МИНОБРНАУКИ РОССИИ

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

Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Программирование на языке С

Студент гр. 1384 _____ Усачева Д.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студентка Усачева Д.В.

Группа 1384

Тема работы: Программирование на языке С

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.

Содержание пояснительной записки:

“Аннотация”, “Содержание”, “Введение”, “Описание структур”, “Ввод”, “Удаление повторно встречающихся предложений”, “Функции”, “Основная программа”, “Примеры работы программы”, “Заключение”

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 25.10.2021

Дата сдачи реферата: 26.12.2021

Дата защиты реферата: 26.12.2021

Студент гр. 1384	_____	Усачева Д.В.
Преподаватель	_____	Жангиров Т.Р.

АННОТАЦИЯ

Программа должна сохранить введенный 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).

ВВЕДЕНИЕ

В рамках курсовой работы нужно реализовать программу, которая считывает текст, длина которого заранее неизвестна, удалить все повторно встречающиеся предложения (сравнивая их посимвольно, без учета регистра) и по запросу пользователя выполнить одно из следующих действий:

1) Для каждого предложения посчитать количество слов “garbage” в нем (без учета регистра). В зависимости от количества вывести следующие строки: 0 - “Clean”, [1 5] - “Must be washed”, >5 - “It is a dump”.
2) Заменить все цифры в предложениях на введенную строку.
3) Удалить все предложения в которых есть три подряд идущие буквы в верхнем регистре.
4) Отсортировать по уменьшению количества слов начинающихся с гласной буквы.

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.

1. ОПИСАНИЕ СТРУКТУР

1.1. Word

Структура Word содержит в себе следующие поля:

- *char *word* – массив символов, представляющий это слово, который создается и изменяется динамически.
- *int len*– текущая длина слова.
- *int size* – текущий размер слова.
- *char char_end* – последний символ слова (без учета разделителя).

1.2. Sentence

Структура Sentence содержит в себе следующие поля:

- *Word *words* –массив структур Word, представляющий это предложение, который создается и изменяется динамически.
- *int len_sen* – текущая длина предложения в словах.
- *int size* – текущий размер предложения.

1.3. Text

Структура Text содержит в себе следующие поля:

- *Sentence *text* – массив структур Sentence, представляющий этот текст, который создается и изменяется динамически.
- *int size* – текущий размер текста.
- *int len_text* – текущая длина текста в предложениях.

2. ВВОД

Для сохранения входных данных в структуру Text была реализована функция *void read_Text(Text *txt),* которая принимает указатель на инициализированную структуру *Text*, в которую будут сохраняться данные.

Внутри этой функции осуществляется вызов функции *Sentence read_Sentence(Sentence *sent),* которая принимает указатель на инициализированную структуру *Sentence*, в которую будут сохраняться данные.

В этой функции, в свою очередь, осуществляется вызов функции *Word read_Word(Word *word),* которая принимает указатель на инициализированную структуру *Word*, в которую будут сохраняться данные. Данная функция посимвольно считывает слова и записывает информацию о них, пока не встретит один из символов конца слова.

Считанные слова сохраняется в поле структуры предложений, пока не будет встречен символ конца предложения.

Считанные предложения сохраняются в поле текста, пока не встретят *«\n»*.

Такое считывание позволяет нам сохранить необходимую информацию о каждой единице текста, и предоставляет удобный доступ к этой информации.

3. УДАЛЕНИЕ ПОВТОРНО ВСТРЕЧАЮЩИХСЯ ПРЕДЛОЖЕНИЙ

Для удаления повторно встречающихся предложений была реализована функция *void rm_sent(Text *txt, int id)*, принимающая указатель на текст, из которого нужно удалить предложение и индекс, удаляемого предложения .

Определить являются ли предложения идентичными нам позволяет фунция *int identical_sent(Sentence *sent_1, Sentence *sent_2),* которая принимает указатели на два предложения и сравнивает их посимвольно.

Для пренебрежения регистром была реализована функция *char *tolower_sent(Sentence *sent),* принимающая указатель на предложение. Она преобразовывала предложение в нижнему регистру и возвращала только массив символов (1 поле структуры предложение), что было сделано для удобства сравнения предложений .

4. ФУНКЦИИ

4.1. Подсчёт количества слов “garbage” в каждом предложении.

По условию, по этому запросу программа должна вывести информацию о количестве слов *“garbage”* в каждом предложении.

