

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

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Программирование»
Тема: Обработка строк на языке С

Студент гр. 0382

Павлов С. Р.

Преподаватель

Жангиров Т.Р

Санкт-Петербург

2020

ЗАДАНИЕ

ВАРИАНТ 1

Студент: Павлов С.Р

Группа: 0382

Тема работы: Обработка строк на языке Си.

Программе на вход подается текст (текст представляет собой предложения, разделенные точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских букв, и цифр. Длина текста и каждого предложения заранее не известна.

Программа должна сохранить этот текст в динамический массив строк и оперировать далее только с ним.

Программа должна найти и удалить все повторно встречающиеся предложения (сравнивать их следует посимвольно, но без учета регистра).

Далее, программа должна запрашивать у пользователя одно из следующих доступных действий (программа должна печатать для этого подсказку. Также следует предусмотреть возможность выхода из программы):

1. Преобразовать предложения так, чтобы каждое первое слово в нем начиналось с заглавной буквы, а остальные символы были прописными.
2. Удалить все предложения, состоящие из четного количества слов.
3. Отсортировать предложения по сумме количеств гласных букв в каждом втором слове. Сумма для предложения “abc qwe defgh prq ijklmno” будет такой: abc = 1, defgh=1, ijklmno=2, итоговая сумма = 4.
4. Вывести на экран все предложения, в которых в середине слова встречаются слова, состоящие из прописных букв. Данные слова нужно выделить синим цветом.

Все сортировки должны осуществляться с использованием функции стандартной библиотеки. Использование собственных функций, при наличии аналога среди функций стандартной библиотеки, запрещается.

Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции.

Предполагаемый объем пояснительной записки не менее 12 страниц.

Дата выдачи задания: 02.11.2020

Дата защиты реферата: 26.12.2020

Студент гр. 0382

Павлов С. Р.

АННОТАЦИЯ

В процессе выполнения курсовой работы создавалась программа для обработки введенного пользователем текста в зависимости от операции, который выберет пользователь . Для хранения текста память выделяется динамично. Разработка велась на операционной системе Windows 10 x64 в редакторе исходного кода CLion с использованием компилятора MinGW.

СОДЕРЖАНИЕ

1. Введение	
2. Ход выполнения работы.	7
2.1 Ввод текста	8
2.2 Первичная обработка	9
2.3 Первая задача	10
2.4 Вторая задача	11
2.5 Третья задача	12
2.6 Четвертая задача	13
3. Заключение	14
Список использованных источников	15
Приложение А. Примеры работы программы	16
Приложение Б. Исходные код программы	17

ВВЕДЕНИЕ

Цель работы – создать консольное приложение для обработки текста согласно запросам пользователя.

Для выполнения работы необходимо выполнить следующие задачи:

- Ввод, хранение , и вывод текста.
- Провести первичные изменения.
- Реализовать проверку регистра слов для первой операции.
- Создать функции для удаления четных слов в предложениях
- Реализовать сортировку для третьей операции.
- Создать возможность выделения цветом для четвертой операции.

Для первой задачи, для хранения текста используется динамическая память, методы которой есть в *stdlib.h* . Ввод текста осуществляется через специальную для этого функцию, которая обрабатывает входные данные посимвольно.

Для второй задачи, используется простая функция, которая удаляет пробелы уже в записанных предложениях.

Для третьей задачи используется функции работающие посимвольно с предложениями и преобразовывающие их.

Для четвертой задачи используется функции удаляющие предложения в словах.

Для пятой задачи используется библиотечная сортировка *qsort()*, которая описана в заголовочной файле *stdlib.h*

ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1 Ввод теста

В начале работы программы, пользователю предлагается ввести текст. Для этого реализована функция *get_sentence()*, она инициализирует массив-строку *tmp_sent*, и выделяет ей 100 байт памяти (типа *char*). Далее функция посимвольно обрабатывает входной поток. Конечным условием для цикла этой функции будет является элемент пробел. Так в функции предусмотрено динамическое выделение памяти, когда длина предложения превосходит текущее значение памяти, то функция с помощью *realloc*, увеличивает память *tmp_sent*. По итогу функция возвращает указатель на уже записанное предложение.

