МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Математического обеспечения электронно-вычислительных машин

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Программирование»

ТЕМА: ИСПОЛЬЗОВАНИЕ УКАЗАТЕЛЕЙ.

Студентка гр. 0382 _____ Рубежова Н.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2020

Цель работы.

Изучить принципы работы с указателями и научиться применять их в программном коде.

Задание.

Напишите программу, которая форматирует некоторый текст и выводит результат на консоль.

На вход программе подается текст, который заканчивается предложением "Dragon flew away!".

Предложение (кроме последнего) может заканчиваться на:

- . (точка)
- ; (точка с запятой)
- ? (вопросительный знак)

Программа должна изменить и вывести текст следующим образом:

- Каждое предложение должно начинаться с новой строки.
- Табуляция в начале предложения должна быть удалена.
- Все предложения, в которых есть цифра 7 (в любом месте, в том числе внутри слова), должны быть удалены.
- Текст должен заканчиваться фразой "Количество предложений до n и количество предложений после m", где n - количество предложений в изначальном тексте (**без учета** терминального предложения "Dragon flew away!") и m - количество предложений в отформатированном тексте (без учета предложения про количество из данного пункта).

*** Порядок предложений не должен меняться**

*** Статически выделять память под текст нельзя**

*** Пробел между предложениями является разделителем, а не частью какого-то предложения**

Основные теоретические положения.

Указатель – это переменная, содержащая адрес другой переменной.

Синтаксис объявления указателя:

<тип_переменной_на_которую_ссылается_указатель> <название переменной>;*

Каждая переменная имеет своё место в оперативной памяти, т.е. адрес, по которому к ней обращается программа и может обращаться программист.

Унарная операция & даёт адрес объекта. Она применима только к переменным и элементам массива

У указателя также есть унарная операция разыменования *, которая позволяет получить значение ячейки, на которую ссылается указатель.

Строка в Си - массив символов, то массив строк это двумерный массив символов, где каждая строка - массив, хранящий очередную символьную строку.

Чтобы в функции изменить аргументы, мы должны передать этой функции указатели.

Выполнение работы.

Название файла: main.c

Подключение библиотек:

```
#include <stdio.h> //для выполнения стандартных функций
#include <stdlib.h> //для использования функций работы с памятью
#include <string.h> //для использования strlen()
#include <ctype.h> //для использования isalpha(),isdigit()
```

Опишем функцию *main():*

1. Сначала нужно реализовать ввод текста и запоминание его в двумерный динамический массив. Для того, чтобы выделить память определенного значения инициализируем переменные *int size_arr=SIZE_ARR*

и *int buffer=BUFFER,* которые будут отвечать за изначальный размер массива указателей на указатели char** *text* и массива указателей на символы *char* buf*.

В *buf* мы будем записывать строки-предложения исходного текста, а в элементы массива *text* указатели на эти «строки». Предварительно, укажем #define SIZE_ARR 10 и #define BUFFER 50, чтобы на этапе компиляции подставленные значения заменились на указанные нами.

2. Организуем посимвольный ввод текста, для этого инициализируем переменную *char c*, также переменную *int length*, которая будет фиксировать какой по счету символ мы записываем в строку нашего двумерного массива.

3. Инициализируем переменные-счетчики *int count_0=0* и *int count_1=0,* которые будут фиксировать количество предложений до и после обработки соответственно.

4. Выделяем память под массив указателей на указатели на символы *char** *text=malloc(size_arr*sizeof(char*)),* то есть под количество size_arr элементов типа указателя char*. Сколько будет строк-предложений в тексте, столько будет и элементов, если size_arr элементов будет мало для записи текста, то в дальнейшем расширим выделяемую память с помощью функции realloc.

5. Инициализируем переменную-флаг flag=1, чтобы изменяя значение флага, можно было прекратить ввод предложений и выйти из цикла *while(flag)*.

