в языку С //www.cplusplus.com URL: http://www.cplusplus.com/reference/

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

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


Char* read_sentence(char sep){
    char* sentence;
    int count_char = 0;
    int memory_for_char = 50;
    char c;
    char* t = malloc(memory_for_char * sizeof(char));
    if (t != NULL){
        sentence = t;
    }
    else{
        puts("Ошибка! Недостаточно памяти!");
        return NULL;
    }
    do{

        if (count_char == memory_for_char){
           memory_for_char += 10;
            t = realloc(sentence, memory_for_char * sizeof(char));
            if(t != NULL){
               sentence = t;
            }
            else{
                puts("Ошибка! Недостаточно памяти!");
                return NULL;
            }
     }

        c = getchar();
        if ((c != '\t') && (c != ' ')){
            sentence[count_char++] = c;
        }
        else{
            if ((c == ' ') && (count_char != 0)){
                sentence[count_char++] = c;
            }
        }

    }while(sentence[count_char-1] != sep);
    sentence[count_char] = '\0';
    return sentence;
}

int read_text(char*** text, char sep){

    int count_sentence = 0;
    int count_char_in_sentence = 0;
```

```
    int memory = 10;
    int memory_for_char = 50;
    char* sentence;
    char** t = malloc(memory*sizeof(char*));
    if(t != NULL){
        *text = t ;
    }
    else{
        puts("Ошибка! Недостаточно памяти!");
        return 0;
    }

    do{
        sentence = read_sentence(sep);
        (*text)[count_sentence++] = sentence;

        if (count_sentence == memory){
           memory += 10;
            t = realloc(*text, memory * sizeof(char*));
            if (t != NULL){
               *text = t;
           }
            else{
                puts("Ошибка! Недостаточно памяти!");
                return 0;
            }
        }

    }while(getchar() != '\n');
    return count_sentence;
}

int delete_sentence(char** text, int len){
    int sent = 0;
    int len_before = len;
    while(sent < len - 1){
        int old_len = len;
        int len_sent = strlen(text[sent]);
        int sent2 = sent +1;
        while(sent2 < len){
            int len_sent2 = strlen(text[sent2]);
            int count_char = 0;
            if(len_sent == len_sent2){
                for(int ch = 0; ch < len_sent;ch++){
                    if((tolower(text[sent][ch]))!=
(tolower(text[sent2][ch])))
                        break;
                    else
                        count_char++;
                }
            }
            if(count_char == len_sent){
                free(text[sent2]);
                len--;
                memmove(text+sent2, text+sent2 + 1, sizeof(char*) * (len
- sent2 +1));
            }
```

```
            else sent2++;
        }
        if (old_len > len){
            free(text[sent]);
            len--;
            memmove(text+sent, text+sent + 1, sizeof(char*) * (len - sent
+ 1));
        }
        else sent++;
    }
    printf("\nВ тексте было %d повторно встречающихся предложений. Они
были удалены!\n", len_before - len);
    return len;
}

void print_garbage(char** text, int len){
    for(int i = 0; i < len; i++){
        int count_garbage = 0;
        int len_sentence = strlen(text[i]);
        if(len_sentence >= strlen("garbage")){
            for(int j = 0; j <= len_sentence - strlen("garbage"); j++){
                if ((tolower(text[i][j])== 'g')&&(tolower(text[i][j+1])==
'a')&&(tolower(text[i][j+2])== 'r')&&(tolower(text[i][j+3])== 'b')
                &&(tolower(text[i][j+4])== 'a')&&(tolower(text[i][j+5])==
'g')&&(tolower(text[i][j+6])== 'e')){
                    count_garbage++;
                }
            }
            if(count_garbage > 0){
                if(count_garbage < 6)
                    puts("Must be washed");
                else
                    puts("It is a dump");

            }
            else
                puts("Clean");
        }
        else
            puts("Clean");
    }

}

void number_string(char** text, int len){
    int memory = 10;
    int count_char = 0;
    char* string;
    char* t = malloc(memory * sizeof(char));
    if(t != NULL){
        string = t;
    }
    else{
        puts("Ошибка! Недостаточно памяти!");
    }
    char c;
    c = getchar();
```