2.2. Первичная обработка

По условия задания, перед выполнением задач, нужно удалить все повторяющиеся предложения в хранящимся тексте. Это осуществляется с помощью функций *check_similar(char** text, char* sentence, int text_sentences)* и *is_similar_sents(char* strA, char* strB)*. Изначальное предложение которое поступает на вход, сразу проходит проверку на оригинальность, т.е оно попадает в *check_similar*, там оно сравнивается со всеми предложениями,

посимвольно с помощью функции *is_similar_sent*. Если предложение повторяется, то оно попросту не добавляется в итоговый текст.

2.3 Выбор операции

С помощью *printf()*, пользователь получает справку о возможных действиях. Далее с помощью *scanf()*, пользователь вводит в консоль нужную ему операцию (1-4). Также при введении нуля (0), программа завершается. Затем, с помощью оператора *switch*, выполняется нужная подзадача.

2.4. Первая задачи

Первая задача заключается в том чтобы, сделать каждую первую букву в предложении заглавной, а остальные прописными, и так для всех предложений.

Данная задача реализуется с помощью функций *upper_transform(textm text_sentences)* обрабатывается и выводится общей функцией *print_text(text, text_sentences)*; Первая функция типа *void*, она работает и изменяет непосредственно сами предложения. Первый цикл функции перебирает предложения, делая их первый символ — заглавным. Затем второй вложенный цикл перебирает все символы, делая их прописными. Так же в этом цикле есть проверка, что является ли символ уже прописным, это сделано чтобы программа не делала лишних действий. Все изменения записываются непосредственно в память.

2.4 Вторая задача

Удалить все предложения состоящие из четного количества слов.

Данная задача выполняется с помощью функций *delete_even_word(char** text, int text_sentences)*, *how_words(char* sentence)* и *print_text(text, text_sentences)*.

Первая функция перебирает предложения, и с условием с помощью дополнительной функции проверяет четное ли, количество слов в предложении.

Если слов в предложении четно, то первому элементу предложения присваивается нуль-символ „\0“.

Вторая функция *how_word(char* sentence)*, считает кол-во слов в предложении и возвращает unsigned int значение. Функция посимвольно сравнивает проходит по предложению, и ищет знаки конца слова (« », «,»), затем итерируя переменную words.

Последняя функция *print_text(char** text, int text_sentences)*, выводит строки в котором первым элементом не является „\0“ (нуль-символ).

2.5 Третья задача

Отсортировать предложения по сумме количеств гласных букв в каждом слове.

Данная задача, выполняется с помощью библиотечной сортировки *qsort()*. Она принимает такой вид *qsort(text, text_sentences, sizeof(char*), compare_vowel)*. Рассмотрим функцию *compare_vowel(const void *a, const void *b)*, она имеет стандартный вид для функции сравнения, непосредственно для *qsort()*. Для этой функции надо найти кол-во гласных в предложении, это реализуется с помощью функций *how_vowels(char* sentence)*. С помощью простой функции *is_vowel()*, (проверка гласная ли буква) реализованной на основе оператора *switch()* происходит подсчет кол-ва гласных в словах. Разделения слов для функции *how_vowels* идентично ранее рассмотренной ранее *how_words()*.

Таким образом по итогу, сортировка изменяет (сортирует) двумерный массив *text* и далее с помощью, функции *print_text()* он выводится на экран.

2.6 Четвертая задача

Вывести на экран все предложения, в которых в середине слова встречаются слова состоящие из прописных букв. И покрасить их в синий цвет.

Данная задача реализуется с помощью ряда данных функций:

lower_words_display(text, text_sentences), get_indexes(char sentence), check_lower(char* sentence),* и функции-вывода *moddif_print(char* sentences, int* indexes);*

Главная функция *lower_words_display()* итерирует каждое предложение. Далее происходит проверка на кол-во слов в этом предложении, если оно четное, то выполняется первое условие, а если нечетное то второе.