6. Заходим в цикл while(flag) и начинаем реализовывать ввод предложений.

7. Обнуляем *length=0,* чтобы при каждой итерации – начало ввода нового предложения счетчик обнулялся.

8. Выполняем ввод символа *c=getchar()*. Чтобы устранить табуляцию в начале предложения мы должны исключить, что первый символ каждого предложения *‘\t’,’ ‘, ‘\n’*. Если первый символ – символ табуляции, то снова вводим символ, никуда не записывая предыдущий. Также нужна проверка на второй символ – символ табуляции, так как в примере есть случай, когда после

предложения вводят символ переноса строки и затем еще один символ табуляции.

9. Выделяем память под буфер, который будет хранить символы вводимой строки-предложения текста: *char* buf=calloc(buffer,sizeof(char))*. Так как длина предложений неизвестна, длину буфера берем произвольную(инициализировали ранее), в дальнейшем, в случае достижения длины буфера, «расширим» выделенную память *realloc'*ом.

10. Запускаем цикл *while(c!='.'&&c!=';'&&c!='?'&&c!='!')*, чтобы вводить символы, пока не встретится символ окончания предложения.

11. В соответствии с счетчиком присваиваем элементу буфера введенный символ: buf[length++]=c, увеличивая счетчик для присваивания последующих символов.

12. Проверяем условие, чтобы при следующем присваивании не произошло переполнения. Если счетчик *length* достигает значения длины буфера – *buffer*, то расширяем выделяемую под буфер память, для этого сначала увеличиваем длину буфера *buffer*=2*, затем применяем buf=realloc(buf, buffer).

13. Заново вводим c=getchar(), чтобы на следующей итерации проверить условие цикла.

14. Как только вводится символ окончания предложения, выходим из цикла. Но нам нужно сохранить в строку и сам символ окончания предложения, так как на последней итерации счетчик увеличился, а переполнение исключилось, спокойно присваиваем последний введеный символ *buf[length]=c*. И последним символом ставим символ '\0', чтобы получить полноценную си-строку и в дальнейшем корректно ее выводить.

15. Наш буфер по итогу хранит си-строку - предложение нашего текста. Мы должны запомнить его, поэтому *text[count_0++]* передаем указатель на наш буфер: *text[count_0++]=buf*. Заодно будет считаться, сколько предложений мы ввели, то есть количество предложений до обработки.

16. При каждом вводе предложения нам нужно проверять, является ли оно терминальным, то есть мы можем сравнить наш буфер-строку со строкой “Dragon flew away!\0’ с помощью функции *strcmp()* из библиотеки <string.h>, если они равны, то функция вернет 0 и флаг изменится на *flag=0,* то есть ввод завершится, если строки различны, программа продолжит выполнять ввод в цикле.

17. Последней проверкой в цикле будет проверка, хватает ли нам размера массива *text* для ввода следующих предложений. Если *count_0,* который считает количество введенных предложений, равен *size_arr*, значит мы должны «расширить» память, выделяемую под наш двумерный массив. Для этого мы увеличим *size_arr*=2* и используем функцию *text=realloc(text,size_arr*sizeof(char*)).*

18. После ввода терминального предложения и записи его в наш массив, мы выходим из цикла, таким образом, записав все введенные предложения. При этом в count_0 у нас посчиталось и терминальное предложение, и прибавилась к этому числу 1. Запомним это для следующих действий.

19. Ввод завершен, табуляции в начале предложений убраны. Можем приступать к удалению предложений, не удовлетворяющих условию.

20. К предложениям будем обращаться, как к элементам массива указателей на указатели *text[i]*, где *i* – счетчик цикла *for*. Будем перебирать все *i* = [0;count_0).

21. На каждой итерации цикла будем вызывать функцию *delete_sent(text[i]),* которую мы опишем отдельно позже. Если функция возвращает единицу *if(delete_sent(text[i]))*, то будем выводить наше предложение *printf(“%s\n”, text[i])* и заодно считать количество предложений после обработки, т.е. *count_1++*.