Для ответа на этот запрос в файле была реализована функция *void dump_txt(Text *txt),* которая принимает указатель на текст. В этой функции текст перебирается по словам, сравнивая каждое слово со словом *“garbage”* без учета регистра. Для подсчёта количества слов была отведена отдельная переменная, в зависимости от которой на экран выводилось или “Clean”, или “Must be washed”, или “It is a dump”.

Для удобства сравнения слов была реализована функция c*har **tolower_word(Sentence *sent),* принимающая предложение и, преобразующая его в двумерный массив символов в нижнем регистре.

4.2. Замена всех цифр в предложении на введенную строку.

Для выполнения задачи была реализована функция *void replace_numbers(Text *txt),* принимающая указатель на текст. Функция посимвольно проверяет каждый символ текста на то, является ли он цифрой, если да, то вместо этой цифры на экран выводится в заранее введённая строка, в противном случае символ остается неизменным.

4.3. Удаление предложений, содержащих три подряд идущие буквы в верхнем регистре, из текста.

Для выполнения данной задачи была реализована функция *void rm_three_up(Text *txt),* которая принимает указатель на текст. Функция посимвольно проверяет каждый символ текста на то, является ли он буквой в верхнем регистре. Если является, то в функции выполняется вызов ранее

описанной функции *void rm_sent(Text *txt, int id)*, удаляющей предложение по индексу.

4.4. Сортировка предложений по количеству слов, начинающихся с гласной.

Для выполнения данной задачи, была реализована *void sort_vowels_count(Text *txt)*, в которой выполняется вызов функции *qsort* из стандартной библиотеки. Для нее была реализована функция *int sent_comparison(const void *s1, const void *s2)*, которая возвращала разницу в количестве слов, начинающихся с гласной буквы.

Подсчёт количества слов в предложении, начинающихся с гласной, проводился при помощи функции *int count_vowels_word(Sentence *sent)*.

5. ОСНОВНАЯ ПРОГРАММА

Для удобства использования программы в ней присутствуют подсказки для пользователя.

Сначала пользователю необходимо ввести текст, обработка которого будет проводиться в программе. Далее необходимо выбрать одно из предложенных в подсказке действий и ввести необходимый символ (предусмотрена возможность выхода из программы).

Вывод текста осуществляется при помощи функции *void display_text(Text *txt).*

После выполнения программы происходит очищение выделенной памяти.

6. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Пример 1:

Текст для проверки корректности выполнения задачи 1:

I love garbage.she is garbage,and he too garbage.axaxax.GArbage.garbage garbage garbage garbage garbage garbage.Ilove garbage.she is garbage,and he too garbage.axaxax.GArbage.garbage garbage garbage garbage garbage garbage.I love garbage.she is garbage,and he too garbage.axaxax.GArbage.garbage garbage garbage garbage garbage garbage.I love garbage.she is garbage,and he too garbage.axaxax.GArbage. garbage garbage garbage garbage garbage garbage.

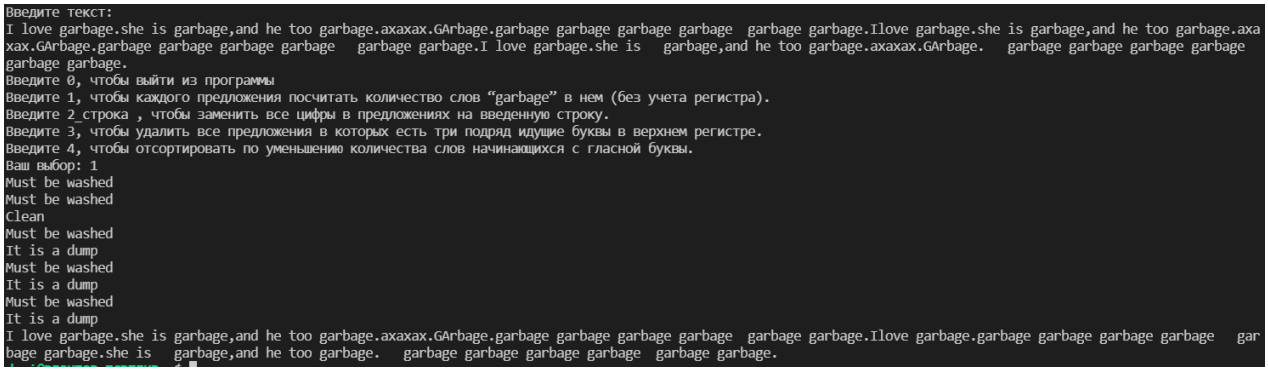


Рисунок 1 – Пример ответа программы на 1 запрос

Пример 2:

Текст для проверки корректности выполнения задачи 2:

2 2 2 2 2 2 2 2 2 2 2 2 2. 333333333333. Hell4,wo7d.

