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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Работа со строками в языке Си

Студент гр. 1304 _____ Маркуш А.Е.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2021

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Маркуш А.Е.

Группа 1304

Тема работы: Работа со строками в языке Си

Исходные данные:
Текст (текст представляет собой строку из предложений, разделенных точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских букв и цифр.)

Содержание пояснительной записки:
Содержание
Введение.
Описание кода программы.
Заключение.
Список используемых источников.

Предполагаемый объем пояснительной записки:
Не менее 10 страниц.

Дата выдачи задания: 15.10.2021

Дата сдачи реферата: 22.12.2021

Дата защиты реферата: 25.12.2021

Студент _____ Маркуш А.Е.

Преподаватель _____ Чайка К.В.

АННОТАЦИЯ

Курсовая работа представляет из себя программу для работы с текстом. 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 сохраняет этот текст в динамический массив строк. Далее она находит и удаляет все повторно встречающиеся предложения (сравнивает их посимвольно, но без учета регистра). Далее, программа запрашивает у пользователя одно из доступных действий. Ввод и остальные операции выполняются с помощью написанных функций. Исходный код и примеры работы программы представлены в приложениях.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является написание программы для считывания и редактирования строк.

Требуется реализовать следующие функции:

1. Функция, которая печатает каждое слово и количество его повторений в тексте.
2. Функция, преобразовывающая каждое предложение так, что символы в каждом слове шли в обратном порядке.
3. Функция, удаляющая предложения, в которых встречается запятая.
4. Функция, сортирующая предложения по уменьшению значения кода 5 символа предложения. Если 5 символ является разделителем между словами, то брать следующий символ. Если символов в предложении меньше 5, то для этого предложения значение равняется -1.

Для демонстрации работы функций, требуется написать диалоговую программу, которая должна запрашивать одно из доступных действий, включая выход из программы.

1. ОПИСАНИЕ КОДА ПРОГРАММЫ

1.1. Функция ввода текста

```
char** get_text(int *text_size){
    int s = 0;
    int c = 0;
    char x;
    int s_length = MEM_STEP;
    int t_length = SIZE;
    char** text = malloc(t_length*sizeof(char*));
    if(text == NULL){
        puts("Memory error!");
        return NULL;
    }
    text[s] = malloc(s_length*sizeof(char));
    if(text[s] == NULL){
        puts("Memory error!");
        return NULL;
    }
    while((x = getchar()) != '\n'){
        if(s == t_length){
            t_length += SIZE;
            text = realloc(text, sizeof(char*)*t_length);
            if(text == NULL){
                puts("Memory error!");
                return NULL;
            }
            text[s] = malloc(s_length*sizeof(char));
            if(text[s] == NULL){
                puts("Memory error!");
                return NULL;
            }
        }
        if(c + 1 == s_length){
            s_length += MEM_STEP;
            text[s] = realloc(text[s], sizeof(char)*s_length);
            if(text[s] == NULL){
                puts("Memory error!");
                return NULL;
            }
        }
        text[s][c] = x;
        c++;
        if(x == '.'){
            text[s][c] = '\0';
            s++;
            s_length = MEM_STEP;
            c = 0;
        }
    }
    *text_size = s;
    return text;
}
```

Функция посимвольно считывает текст. Память под предложения выделяется по мере необходимости, а вод символы в предложениях блоками. Считывание текста прекращается при вводе символа переноса строки *‘\n’*.

1.2. Функция удаления повторяющихся предложений

```
int del_repeats(char*** text, int text_size){
    int i = 0;
    while(i < text_size){
        int count = 0;
        int j = i+1;
        while(j < text_size){
            if(!strcasecmp((*text)[i], (*text)[j])){
                count = 1;
                free((*text)[j]);
                for(int k = j; k < text_size - 1; k++){
                    (*text)[k] = (*text)[k + 1];
                }
                text_size--;
            }
            else{
                j++;
            }
        }
        if(count != 0){
            free((*text)[i]);
            for(int j = i; j < text_size -1; j++){
                (*text)[j] = (*text)[j + 1];
            }
            text_size--;
        }
        else{
            i++;
        }
    }
    return text_size;
}
```

Функция с помощью вложенных циклов *for()* проходит по тексту в поисках повторных предложений, сначала удаляются повторы, а затем удаляет предложение по которому производилось сравнение. Функция возвращает размер отредактированного текста.