```
    do{
        if (memory == count_char){
            memory += 5;
            t = realloc(string, memory * sizeof(char));
            if(t != NULL){
                string = t;
            }
            else{
                puts("Ошибка! Недостаточно памяти!");
            }
        }
        c = getchar();
        if(c != '\n'){
            string[count_char++] = c;
        }
    }while(c != '\n');


    for(int sent = 0; sent < len; sent++){
        int count_num = 0;
        int len_sent = strlen(text[sent]);

        for (int i = 0; i < len_sent; i++){
            if(isdigit(text[sent][i]))
                count_num++;
        }

        if(count_num != 0){
            char* m = malloc(len_sent*sizeof(char));
            char* sentence;
            if(m != NULL){
                sentence = m;
            }
            else{
                puts("Ошибка! Недостаточно памяти!");
            }
            strcpy(sentence, text[sent]);
            char* l = realloc(text[sent],((len_sent+count_num *
(strlen(string)-1))* sizeof(char)));
            if (l != NULL){
                text[sent] = l;
            }
            else{
                puts("Ошибка! Недостаточно памяти!");
            }
            int i = 0;
            int n = 0;
            while(i < (len_sent + (strlen(string)-1)*count_num)){
                if(isdigit(sentence[n])){
                    for(int k = i, j = 0; j <= strlen(string); k++, j++){
                        text[sent][k] = string[j];

                    }
                    n++;
                    i += strlen(string);
                }
                else{
```

```
                    text[sent][i++] = sentence[n++];

                }
            }
            free(sentence);

        }


    }
    free(string);

}

int up(char** text, int len){
    int sentence = 0;
    while(sentence < len){
        int flag = 0;
        int len_sentence = strlen(text[sentence]);
        for(int j = 0; j < len_sentence - 2; j++){
            if((isupper(text[sentence][j]))&&
(isupper(text[sentence][j+1])) && (isupper(text[sentence][j+2]))){
                flag = 1;
            }
        }
        if(flag){
                free(text[sentence]);
                len--;
                memmove(text+sentence, text+sentence+1, sizeof(char*) *
(len-sentence+1));
        }
        else{
            sentence++;
        }
    }
    return len;
}
int test_vowel(char symbol){
    if((tolower(symbol)=='a') || (tolower(symbol)=='i')
||(tolower(symbol)=='o') || (tolower(symbol)=='e')
||(tolower(symbol)=='y') || (tolower(symbol)=='u'))
        return 1;
    else
        return 0;
}

void vowel(char** text, int len){
    int count_vowel[len];
    for(int sentence = 0; sentence < len; sentence++){
        int count = 0;
        int len_sentence = strlen(text[sentence]) - 1;
        for(int ch = 0; ch < len_sentence - 1; ch++){
            if(((text[sentence][ch]==' ') &&
(test_vowel(text[sentence][ch+1]))) || (ch == 0) &&
(test_vowel(text[sentence][ch]))){
                count++;
            }
```

```
        }
        count_vowel[sentence] = count;

    }

    int buff_count;
    char* buff_sent;
    for(int i = len-1; i > 0; i--){
        for(int j = 0; j < i; j++){
            if(count_vowel[j] < count_vowel[j+1]){
                buff_sent = text[j];
                text[j] = text[j+1];
                text[j+1] = buff_sent;

                buff_count = count_vowel[j];
                count_vowel[j] = count_vowel[j+1];
                count_vowel[j+1] = buff_count;

            }
        }
    }
}


void print_text(char **text, int sentence){

    for(int i = 0; i < sentence; i++){
        printf("%s ",text[i]);
    }
    printf("\n\n");
}

void free_memory(char ***text, int sentence){

    for(int i = 0; i < sentence; i++){
        free((*text)[i]);
    }

    free((*text));

}

int main(){
    char** text;
    char sep = '.';
    puts("Добро пожаловать!");
    puts("Введите текст, который хотите обработать.");
    int len = read_text(&text, sep);
    len = delete_sentence(text, len);
    print_text(text,len);
    while (1){
        puts("Как вы хотите обработать текст?");
        puts("1)Вывести количество слов ""garbage"" в каждом
предложении.");
        puts("2)Заменить все цифры на введеную вами строку.");
        puts("3)Удалить предложения, в которых встречаются три подряд
идущие буквы в верхнем регистре.");
```