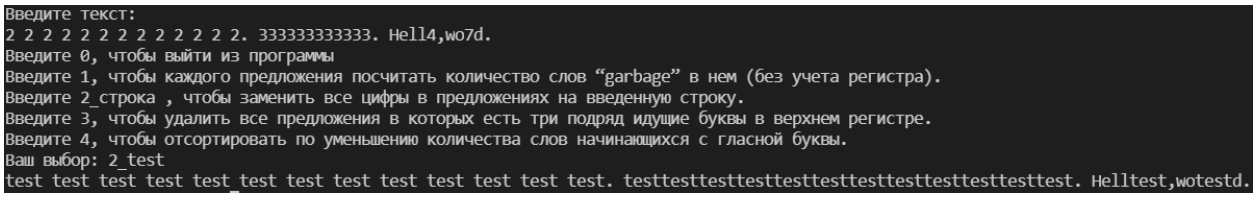
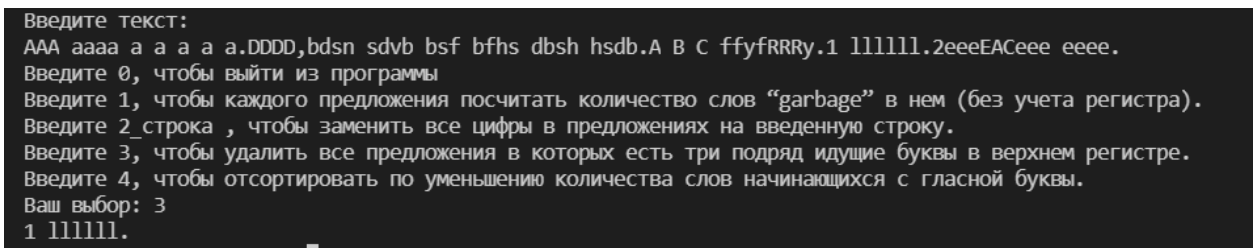


Рисунок 2 – Пример ответа программы на 2 запрос

Пример 3:

Текст для проверки корректности выполнения задачи 3:

AAA aaaa a a a a a.DDDD,bdsn sdvb bsf bfhs dbsh hsdb.A B C ffyfRRRy.1 llllll.2eeeEACeee eeee.



Пример 4:

Текст для проверки корректности выполнения задачи 4:

2) a a a a.3)b Acfcf Dcc OOO.1)axax acac acaca acac YYY UUU.4)allo girl.

```
Введите текст:
2) a a a a.3)b Acfcf Dcc OOO.1)axax acac acaca acac YYY UUU.4)allo girl.
Введите 0, чтобы выйти из программы
Введите 1, чтобы каждого предложения посчитать количество слов “garbage” в нем (без учета регистра).
Введите 2_строка , чтобы заменить все цифры в предложениях на введенную строку.
Введите 3, чтобы удалить все предложения в которых есть три подряд идущие буквы в верхнем регистре.
Введите 4, чтобы отсортировать по уменьшению количества слов начинающихся с гласной буквы.
Ваш выбор: 4
1)axax acac acaca acac YYY UUU.2) a a a a.3)b Acfcf Dcc OOO.4)allo girl.
```

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы, были применены на практике приёмы работы с динамической памятью. Были использованы стандартные библиотеки языка программирования Си. Была реализована программа, которая выполняет функции, указанные в задании. Программа осуществляет взаимодействие с пользователем через консольный интерфейс и корректно обрабатывает различные виды выходных данных в соответствии с условием.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ.

1. Керниган Б. и Ритчи Д. Язык программирования Си. М.: Вильямс, 1978 288с.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

#define MEMORY_1 1000
#define MEMORY_2 100
#define MEMORY_3 10

//-------------------------STRUCT--------------------------------

typedef struct Word {
    char *word;
    int len;
    char char_end;
    int size;
} Word;

typedef struct Sentence {
    Word *sent;
    int len_sen;
    int size;
} Sentence;


typedef struct Text {
    Sentence *text;
    int len_text;
    int size;
} Text;

//------------------FREE-----------------------------------

void free_Word(Word *word) {
    free(word->word);
    word->size = 0, word->len = 0, word->char_end = 0;
}

void free_Sentence(Sentence *sent) {
    for (int i = 0; i < sent->len_sen; ++i)
        free_Word(&sent->sent[i]);
    free(sent->sent);
    sent->size = 0, sent->len_sen = 0;
}

void free_Text(Text *txt) {
    for (int i = 0; i < txt->len_text; ++i)
        free_Sentence(&txt->text[i]);
    free(txt->text);
    txt->size = 0, txt->len_text = 0;
}
```