1.3. Функция поиска и удаления предложений с запятыми.

```
int del_sentences(char*** text, int text_size){
    int count;
    int i = 0;
    while(i < text_size){
        count = 0;
        for(int j = 0; j < strlen((*text)[i]); j++){
            if((*text)[i][j] == ','){
                count++;
            }
```

```
            }
            if(count >= 1){
                free((*text)[i]);
                for(int j = i; j < text_size - 1; j++){
                    (*text)[j] = (*text)[j + 1];
                }
                text_size--;
            }
            else{
                i++;
            }
        }
        return text_size;
    }
```

Функция производит поиск символа *‘,’* в каждом предложении. При наличии данного символа предложение удаляется из теста путём освобождения выделенной под него памяти с помощью функции *free()*. После этого происходит сдвиг остальных предложений. Поиск продолжается с предложения с тем же индексом, на котором он остановился. Функция возвращает размер отредактированного текста.

1.4. Функция сравнения кодов пятых символов двух предложений

```
int sort_comparison(const void *s1,const void *s2){
    char *str1 = *(char**)s1;
    char *str2 = *(char**)s2;
    int ord1;
    int ord2;
    if(strlen(str1) < 5){
        ord1 = -1;
    }
    else{
        if(str1[COMP_CHAR] == ',' || str1[COMP_CHAR] == ' '){
            ord1 = str1[COMP_CHAR+1];
        }
        else{
            ord1 = str1[COMP_CHAR];
        }
    }
    if(strlen(str2) < 5){
        ord2 = -1;
    }
    else{
        if(str2[COMP_CHAR] == ',' || str2[COMP_CHAR] == ' '){
            ord2 = str2[COMP_CHAR+1];
        }
        else{
            ord2 = str2[COMP_CHAR];
        }
    }
    if(ord1 > ord2){
        return -1;
    }
```

```
    else if(ord1 < ord2){
        return 1;
    }
    return 0;
}
```

Функция ищет пятый символ в каждом предложении и записывает их коды. Если предложение меньше чем пять символов, его значение равняется -1. Если пятый символ является разделителем, то берётся код следующего символа. Далее происходит сравнение. Если код больше у первого предложения функция возвращает -1, если у второго, то 1. При равенстве функция возвращает 0.

1.5. Функция сортировки предложений по коду пятого символа

```
void sorting(char**text, int text_size){
    qsort(text, text_size, sizeof(char*), sort_comparison);
}
```

Функция производит сортировку предложений по уменьшению кода пятого символа с помощью функции сортировки *qsort()*. В *qsort()* подаётся функция *sort_comparison()*, описанная ранее. Функция ничего не возвращает, т.к. *qsort()* напрямую изменяет исходный текст.

1.6. Функция изменения порядка символов в словах на обратный

```
void reverse_words(char **text, int text_size){
    char *first_letter;
    char *last_letter;
    for(int i = 0; i < text_size; i++){
        char c;
        int x;
        int begin = 0;
        first_letter = &text[i][0];
        for(int j = 0; j < strlen(text[i]); j++){
            if((text[i][j] == '.') || (text[i][j] == ',') ||
(text[i][j] == ' ')){
                last_letter = &text[i][j-1];
                x = 0;
                while(((j - begin)/2) != x){
                    c = *(first_letter + x);
                    *(first_letter + x) = *(last_letter - x);
                    *(last_letter - x) = c;
                    x++;
                }
                first_letter = &text[i][j+1];
                begin = j+1;
            }
        }
    }
}
```

Функция проходит по каждому предложению в поисках разделителей, Словом считается набор символов между ними. Далее в переменные

передаются адреса начала и конца слова и происходит замена символов находящихся на равном расстоянии до серины слова друг на друга. Функция ничего не возвращает, т.к. напрямую редактирует исходный текст.