С помощью функции *get_indexes()* , основная функция получает индексы двух средних слов в предложении. Далее происходит проверка являются ли слова в этих предложениях прописными, если да то с помощью функции *moddif_print()* , они выводятся на экран, и слова серединные слова красятся в синий цвет.

Выделение цветом и печать происходит в функции *moddif_print()*, выделение синим происходит перед индексом первого элемента и конечного элемента. Это реализовано с помощью ASCII кода. Сначала установка символ-кода перед словом, а в конце возвращение символ-кодом, к стандартному цвету.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было создано консольное предложение для обработки текста согласно запросам пользователя. Все задачи также были успешно выполнены: программа преобразовывает предложения с заглавной буквы; удаляет предложения с четным кол-вом слов; сортирует предложения по сумме количеств главных букв в каждом слове.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сайт (онлайн-справочник) www.c-cpp.ru

ПРИЛОЖЕНИЕ А

ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

QQ,QQ. TT TT. QQ,QQ. Aa aa.

Выбор действия:

- {1} - Сделать первую букву в каждом предложении заглавной, а остальные символы в предложении прописными.
- {2} - Удалить предложения состоящие из четного кол-ва слов.
- {3} - Отсортировать предложения по сумме количеств гласных букв в каждом втором слове.
- {4} - Вывести на экран все предложения, в которых в середине слова встречаются слова, состоящие из прописных букв.
- {0} - Завершить программу.

Command: 1

Qq,qq. Tt tt. Aa aa.

Прим
ер 1
—
удале
ние

повторяющихся предложений в начале.

He likes to work. Lore Ipsum. TTT,QQQ AAA.

Выбор действия:

- {1} - Сделать первую букву в каждом предложении заглавной, а остальные символы в предложении прописными.
- {2} - Удалить предложения состоящие из четного кол-ва слов.
- {3} - Отсортировать предложения по сумме количеств гласных букв в каждом втором слове.
- {4} - Вывести на экран все предложения, в которых в середине слова встречаются слова, состоящие из прописных букв.
- {0} - Завершить программу.

Command: 2

TTT,QQQ AAA.

Приме
р 2 —
удале
ние
предл

ожений с четным кол-вом слов.

```
AAA XXX 000. AAAAA XXXX YYYYYYYY. AA XX EE XX AA 00.
```

Выбор действия:

- {1} - Сделать первую букву в каждом предложении заглавной, а остальные символы в предложении прописными.
 - {2} - Удалить предложения состоящие из четного кол-ва слов.
 - {3} - Отсортировать предложения по сумме количеств гласных букв в каждом втором слове.
 - {4} - Вывести на экран все предложения, в которых в середине слова встречаются слова, состоящие из прописных букв.
- {0} - Завершить программу.

Command: 3

```
AAAAA XXXX YYYYYYYY. AAA XXX 000. AA XX EE XX AA 00.
```

П

пример

3 —

сортир

овка

букв

по

каждой второй гласной букве.

```
aaa aaa aaa aaa. Lore Ipsum aaaaa. Going to commit this.
Выбор действия:
  {1} - Сделать первую букву в каждом предложении заглавной, а о
        стальные символы в предложении прописными.
  {2} - Удалить предложения состоящие из четного кол-ва слов.
  {3} - Отсортировать предложения по сумме количеств гласных бук
        в в каждом втором слове.
  {4} - Вывести на экран все предложения, в которых в середине с
        лова встречаются слова, состоящие из прописных букв.

  {0} - Завершить программу.
Command: 4

aaa aaa aaa aaa. Lore Ipsum aaaaa. Going to commit this. ✖
```

Пример 4 — вывод предложений в которых слова подсвечены синим
имеют прописные символы.