```
//-------------------READ----------------------------------------

Word read_Word(Word *word) {
    word->size = MEMORY_3;
    word->word = (char *) malloc(word->size * sizeof(char));
    word->len = 0;
    word->char_end = 'a';
    do {
        if (word->len > word->size - 2) {
            word->size *= 2;
            word->word = (char *) realloc(word->word, word->size *
sizeof(char));
            if (word->word == NULL) {
                puts("Невозможно выделить память");
                exit(0);
            }
        }
        word->word[word->len] = getchar();
        word->len++;
    } while (word->word[word->len - 1] != ' ' && word->word[word->len -
1] != ',' && word->word[word->len - 1] != '.' &&
             word->word[word->len - 1] != '\n');
    word->word[word->len] = '\0';
    word->char_end = word->word[word->len - 1];
    return *word;
}

Sentence read_Sentence(Sentence *sent) {
    sent->size = MEMORY_2;
    sent->sent = (Word *) malloc(sent->size * sizeof(Word));
    char flag = 0;
    sent->len_sen = 0;
    do {
        if (sent->len_sen >= sent->size - 2) {
            sent->size = sent->size * 2;
            sent->sent = (Word *) realloc(sent->sent, sent->size *
sizeof(Word));
            if (sent->sent = NULL) {
                puts("Невозможно выделить память");
                exit(0);
            }
        }
        sent->sent[sent->len_sen]     =     read_Word(&sent->sent[sent-
>len_sen]);
        flag = sent->sent[sent->len_sen].char_end;
        sent->len_sen++;
    } while (flag != '.' && flag != '\n');
    return *sent;
}

void read_Text(Text *txt) {
    txt->size = MEMORY_1;
    txt->text = (Sentence *) malloc(txt->size * sizeof(Sentence));
    char flag = 0;
    txt->len_text = 0;
    do {
        if (txt->len_text >= txt->size - 2) {
            txt->text = (Sentence *) realloc(txt->text, txt->size * 2 *
```

```
sizeof(Sentence));
            txt->size = txt->size * 2;
            if (txt == NULL) {
                puts("Невозможно выделить память");
                exit(0);
            }
        }
        txt->text[txt->len_text]    =    read_Sentence(&txt->text[txt-
>len_text]);
        flag = txt->text[txt->len_text].sent->char_end;
        txt->len_text++;
    } while (flag != '\n');
}

//-----------------------FUNC---------------------------------

char **tolower_word(Sentence *sent) {
    int k1 = sent->len_sen;
    char **res = (char **) malloc(sent->len_sen * sizeof(Word));
    for (int i = 0; i < k1; i++) {
        res[i] = (char *) malloc(100);
        for (int j = 0; j < strlen(sent->sent[i].word); j++)
            res[i][j] = tolower(sent->sent[i].word[j]);
    }

    return res;
}

char *tolower_sent(Sentence *sent) {
    int k1 = sent->len_sen;
    int k2 = sent->sent[0].len;
    char *res = (char *) malloc(sent->len_sen * sizeof(Word));
    char *sentence = (char *) malloc(sent->len_sen * sizeof(Word));
    sentence = strcat(sentence, sent->sent[0].word);
    for (int i = 1; i < k1; i++) {
        k2 += sent->sent[i].len;
        char *str = sent->sent[i].word;
        sentence = strcat(sentence, str);
    }
    for (int i = 0; i < k2; i++) {
        res[i] = tolower(sentence[i]);
    }
    return res;
}

int identical_sent(Sentence *sent_1, Sentence *sent_2) {
    char *s1 = tolower_sent(sent_1);
    char *s2 = tolower_sent(sent_2);
    if (strcmp(s1, s2) == 0)
        return 1;
    return 0;
}

void rm_sent(Text *txt, int id) {
    if (id < 0 || id >= txt->len_text)
        return;
    free_Sentence(&txt->text[id]);
    memmove(&txt->text[id], &txt->text[id + 1], (txt->len_text - id -
```