1.7. Функция подсчёта и вывода всех слов в тексте

```
void word_count(char** text, int text_size){
    char** words;
    int words_len = 0;
    char *first_letter;
    words = malloc(WORDS_NUMBER*sizeof(char*));
    if(words == NULL){
        puts("Memory error!");
    }
    int count[WORDS_NUMBER];
    for(int i = 0; i < text_size; i++){
        int x;
        int begin = 0;
        first_letter = &text[i][0];
        for(int j = 0; j < strlen(text[i]); j++){
            if((text[i][j] == '.') || (text[i][j] == ',') ||
(text[i][j] == ' ')){
                if(j == 0){
                    first_letter = &text[i][j+1];
                    begin = j+1;
                    continue;
                }
                char* word = malloc((j-begin+1)*sizeof(char));
                if(word == NULL){
                    puts("Memory error!");
                }
                x = 0;
                while(((j - begin)) != x){
                    word[x] = *(first_letter + x);
                    x++;
                    if(x == j - begin){
                        word[x] = '\0';
                    }
                }
                if(words_len == 0){
                    words[0] = word;
                    count[0] = 0;
                    words_len++;
                }
                int z = words_len;
                int n = 0;
                for(int k = 0; k < z; k++){
                    if(!strcmp(word, words[k])){
                        count[k]++;
                        n = 1;
                    }
                }
                if(n == 0){
                    words[words_len] = word;
                    count[words_len] = 1;
                    words_len++;
```

```
                    }
                    first_letter = &text[i][j+1];
                    begin = j+1;
                }
            }
        }
        for(int i = 0; i < words_len; i++){
            printf("%s : %d\n", words[i], count[i]);
        }
        memory_free(words, words_len);
    }
```

Функция ищет слова в предложения как в функции *reverse_words()* и записывает их в двумерный динамический массив символов, а также количество их повторений в статический массив целых чисел. Слово, которое встречается повторно не записывается. В данном случае увеличивается значение в массиве чисел под индексом, соответствующим индексу слова, копия которого обнаружена. Функция также ничего не возвращает.

1.8. Функция освобождения памяти

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

Функция освобождает память, выделенную под текст с помощью цикла *for* и функции *free()*.

1.9. Функция вывода текста

```
void text_print(char** text, int text_size){
    for(int i = 0; i < text_size; i++){
        printf("%s", text[i]);
    }
    puts("");
}
```

Функция выводит текст на экран с помощью цикла *for*.

1.10. Главная функция

```
int main(){
    setlocale(LC_ALL, "Russian");
    int text_size = 0;
    puts("Введите пожалуйста текст.");
    char **text = get_text(&text_size);
    text_size = del_repeats(&text, text_size);
    puts("");
    puts("Доступные действия:");
    puts("1)Распечатать каждое слово и количество его повторений в
тексте.");
```

```
        puts("2)Преобразовать каждое предложение так, что символы в
каждом слове шли в обратном порядке.");
        puts("3)Удалить предложения в котором встречается запятая.");
        puts("4)Отсортировать предложения по уменьшению значения кода 5
символа предложения.");
        puts("5)Выйти из программы.\n");
        int end = 0;
        while(end == 0){
            printf("Выберите действие: ");
            int move;
            scanf("%d", &move);
            puts("");
            switch(move){
                case 1:
                    word_count(text, text_size);
                    puts("\n----------------------------------------
----------------------------");
                    break;
                case 2:
                    reverse_words(text, text_size);
                    text_print(text, text_size);
                    puts("\n----------------------------------------
----------------------------");
                    break;
                case 3:
                    text_size = del_sentences(&text, text_size);
                    text_print(text, text_size);
                    puts("\n----------------------------------------
----------------------------");
                    break;
                case 4:
                    sorting(text, text_size);
                    text_print(text, text_size);
                    puts("\n----------------------------------------
----------------------------");
                    break;
                case 5:
                    puts("Работа программы завершена.");
                    end = 1;
                    break;
                default:
                    puts("Данной команды не существует!");
                    end = 1;
                    break;
            }
        }
        memory_free(text, text_size);

        return 0;
    }
```

Функция выводит на экран подсказки для пользователя, считывает выбранные им действие и вызывает нужные функции.