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

#define mem_add 100
#define RESET "\033[0m"
#define BLUE  "\033[34m"

// Прием предложения
char* get_sentence(){

    char* tmp_sent = malloc(100*sizeof(char));
    int tmp_mem = 50;
    int len=0;
    char ch;

    while(1){

        ch = getchar();
        //printf("CH: [%c]\t", ch);
        if (ch == '\n')
            return NULL;

        tmp_sent[len] = ch;
        len += 1;

        if (ch == '.') break;

        if (len>=tmp_mem){
            tmp_mem += mem_add;
            tmp_sent = realloc(tmp_sent, tmp_mem);
        }

    }

    if(tmp_sent[0] == ' '){
        char* r_tmp_sent = malloc((tmp_mem-1)*sizeof(char));

        for(int j=0;j < len;j++){

            r_tmp_sent[j] = tmp_sent[j+1];

        }
        r_tmp_sent[len-1] = '\0';
        return r_tmp_sent;
    }
}
```

```

        else {
            tmp_sent[len] = '\0';
            return tmp_sent;
        }
    }

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

    _Bool check_similar(char** text, char* sentence, int
text_sentences){
        int i;
        for (i=0;i<=text_sentences-1;i++){
            //printf("Text[%d]: [%s] Sent: [%s]\n", i,text[i],sentence);
            if (is_similar_sents(text[i], sentence)){
                return 0;
            }
        }
        return 1;
    }

    void print_text(char** text, int text_sentences){
        for (int k=0;k<text_sentences; k++){
            if (text[k][0] != '\0'){
                printf("%s ",text[k]);
                free(text[k]);
            }
        }
        printf("\n");
        free(text);
    }

    void upper_transform(char** text, int text_sentences){
        unsigned int len_str;
        int num;
        for (int i=0;i<text_sentences;i++){
            len_str = strlen(text[i]);
            text[i][0] = (char)toupper(text[i][0]);

            for(int j=1;j<len_str;j++){

```

```

        num = ((int) text[i][j]);
        if ((num<91)&&(num>64)){
            //printf("CH: [%c]\n", text[i][j]);
            text[i][j] = ((char) tolower(text[i][j]));
        }
    }
}

unsigned int how_words(char* sentence){
    unsigned int sent_len = strlen(sentence);
    unsigned int words=0;
    char ch;
    char pre_ch;
    for(int i=0;i<(sent_len-1);i++){
        ch = sentence[i];
        if (((ch == ' ')||(ch == ','))&&(pre_ch != ','))
            words += 1;
        pre_ch = ch;
    }
    words += 1;
    return words;
}

void delete_even_word(char** text, int text_sentences){
    for (int i=0;i<text_sentences;i++){
        //printf("Sent:          [%s]Words:          %i\n",text[i],
how_words(text[i]));

        if (how_words(text[i])%2 == 0)
            text[i][0] = '\0';
    }
}

int is_vowel(char c) {
    switch(tolower(c)){
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'y':
            return 1;
        default:
            return 0;
    }
}

int how_vowels(char* sentence){
    int vowels = 0;

```

```

    unsigned int str_len = strlen(sentence);
    int word = 1;
    char ch,pre_ch;

    for (int i=0;i<str_len;i++){

        // (!!!) CJOBA
        ch = sentence[i];
        if (((ch == ' ')||(ch == ','))&&(pre_ch != ','))
            word += 1;
        pre_ch = ch;
        if ((is_vowel(sentence[i]) == 1)&&(word%2==1))
            vowels += 1;
    }
    return vowels;
}

int compare_vowel(const void *a, const void *b)
{
    char **x = (char**)a;
    char **y = (char**)b;

    int xx = how_vowels(*x);
    int yy = how_vowels(*y);

    return (yy-xx);
}
/*
char** split_words(char* sentence){
    unsigned int str_len = strlen(sentence);
    unsigned int words = how_words(sentence);
    char** words_mass = (char**) malloc(100*sizeof(char*));
    int word=0;
    char ch,pre_ch;
    int j=0,i=0;

    for(i=0;i<strlen;i++){
        char* tmp_str = malloc(str_len*sizeof(char));

        ch = sentence[i];

        if ((ch != ' ')&&(ch != ',')&&(ch != '.')){
            tmp_str[j] = ch;
            printf("CH: %c\n", tmp_str[j]);
        }

        if ((ch == ' ')||(ch == ',')) {
            tmp_str[j] = '\0';
            printf("STR %s\n", tmp_str);
            words_mass[word] = tmp_str;
            j = 0;
            word += 1;
        }
    }
}

```