```
        puts("4)Отсортировать по уменьшению количества слов начинающихся
с гласной буквы.");
        puts("5)Завершить программу.");
        printf("Введите номер команды: ");
        int value;
        scanf("%d", &value);
        switch (value){
            case 1:
                puts("\nРезультат:");
                print_garbage(text,len);
                printf("\n");
                break;

            case 2:
                puts("\nВведите строку");
                number_string(text, len);
                puts("\nРузультат:");
                print_text(text,len);
                break;

            case 3:
                len = up(text, len);
                puts("\nРузультат:");
                print_text(text,len);
                break;

            case 4:
                puts("\nРузультат:");
                vowel(text,len);
                print_text(text,len);
                break;

            case 5:
                puts("\nВсего доброго! До свидания!");
                free_memory(&text,len);
                return 0;

            default:
                puts("\nВы ввели некорректное значение. Попробуйте еще
раз!\n");

        }
    }
}
```

ПРИЛОЖЕНИЕ Б
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Программа приняла на вход текст, удалила повторно встречающиеся предложения без учета регистра и выполнила первую команду. Рис.1

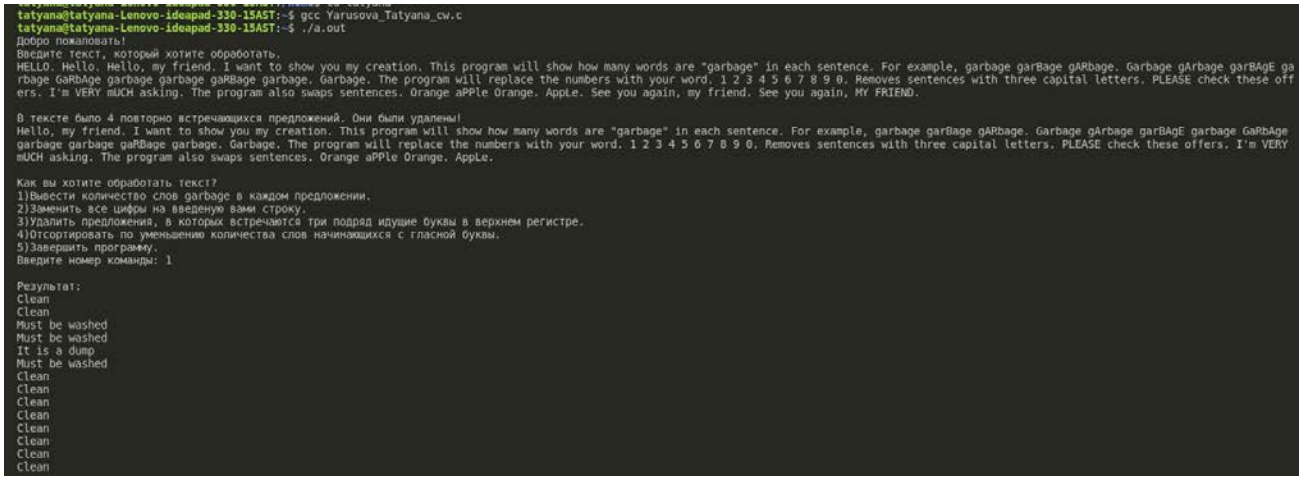

Рис.1

Программа выполнила вторую команду. Рис.2.

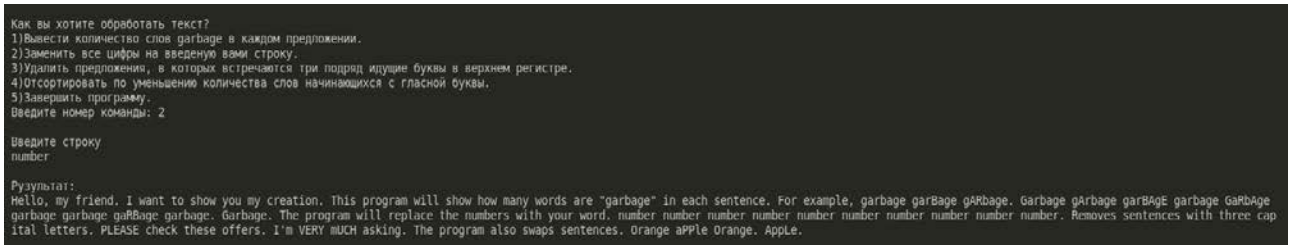

Рис.2.

Программа выполнила 3 команду. Рис.3.

Рис.3.

Программа выполнила 4 команду. Рис.4.

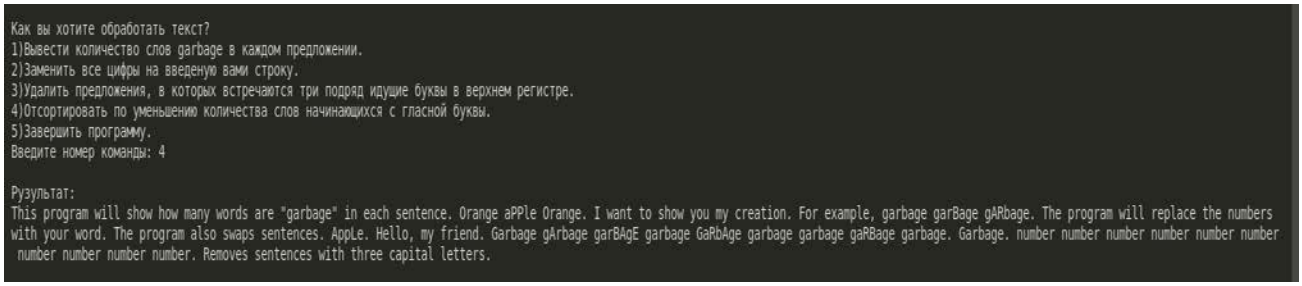

Рис.4.

Программа выполняет 5 команду и завершает свою работу. Рис.5.

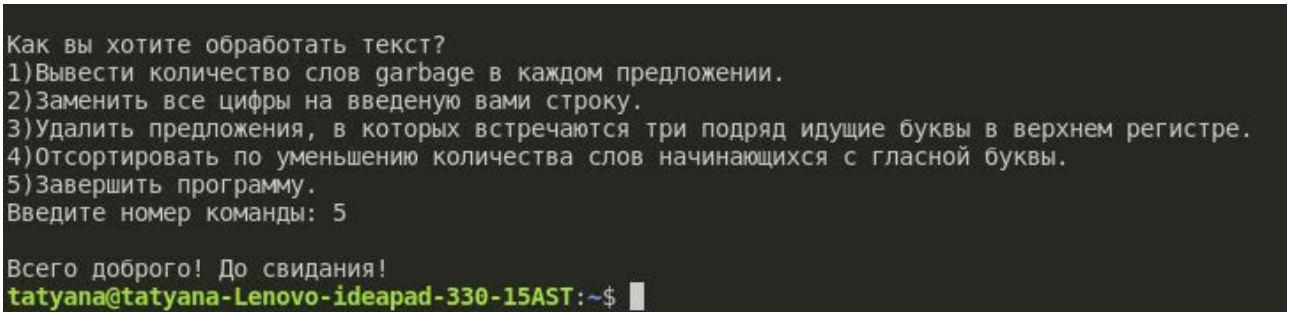

Рис.5.

ПРИЛОЖЕНИЕ В

ПРИМЕРЫ ОБРАБОТКИ ОШИБОК

```
Как вы хотите обработать текст?
1)Вывести количество слов garbage в каждом предложении.
2)Заменить все цифры на введеную вами строку.
3)Удалить предложения, в которых встречаются три подряд идущие буквы в верхнем регистре.
4)Отсортировать по уменьшению количества слов начинающихся с гласной буквы.
5)Завершить программу.
Введите номер команды: 78

Вы ввели некорректное значение. Попробуйте еще раз!

Как вы хотите обработать текст?
1)Вывести количество слов garbage в каждом предложении.
2)Заменить все цифры на введеную вами строку.
3)Удалить предложения, в которых встречаются три подряд идущие буквы в верхнем регистре.
4)Отсортировать по уменьшению количества слов начинающихся с гласной буквы.
5)Завершить программу.
Введите номер команды: 5

Всего доброго! До свидания!
tatyana@tatyana-Lenovo-ideapad-330-15AST:~$
```