```
1) * sizeof(Sentence));
    txt->text = (Sentence *) realloc(txt->text, sizeof(Sentence) * (--
txt->len_text));
    if (txt->text == NULL) {
        puts("Невозможно выделить память");
        exit(0);
    }
    txt->size = txt->len_text;
}

void rm_three_up(Text *txt) {
    int count_up = 0;
    for (int i = 0; i < txt->len_text; i++) {
        for (int j = 0; j < txt->text[i].len_sen; j++) {
            for (int k = 0; k < txt->text[i].sent[j].len; k++) {
                if (isupper(txt->text[i].sent[j].word[k]) != 0) {
                    count_up++;
                } else {
                    count_up = 0;
                }
                if (count_up == 3) {
                    count_up = 0;
                    rm_sent(txt, i);
                    i = 0;
                    k = 0;
                    j = 0;
                }
            }
        }
    }
}

void rm_identical_sent(Text *txt) {
    for (int i = 0; i < txt->len_text; ++i)
        for (int j = txt->len_text - 1; j > i; --j)
            if (identical_sent(&txt->text[i], &txt->text[j])) {
                rm_sent(txt, j);
            }
}

void dump_txt(Text *txt) {
    char **s = (char **) malloc(1000);
    char orig[] = "garbage";
    int count_garbage = 0;
    for (int i = 0; i < txt->len_text - 1; i++) {
        s = tolower_word(&txt->text[i]);
        for (int j = 0; j < txt->text[i].len_sen; j++) {
            if (strncmp(s[j], orig, 7) == 0)
                count_garbage++;
        }
        if (count_garbage == 0)
            puts("Clean");
        else if (count_garbage < 6 && count_garbage > 0)
            puts("Must be washed");
        else if (count_garbage > 5)
            puts("It is a dump");
        free(s);
        count_garbage = 0;
```

```
    }
}

int count_vowels_word(Sentence *sent) {
    int count_w = 0;
    const char *st = "AEIOUYaeiouy";
    for (int i = 0; i < sent->len_sen; i++) {
        char ch_0 = sent->sent[i].word[0];
        if (strchr(st, ch_0) != NULL) {
            count_w++;
        }
    }
    return count_w;
}

int sent_comparison(const void *s1, const void *s2) {
    Sentence *s_1 = (Sentence *) s1;
    Sentence *s_2 = (Sentence *) s2;
    int count_s_1 = count_vowels_word(s_1);
    int count_s_2 = count_vowels_word(s_2);
    return count_s_2 - count_s_1;
}

void sort_vowels_count(Text *txt) {
    qsort(txt->text, txt->len_text, sizeof(Sentence), sent_comparison);
}


void replace_numbers(Text *txt) {
    Word *str_r = (Word *) malloc(MEMORY_2 * sizeof(Word));
    *str_r = read_Word(str_r);
    str_r->word[str_r->len - 1] = '\0';
    for (int i = 0; i < txt->len_text; i++) {
        for (int j = 0; j < txt->text[i].len_sen; j++) {
            for (int k = 0; k < txt->text[i].sent[j].len; k++) {
                if (isdigit(txt->text[i].sent[j].word[k]) != 0)
                    printf("%s", str_r->word);
                else
                    printf("%c", txt->text[i].sent[j].word[k]);


            }

        }

    }

}

void display_text(Text *txt) {
    for (int i = 0; i < txt->len_text; i++) {
        for (int j = 0; j < txt->text[i].len_sen; j++)
            printf("%s", txt->text[i].sent[j].word);
    }

}

//--------------------------MAIN-----------------------------
```

```
int main() {
    Text txt;
    puts("Введите текст:");
    read_Text(&txt);
    rm_identical_sent(&txt);
    char fn;
    puts("Введите 0, чтобы выйти из программы");
    puts("Введите 1, чтобы каждого предложения посчитать количество слов
“garbage” в нем (без учета регистра).");
    puts("Введите 2_строка , чтобы заменить все цифры в предложениях на
введенную строку.");
    puts("Введите 3, чтобы удалить все предложения в которых есть три
подряд идущие буквы в верхнем регистре.");
    puts("Введите 4, чтобы отсортировать по уменьшению количества слов
начинающихся с гласной буквы.");
    printf("Ваш выбор: ");
    scanf("%c_", &fn);
    switch (fn) {
        case '0':
            puts("Вы покинули программу!");
            exit(0);
        case '1':
            dump_txt(&txt);
            display_text(&txt);
            break;
        case '2':
            replace_numbers(&txt);
            break;
        case '3':
            rm_three_up(&txt);
            display_text(&txt);
            break;
        case '4':
            sort_vowels_count(&txt);
            display_text(&txt);
            break;
        default:
            puts("Ваш выбор некорректный");
            exit(0);
    }
    free_Text(&txt);
    return 0;

}
```