22. Последним этапом очищаем всю выделенную на ранних тапах память, но нужно помнить, что очищаем ее в порядке, обратном выделению памяти.

Значит, сначала очищаем с помощью *for(i=0;i<count_0;i++)free(text[i]);* Т.е. строки нашего текста, затем очищаем массив указателей на них, то есть в функцию *free()* передаем адрес первого элемента этого массива, или *free(text);*

Таким образом, вся выделенная ранее память очищена, «утечка памяти» исключена.

Программа завершает свою работу.

Опишем подробнее написанную нами функцию *int delete_sent(char* sent):*

Тип функции: функция будет возвращать 0 или 1, поэтому тип *int.*

Название функции и описание: delete_sent() – функция будет обрабатывать предложение, расположенное по передаваемому адресу. Если внутри слов этого предложения есть цифры(подробнее в условии задачи), то программа возвращает 0, если предложение удовлетворяет условию, то возвращает 1.

Аргументы функции: в функции *main()* передается адрес *text[i],* по которому хранится искомое предложение, поэтому аргумент назовем *char* sent.*

Ход работы функции *int delete_sent(char* sent)*:

1. Инициализируем необходимые переменные, такие как int i=0(будем передвигаться по предложению, с помощью этого индекса), int ind_start=0 – сюда будем запоминать индекс начала слова, int ind_end=0 – сюда запоминаем индекс конца слова.

2. Изначально индекс начала слова равен нулю. Принцип будет таким, ищем пробел/символ окончания в предложении, чтобы выделить первое слово. Пока они не найдены, *i++*. Как только индекс пробела/символ конца предложения найден, можем определить индекс конца слова. Ведь индекс конца слова *i-1(*последний символ слова перед пробелом/перед символом окончания предложения) или *i-2*(если перед пробелом запятая).

3. Инициализируем переменные *int j=0, check_dgt=0, check_alpha=0.* Последние два будут считать количество цифр и букв «внутри» слова соответственно.

4. С помощью цикла будем перебирать символы слова, расположенные внутри, то есть [ind_start+1;ind_end)

f*or(j=ind_start+1;j<ind_end;j++)*

5. На каждой итерации проверяем, если символ буква (с помощью *if(isalpha(*(sent+j)))check_alpha++;),* если цифра, то check_dgt++.

6. После всех итераций проверяем:

1) Если есть и цифры, и буквы, значит внутри слова цифры –> не подходит.

if(check_dgt>0&&check_alpha>0) return 0;

2) Если есть только цифры, то на крайних позициях должна быть хотя бы 1 буква, чтобы это являлось словом, внутри которого цифры, поэтому проверяем крайние буквы, если хотя бы одна из них буква, то предложение нам не подходит, возвращаем 0:

```
if(check_dgt>0&&check_alpha==0){
        if(isalpha(*(sent+ind_start))==1||isalpha(*(sent+ind_end))==1)
            return 0;
}
```

7. Дальше нам надо перейти к следующему слову, но только в том случае, если оно там есть, т.е. если *i* не равен последнему символу предложения, то запоминаем индекс начала уже нового слова, которое будем обрабатывать на следующей итерации *ind_start=i+1* и увеличиваем *i++*.