ЗАКЛЮЧЕНИЕ

В ходе данной курсовой работы были изучены управляющие конструкции языка Си, работа с указателями и массивами, работа с символами и строками, а так же разработан и реализован программный код. Так же были проведены тесты программы представленные в приложении Б.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Язык программирования СИ / Керниган Б., Ритчи Д. СПб.: Издательство “Невский диалект”, 2001. 352 с.

2. Основы программирования на языках С и С++ [Электронный ресурс] URL: http://cplusplus.com.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>
#define SIZE 1
#define MEM_STEP 30
#define COMP_CHAR 4
#define WORDS_NUMBER 3000


char** get_text(int *text_size);
int del_repeats(char*** text, int text_size);
int del_sentences(char*** text, int text_size);
int sort_comparison(const void *s1,const void *s2);
void sorting(char**text, int text_size);
void reverse_words(char **text, int text_size);
void word_count(char** text, int text_size);
void text_print(char** text, int text_size);
void memory_free(char** text,int text_size);


char** get_text(int *text_size){
    int s = 0;
    int c = 0;
    char x;
    int s_length = MEM_STEP;
    int t_length = SIZE;
    char** text = (char**)malloc(t_length*sizeof(char*));
    if(text == NULL){
        puts("Memory error!");
        return NULL;
    }
    text[s] = malloc(s_length*sizeof(char));
    if(text[s] == NULL){
        puts("Memory error!");
        return NULL;
    }
    while((x = getchar()) != '\n'){
        if(s == t_length){
            t_length += SIZE;
            text = realloc(text, sizeof(char*)*t_length);
            if(text == NULL){
                puts("Memory error!");
                return NULL;
            }
            text[s] = malloc(s_length*sizeof(char));
            if(text[s] == NULL){
                puts("Memory error!");
                return NULL;
            }
        }
        if(c + 1 == s_length){
            s_length += MEM_STEP;
            text[s] = realloc(text[s], sizeof(char)*s_length);
```

```
            if(text[s] == NULL){
                puts("Memory error!");
                return NULL;
            }
        }
        text[s][c] = x;
        c++;
        if(x == '.'){
            text[s][c] = '\0';
            s++;
            s_length = MEM_STEP;
            c = 0;
        }
    }
    *text_size = s;
    return text;
}


int del_repeats(char*** text, int text_size){
    int i = 0;
    while(i < text_size){
        int count = 0;
        int j = i+1;
        while(j < text_size){
            if(!strcasecmp((*text)[i], (*text)[j])){
                count = 1;
                free((*text)[j]);
                for(int k = j; k < text_size - 1; k++){
                    (*text)[k] = (*text)[k + 1];
                }
                text_size--;
            }
            else{
                j++;
            }
        }
        if(count != 0){
            free((*text)[i]);
            for(int j = i; j < text_size -1; j++){
                (*text)[j] = (*text)[j + 1];
            }
            text_size--;
        }
        else{
            i++;
        }
    }
    return text_size;
}


int del_sentences(char*** text, int text_size){
    int count;
    int i = 0;
    while(i < text_size){
        count = 0;
```