```

        if (ch=='.') {
            tmp_str[j] = '\0';
            break;
        }
    }

    printf("Word[0]: %s\n", words_mass[0]);
    printf("Word[1]: %s\n", words_mass[1]);
    printf("Word[2]: %s\n", words_mass[2]);
    printf("Word[3]: %s\n", words_mass[3]);
    return words_mass;
}
*/

int* get_indexes(char* sentence){

    unsigned int words = how_words(sentence);

    int* ans = malloc(10*sizeof(int));
    unsigned int sent_len = strlen(sentence);
    char ch, pre_ch;
    int word=0;

    ans[0] = 0;
    ans[1] = 0;
    ans[2] = 0;
    ans[3] = 0;

    if (words%2 == 0) {
        unsigned int first_word_i = (words/2)-1;
        unsigned int second_word_i = (words/2);
        for (int i = 0; i < sent_len; i++) {

            ch = sentence[i];
            if ((ch == ' ') || (ch == ',')) && (pre_ch != ',')
                word += 1;

            if ((word == first_word_i) && ((ch == ' ') || (ch ==
',',')))) {

                ans[0] = i + 1;
                int i_tmp = i;

                while (1) {
                    ch = sentence[i_tmp + 1];

                    if ((ch == ' ') || (ch == ',')) {
                        ans[1] = i_tmp;
                        break;
                    }

                    i_tmp += 1;

```

```

        }

    }

    if ((word == second_word_i) && ((ch == ' ') || (ch ==
', '))) {

        ans[2] = i + 1;
        int i_tmp = i;

        while (1) {
            ch = sentence[i_tmp + 1];

            if ((ch == ' ') || (ch == ', ')) {
                ans[3] = i_tmp;
                break;
            }

            i_tmp += 1;
        }

    }

    pre_ch = ch;
}
else{
    unsigned int first_word_i = (words/2);
    unsigned int second_word_i = (words+1/2);
    for (int i = 0; i < sent_len; i++) {

        ch = sentence[i];
        if (((ch == ' ') || (ch == ', ')) && (pre_ch != ', '))
            word += 1;

        if ((word == first_word_i) && ((ch == ' ') || (ch ==
', '))) {

            ans[0] = i + 1;
            int i_tmp = i;

            while (1) {
                ch = sentence[i_tmp + 1];

                if ((ch == ' ') || (ch == ', ')) {
                    ans[1] = i_tmp;
                    break;
                }

                i_tmp += 1;
            }

        }

    }
}

```

```

        if ((word == second_word_i) && ((ch == ' ') || (ch ==
', '))) {
            ans[2] = i + 1;
            int i_tmp = i;

            while (1) {
                ch = sentence[i_tmp + 1];

                if ((ch == ' ') || (ch == ', ')) {
                    ans[3] = i_tmp;
                    break;
                }

                i_tmp += 1;
            }

        }

        pre_ch = ch;
    }
}

return ans;
}

```

```

_Bool check_lower(char* sentence, int* indexes, int multi){
    char ch;

    if (multi == 1){
        int i = indexes[0];
        _Bool tmp1=1;
        _Bool tmp2=1;
        while (i<=indexes[1]){
            ch = sentence[i];
            int num = (int) ch;
            if ((num<91)&&(num>64)) {
                tmp1 = 0;
                break;
            }
            i += 1;
        }
        i = indexes[2];
        while (i<=indexes[3]){
            ch = sentence[i];
            int num = (int) ch;
            if ((num<91)&&(num>64)) {
                tmp2 = 0;
                break;
            }
            i += 1;
        }
    }
}

```

