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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Программирование»
Тема: Использование указателей.

Студент гр. 0382 Санников В.А.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Научиться работать с указателями и динамической памятью.

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 больше одной заглавной буквы, 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 некоторая переменная, значением которой является адрес в памяти некоторого объекта, определяемого типом указателя. Для работы с указателями используется 2 оператора:

* - оператор разыменования.

& - оператор взятия адреса.

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, конструкции вида &(x-1) и &3 являются незаконными. Формально, в языке Си нет специального типа данных для строк, но представление их довольно естественно - строки в языке Си это массивы символов, завершающиеся нулевым символом ('\0'). Это порождает следующие особенности, которые следует помнить:

•Нулевой символ является обязательным.

•Символы, расположенные в массиве после первого нулевого символа никак не интерпретируются и считаются мусором.

•Отсутствие нулевого символа может привести к выходу за границу массива.

•Фактический размер массива должен быть на единицу больше количества символов в строке (для хранения нулевого символа)

•Выполняя операции над строками, нужно учитывать размер массива, выделенный под хранение строки. Строки могут быть инициализированы при объявлении. char*fgets(char *str, int num, FILE *stream)

•Безопасный способ (явно указывается размер буфера)

•Считывает до символа переноса строки

•Помещает символ переноса строки в строку-буфер (!) int scanf(const char * format, arg1, arg2, ...argN);

•%s в форматной строке для ввода строки

•Считывает символы до первого символа табуляции (не помещая его в строку)

•Не контролирует размер буфера

•Потенциально опасна char* gets(char* str);

•Не контролирует размер буфера

•Потенциально опасна

Как вы уже могли догадаться, если строка в Си - массив символов, то массив строк это двумерный массив символов, где каждая строка - массив, хранящий очередную символьную строку.

Статический массив строк может быть также инициализирован при объявлении

Примеры:// статический массив строк

```
char stat_strs[][LEN]={"Hello",
                       "It's string array",
                       "of three rows" };
// создание динамического массива строк
char **dyn_strs = malloc(N * sizeof(char*));
for(i=0;i<N;i++){
                  dyn_strs[i] = calloc(LEN, sizeof(char));
                  strncpy(dyn_strs[i], stat_strs[i], LEN - 1);
                 }
// освобождение памяти
for(i=0;i<N;i++)
      free(dyn_strs[i]);
free(dyn_strs);
```

Выполнение работы.

Ход работы: В программе Используется стандартная библиотека языка си, её заголовочные файлы stdio.h, stdlib.h, string.h для работы со строками и ctype.h для работы с символами.

В функции main() - объявляется массив strs и выделяется по него память. Затем выполняется функция print_text() для strs. Выводится количество до обработки и после, очищается память (чтобы не перегружать систему) с помощью функции free().

Функция print_text() - вводит предложения с помощью функции print_string(). Выделяется память с помощью realloc, если ее не хватает. Далее происходит подсчет количества всех предложений и предложений, в которых меньше 2 букв верхнего регистра. Если вводится предложение «Dragon flew away!», то цикл завершается.

print_string() - получает предложения, благодаря функции getchar(). Память под массив символов увеличивается в том случае, если ее не хватает (realloc). Если обнаруживается символ верхнего регистра, то увеличивается переменная sum_upper. Функция не обрабатывает символы «\t» и «\n». Ввод завершается если встречены символы: «.», «;», «?», «!».

s_index — количество символов в динамическом массиве символов.

strs_index — количество предложений в динамическом массиве символов.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Privet ya begemot! U menay est NOGA? A vse ravno ved. Dragon flew away!	Privet ya begemot! A vse ravno ved. Dragon flew away!	Программа работает верно

		3 2	
2.	Ya poel borsha! Eto bilo nezabivaemo, I've read some magic books. They told asa S BFBRSDHtfb. and? Dragon flew away!	Ya poel borsha! and? Dragon flew away! 4 2	Программа работает верно

Выводы.

Была завершена и освоена работа с указателями и динамической памятью. Разработана программа выполняющая следующий алгоритм: считать с клавиатуры текст → с помощью функций и библиотек обработать символы в предложениях и сами предложения → вывести количество предложений до обработки и после.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

char* print_string(char *s);
char** print_text(char **strs);

int n = 0; //до обработки
int m = 0; //после обработки
int sum_upper = 0;
char c;
int strs_index = 0;

char** print_text(char **strs){  //предложения
    int strsize = 10;
    while(1){
        if(strs_index==strsize){
            strsize+=10;
            strs = realloc(strs, strsize*sizeof(char*));
        }
        char* s = malloc(40*sizeof(char));
        s = print_string(s);
        if (s[0]==' ')
            s++;
        n++;
        if (sum_upper<=1){
```

```
            strs[strs_index++]=s;
            m++;
        }
        if (!strcmp(s, "Dragon flew away!")) {
            break;
        }
    }
    return strs;
}

char* print_string(char *s){  //символы
    int s_size = 40;
    int s_index = 0;
    sum_upper = 0;
    while(1){
        if(s_index==s_size-1){
            s_size+=40;
            s = realloc(s, s_size*sizeof(char));

        }
        c = getchar();
        if (isupper(c))
            sum_upper++;
        if (c!='\t' && c!='\n'){
            if (s!="" || c!=' '){
                s[s_index++]=c;
            }
        }
        if (c=='.' || c==';' || c=='?'){
            c=getchar();
```

```
            while (c=='\n')
                c = getchar();
            s[s_index]='\0';
            break;
        }
        if(c=='!'){
            s[s_index]='\0';
            break;
        }
    }
    return s;
}

int main() {
    char **strs = malloc(10*sizeof(char*));
    strs = print_text(strs);
    for (int i = 0; i<strs_index; i++){
        printf("%s\n", strs[i]);
        free(strs[i]);
    }
    free(strs);
    printf("Количество предложений до %d и количество предложений после %d", n-1, m-1);
    return 0;
}
```