```
        for(int j = 0; j < strlen((*text)[i]); j++){
            if((*text)[i][j] == ','){
                count++;
            }
        }
        if(count >= 1){
            free((*text)[i]);
            for(int j = i; j < text_size - 1; j++){
                (*text)[j] = (*text)[j + 1];
            }
            text_size--;
        }
        else{
            i++;
        }
    }
    return text_size;
}


int sort_comparison(const void *s1,const void *s2){
    char *str1 = *(char**)s1;
    char *str2 = *(char**)s2;
    int ord1;
    int ord2;
    if(strlen(str1) < 5){
        ord1 = -1;
    }
    else{
        if(str1[COMP_CHAR] == ',' || str1[COMP_CHAR] == ' '){
            ord1 = str1[COMP_CHAR+1];
        }
        else{
            ord1 = str1[COMP_CHAR];
        }
    }
    if(strlen(str2) < 5){
        ord2 = -1;
    }
    else{
        if(str2[COMP_CHAR] == ',' || str2[COMP_CHAR] == ' '){
            ord2 = str2[COMP_CHAR+1];
        }
        else{
            ord2 = str2[COMP_CHAR];
        }
    }
    if(ord1 > ord2){
        return -1;
    }
    else if(ord1 < ord2){
        return 1;
    }
    return 0;
}
```

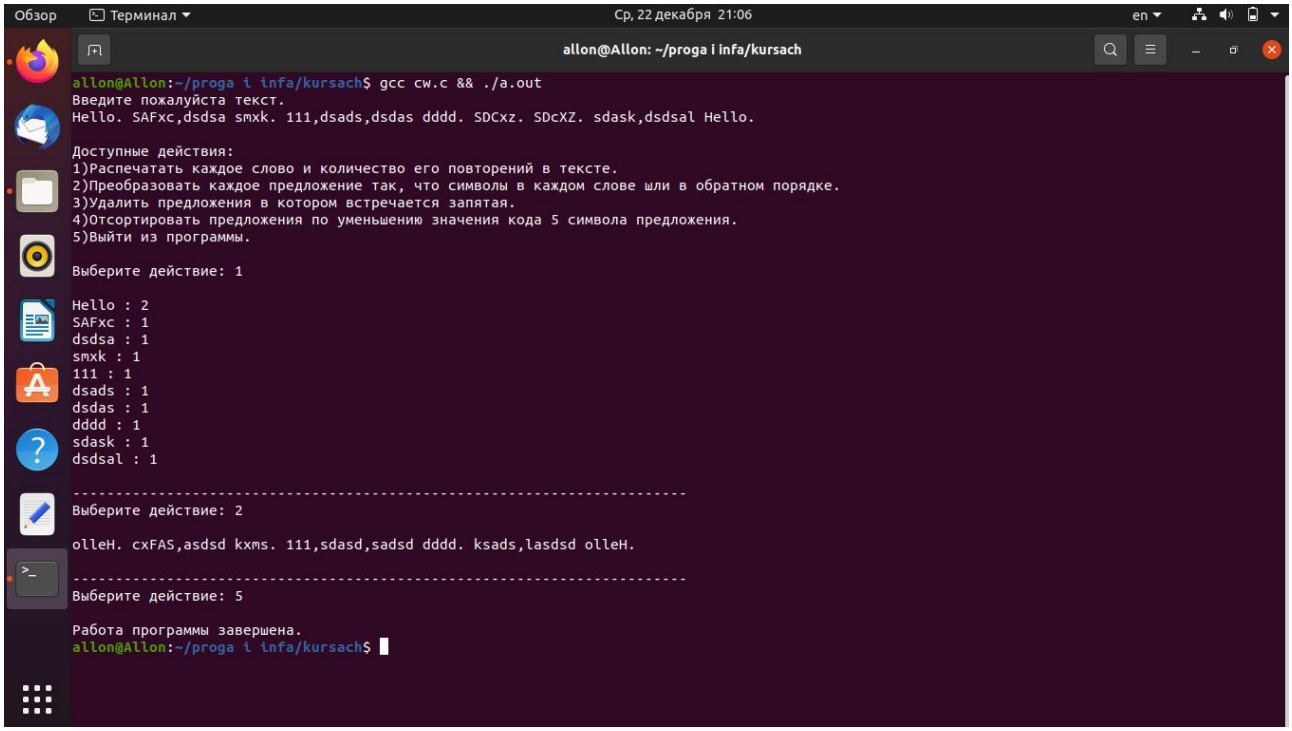
```
void sorting(char**text, int text_size){
    qsort(text, text_size, sizeof(char*), sort_comparison);
}