```

        if (tmp1 == tmp2)
            return 1;
        else return 0;

    }
    else if (multi == 0){
        int i = indexes[0];
        _Bool tmp1=1;
        while (i<=indexes[1]){
            ch = sentence[i];
            int num = (int) ch;
            if ((num<91)&&(num>64)) {
                return 0;
            }
            i += 1;
        }
        return tmp1;
    }
}

void moddiff_print(char* sentence, int* indexes){

    unsigned len = strlen(sentence);
    int words = how_words(sentence);

    char ch;
    int word=0;

    for (int i=0;i<len;i++){

        ch = sentence[i];

        if (i==indexes[0]) printf("%s", BLUE);

        if (i==indexes[1]+1) printf("%s", RESET);

        if (words%2 == 0) {
            if (i == indexes[2]) printf("%s", BLUE);

            if (i == indexes[3] + 1) printf("%s", RESET);
        }
        printf("%c", ch);
    }
    printf(" ");
}

void lower_words_display(char** text, int text_sentences){

    unsigned int sent_len;
    unsigned int words;
    char ch;

```

```

int* indexes;
// Перебор предложений
for (int i=0;i<text_sentences;i++){

    sent_len = strlen(text[i]);
    words = how_words(text[i]);
    //EVEN
    if ((words % 2)== 0){
        indexes = get_indexes(text[i]);
        // если прописные то
        if ((check_lower(text[i], indexes, 1))==1){
            moddiff_print(text[i], indexes);
            free(text[i]);
        }
        else{
            printf("%s ",text[i]);
            free(text[i]);
        }
    }
    //ODD
    if ((words % 2) != 0){
        indexes = get_indexes(text[i]);
        // если прописные то
        if ((check_lower(text[i], indexes, 1))==1){
            moddiff_print(text[i], indexes);
            free(text[i]);
        }
        else{
            printf("%s ",text[i]);
            free(text[i]);
        }
    }
}
free(text);
}

int main() {
    // доп
    //инициализация
    char **text = malloc(200 * sizeof(char *));
    char *sentence;
    int text_sentences = 0;
    int text_mem = 200;
    int i = 0;
    //Цикл приема и пред-обработки

    while (1) {
        sentence = get_sentence();
        // printf("[STR]: %s\n", sentence);
    }
}

```

```

        if (sentence == NULL) break;
        else {
            // Добавление памяти
            if (text_sentences >= text_mem) {
                //printf("(!) Mem added!\n");
                text_mem += mem_add;
                text = realloc(text, (text_mem * sizeof(char *)));
            }
            // Проверка на одинаковые и запись в text
            if ((check_similar(text, sentence, text_sentences) != 0)
|| (text_sentences == 0)) {
                //printf(" |--> Added: %s\n", sentence);
                text[text_sentences] = sentence;
                text_sentences += 1;
            }
        }
    }

    int command;

    printf("Выбор действия:\n");
    printf("    {1} - Сделать первую букву в каждом предложении
заглавной, а остальные символы в предложении прописными.\n");
    printf("    {2} - Удалить предложения состоящие из четного кол-ва
слов.\n");
    printf("    {3} - Отсортировать предложения по сумме количеств
гласных букв в каждом втором слове.\n");
    printf("    {4} - Вывести на экран все предложения, в которых в
середине слова встречаются слова, состоящие из прописных букв.\n");
    printf("\n    {0} - Завершить программу.\n");

    printf("Command: ");

    scanf("%d", &command);
    printf("\n");

    //Выбор команды
    switch (command) {
        case 0:
            break;
        case 1:
            upper_transform(text, text_sentences);
            print_text(text, text_sentences);
            break;
        case 2:
            delete_even_word(text, text_sentences);
            print_text(text, text_sentences);
            break;
        case 3:
            qsort(text, text_sentences, sizeof(char*), compare_vowel);
            print_text(text, text_sentences);
            break;
    }

```

```
        case 4:
            lower_words_display(text, text_sentences);
            break;
        default:
            break;

    }
    return 0;
}
```