8. Если предложение прошло все итерации цикла, значит, все его слова удовлетворяют условию задачи. Значит, и предложение удовлетворяет условию задачи. Следовательно, возвращаем 1, чтобы потом в функции *main()* это предложение вывелось на экран, как удовлетворяющее условию.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Fusce finibus sapien magna, quis scelerisque ex sodales tristique. Ut auctor augue vel tincidunt tincidunt 555. Fusce finibus sapien magna, quis scelerisque ex sodales tristique. Ut auctor augue vel tincidunt tincidunt 555. Aliquam 555 condimentum ligula arcu, non mollis ex pell555entesque finibus. Dragon flew away!	Fusce finibus sapien magna, quis scelerisque ex sodales tristique. Ut auctor augue vel tincidunt tincidunt 555. Fusce finibus sapien magna, quis scelerisque ex sodales tristique. Ut auctor augue vel tincidunt tincidunt 555. Dragon flew away! Количество предложений до 5 и количество предложений после 4	Верно, каждое предложение – с новой строки, табуляция в начале предложений удалена, а также удалены предложения, внутри слов которых находятся цифры(подробнее в условии задачи).
2.	Gjkds ghjdjk Hjdkl; hf45dhj dhjsj? dhsjk 7dksk. Dragon flew away!	Gjkds ghjdjk Hjdkl; dhsjk 7dksk. Dragon flew away! Количество предложений до 3 и количество предложений после 2	Верно, каждое предложение – с новой строки, табуляция в начале предложений удалена, а также удалены предложения, внутри слов которых находятся цифры(подробнее в условии задачи).

Выводы.

Были изучены принципы работы с указателями в си, а полученные знания были применены в написании программного кода.

Разработана программа, которая принимает на вход текст, записывает его в двумерный динамический массив, форматирует согласно требованиям, выводит результат на консоль и память, выделенную под динамический массив, освобождает, чтобы исключить так называемую «утечку памяти».

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define SIZE_ARR 10
#define BUFFER 50
int delete_sent(char* sent){
    int i=0;
    int ind_start=0,ind_end=0;//индексы начала и конца слова
    while(i<strlen(sent)-1){
        while(*(sent+i)!='
'&&*(sent+i)!='.'&&*(sent+i)!=';'&&*(sent+i)!='?'&&*(sent+i)!='!')i++;
        if(*(sent+i)==' '){
                if(*(sent+i-1)!=',')ind_end=i-1;
                else ind_end=i-2;
        }
        else ind_end=i-1;
        int j,check_dgt=0,check_alpha=0;
        for(j=ind_start+1;j<ind_end;j++){
            if(isdigit(*(sent+j))check_dgt++;
            if(isalpha(*(sent+j))check_alpha++;
        }
        if(check_dgt>0&&check_alpha>0) return 0;
        if(check_dgt>0&&check_alpha==0){

if(isalpha(*(sent+ind_start))==1||isalpha(*(sent+ind_end))==1)
                return 0;
        }
        if(i!=strlen(sent)-1){
            ind_start=i+1;
            i++;
        }
    }
    return 1;
}

int main()
{
    int size_arr=SIZE_ARR;
    int buffer=BUFFER;
    char c;
    int length;
    int count_0=0,count_1=0;
    char** text=malloc(size_arr*sizeof(char*));
    int flag=1,i;
    while(flag==1){
        length=0;
        c=getchar();
        if(c=='\n'||c==' '||c=='\t') c=getchar();
        if(c=='\n'||c=='\t')c=getchar();
        char* buf=calloc(buffer,sizeof(char));
        while(c!='.'&&c!=';'&&c!='?'&&c!='!'){
```

```
            buf[length++]=c;
            if(length==buffer){
                buffer*=2;
                buf=realloc(buf,buffer);
            }
            c=getchar();
        }
        buf[length]=c;
        buf[length+1]='\0';
        text[count_0]=buf;
        if(strcmp(buf,"Dragon flew away!\0")==0)
            flag=0;
        count_0++;
        if(count_0==size_arr){
            size_arr*=2;
            text=realloc(text, size_arr*sizeof(char*));
        }
    }
    for(i=0;i<count_0;i++){
        if(delete_sent(text[i])){
                printf("%s\n", text[i]);
                count_1++;
        }
    }
    for(i=0;i<count_0;i++)free(text[i]);
    free(text);
    count_0-=1;
    count_1-=1;
    printf("Количество предложений до %d и количество предложений
после %d\n",count_0,count_1);
    return 0;
}
```