void reverse_words(char **text, int text_size){
    char *first_letter;
    char *last_letter;
    for(int i = 0; i < text_size; i++){
        char c;
        int x;
        int begin = 0;
        first_letter = &text[i][0];
        for(int j = 0; j < strlen(text[i]); j++){
            if((text[i][j] == '.') || (text[i][j] == ',') || (text[i][j]
== ' ')){
                last_letter = &text[i][j-1];
                x = 0;
                while(((j - begin)/2) != x){
                    c = *(first_letter + x);
                    *(first_letter + x) = *(last_letter - x);
                    *(last_letter - x) = c;
                    x++;
                }
                first_letter = &text[i][j+1];
                begin = j+1;
            }
        }
    }
}


void word_count(char** text, int text_size){
    char** words;
    int words_len = 0;
    char *first_letter;
    words = malloc(WORDS_NUMBER*sizeof(char*));
    if(words == NULL){
        puts("Memory error!");
    }
    int count[WORDS_NUMBER];
    for(int i = 0; i < text_size; i++){
        int x;
        int begin = 0;
        first_letter = &text[i][0];
        for(int j = 0; j < strlen(text[i]); j++){
            if((text[i][j] == '.') || (text[i][j] == ',') || (text[i][j]
== ' ')){
                if(j == 0){
                    first_letter = &text[i][j+1];
                    begin = j+1;
                    continue;
                }
                char* word = malloc((j-begin+1)*sizeof(char));
                if(word == NULL){
                    puts("Memory error!");
                }
```

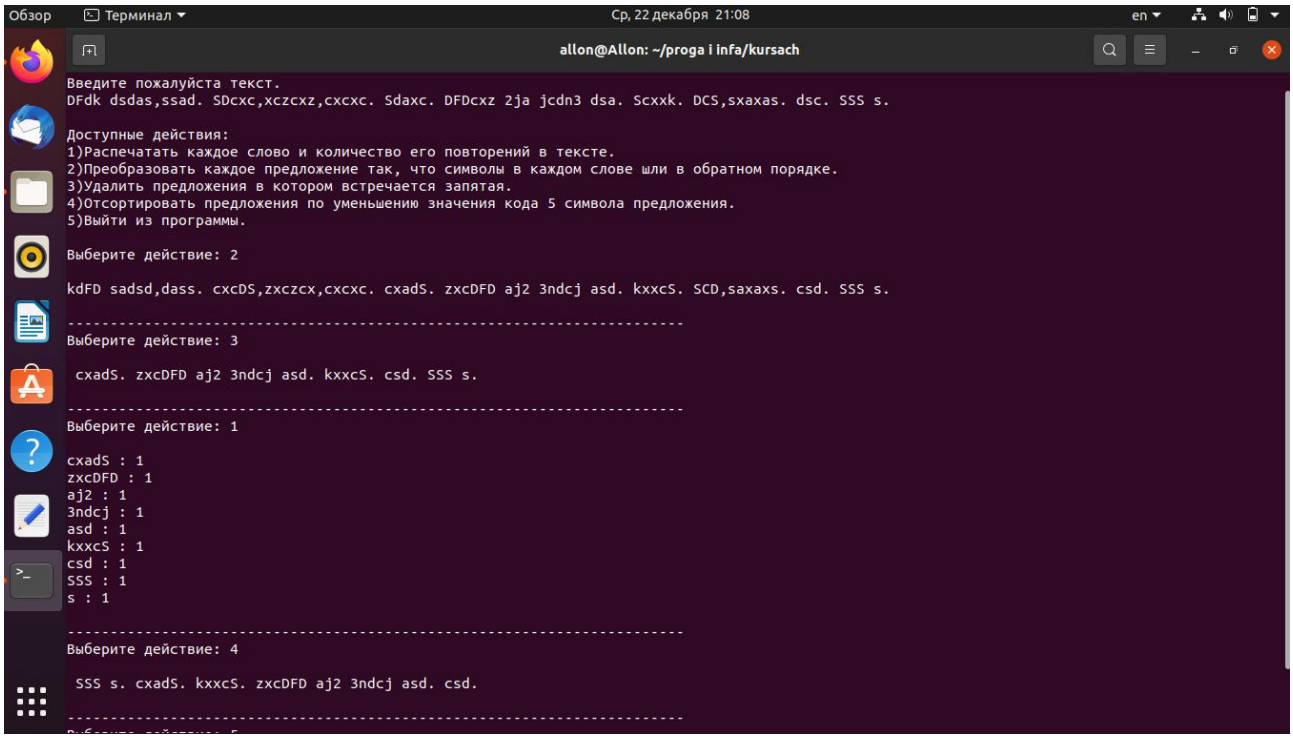
```
                x = 0;
                while(((j - begin)) != x){
                    word[x] = *(first_letter + x);
                    x++;
                    if(x == j - begin){
                        word[x] = '\0';
                    }
                }
                if(words_len == 0){
                    words[0] = word;
                    count[0] = 0;
                    words_len++;
                }
                int z = words_len;
                int n = 0;
                for(int k = 0; k < z; k++){
                    if(!strcmp(word, words[k])){
                        count[k]++;
                        n = 1;
                    }
                }
                if(n == 0){
                    words[words_len] = word;
                    count[words_len] = 1;
                    words_len++;
                }
                first_letter = &text[i][j+1];
                begin = j+1;
            }
        }
    }
    for(int i = 0; i < words_len; i++){
        printf("%s : %d\n", words[i], count[i]);
    }
    memory_free(words, words_len);
}


void memory_free(char** text, int text_size){
    for(int i = 0; i < text_size; i++){
        free(text[i]);
    }
    free(text);
}


void text_print(char** text, int text_size){
    for(int i = 0; i < text_size; i++){
        printf("%s", text[i]);
    }
    puts("");
}


int main(){
    setlocale(LC_ALL, "Russian");
    int text_size = 0;
```

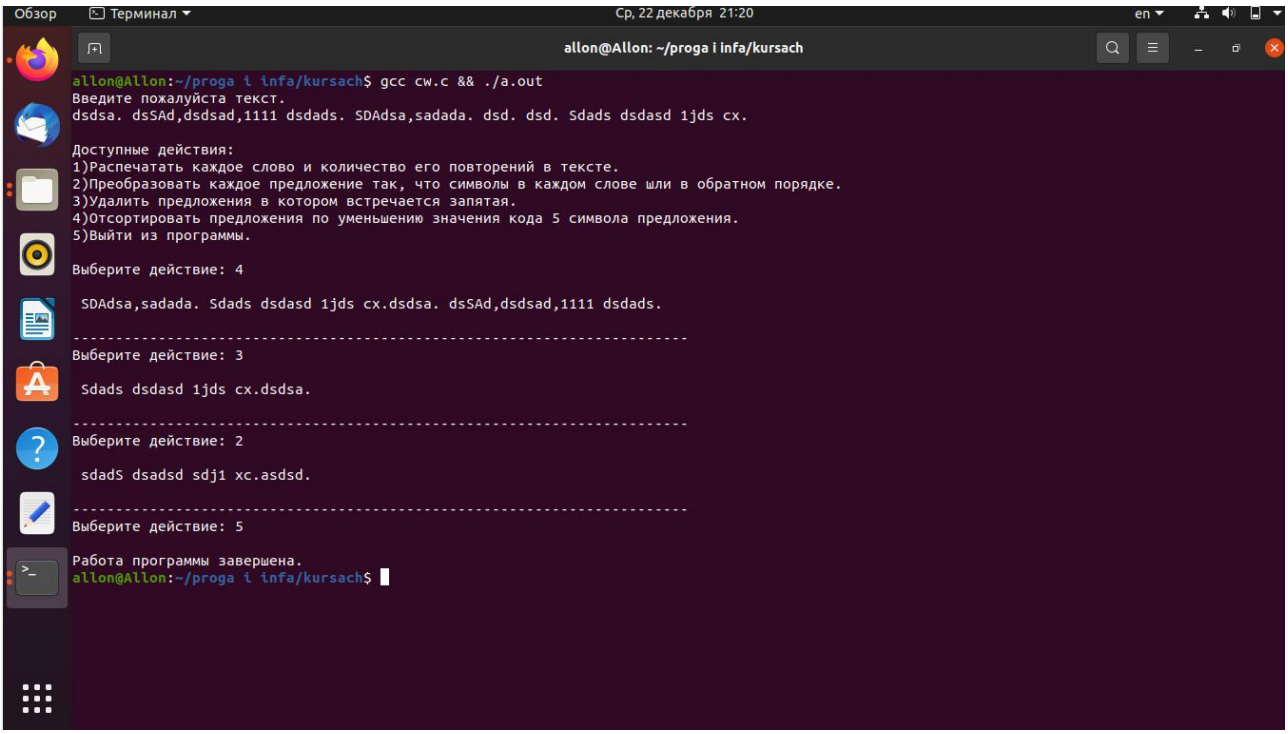
```
    puts("Введите пожалуйста текст.");
    char **text = get_text(&text_size);
    text_size = del_repeats(&text, text_size);
    puts("");
    puts("Доступные действия:");
    puts("1)Распечатать каждое слово и количество его повторений в
тексте.");
    puts("2)Преобразовать каждое предложение так, что символы в каждом
слове шли в обратном порядке.");
    puts("3)Удалить предложения в котором встречается запятая.");
    puts("4)Отсортировать предложения по уменьшению значения кода 5
символа предложения.");
    puts("5)Выйти из программы.\n");
    int end = 0;
    while(end == 0){
        printf("Выберите действие: ");
        int move;
        scanf("%d", &move);
        puts("");
        switch(move){
            case 1:
                word_count(text, text_size);
                puts("\n-----------------------------------------------------
----------------------");
                break;
            case 2:
                reverse_words(text, text_size);
                text_print(text, text_size);
                puts("\n-----------------------------------------------------
----------------------");
                break;
            case 3:
                text_size = del_sentences(&text, text_size);
                text_print(text, text_size);
                puts("\n-----------------------------------------------------
----------------------");
                break;
            case 4:
                sorting(text, text_size);
                text_print(text, text_size);
                puts("\n-----------------------------------------------------
----------------------");
                break;
            case 5:
                puts("Работа программы завершена.");
                end = 1;
                break;
            default:
                puts("Данной команды не существует!");
                end = 1;
                break;
        }
    }
    memory_free(text, text_size);

    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ






Обзор
Терминал
Ср, 22 декабря 21:20
en
allon@Allon: ~/proga i infa/kursach
allon@Allon:~/proga i infa/kursach$ gcc cw.c && ./a.out
Введите пожалуйста текст.
dsdsa. dsSAd,dsdsad,1111 dsdads. SDAdsa,sadada. dsd. dsd. Sdads dsdasd 1jds cx.
Доступные действия:
1)Распечатать каждое слово и количество его повторений в тексте.
2)Преобразовать каждое предложение так, что символы в каждом слове шли в обратном порядке.
3)Удалить предложения в котором встречается запятая.
4)Отсортировать предложения по уменьшению значения кода 5 символа предложения.
5)Выйти из программы.
Выберите действие: 4
SDAdsa,sadada. Sdads dsdasd 1jds cx.dsdsa. dsSAd,dsdsad,1111 dsdads.
Выберите действие: 3
Sdads dsdasd 1jds cx.dsdsa.
Выберите действие: 2
sdadS dsadsd sdj1 xc.asdsd.
Выберите действие: 5
Работа программы завершена.
allon@Allon:~/proga i infa/kursach$