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

ОТЧЕТ

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

Студент гр. 1304 _____ Байков Е.С.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2021

Цель работы

Освоить работу с указателями и динамической памятью.

Задание

Вариант 2

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, которые заканчиваются на '?' 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**

Выполнение работы

Подключение заголовочных файлов *string.h, stdio.h, stdlib.h.* Введение макросов *MEMORY_STEP* и *END_SENTENCE*, равные 20 и строке, являющейся окончанием ввода по условию, соответственно. Описание вспомогательных функций *delTabs* – удаляет пробелы, переносы строки и

знаки табуляции и возвращает знак неравный предыдущим, *readSentnence* – функция содержит внутри несколько локальных переменных: *size* – размер буфера, *ind_ch* – индекс текущего символа, *sentence* – массив символов (указатель на первый элемент данного массива), *t* – временная переменная которая будит хранить в себе адрес расширенного массива *sentence* или *NULL*, а также *temp* – временная переменная, которая будет хранить в себе временный символ. Возвращает указатель на первый элемент массива символов (строка). *readText* – функция на вход принимает указатель на двумерный массив *text* и строку *end_sentence.* Содержит внутри себя несколько локальных переменных: *size* – размер буфера, *i* – индекс элемента в *text, sentence* – массив символов, который будет хранить в себе текущее предложение, *t* – временная переменная для хранения адреса расширенного массива **text*. Возвращает *i*. *deleteSentence* – функция принимает на вход двумерный массив *text* и длину данного массива *len*. Внутри функции находится переменная *sentence,* которая принимает значение первого элемента в массиве *text.* Возвращает *len*.

В функции *main* объявляются переменные *char** text, int len_text, new_len*. В переменную *len_text* передается значение функции *readText*, которой на вход переданы адрес *text* и макрос *END_SENTENCE*. Внутри данной функции переменной *size* присваивается макрос *MEMORY_STEP*. Для переменной *t* типа *char** динамически выделяется память и в случае если в *t* передается *NULL* то функция возвращает 0**.** В (**text*) передается значение *t*, объявляется переменная *i*, которая хранит значение 0, и переменная *char* sentence*. Затем с помощью цикла *do while* (до тех пор пока *strcmp(sentence, end_sentence)* не вернет 0) совершаются следующие действия: проверяется достаточно ли памяти выделено (сравнение значения текущего индекса(*i*) и значения *size*). Если вдруг памяти не хватает, значение *size* увеличивается на *MEMORY_STEP,* в переменную *t* передается адрес расширенной памяти для массива *text* и в случае если *t != NULL,* то в **text* передается значение *t,* иначе отчищается помять под каждый элемент в массиве (**text*), а также память выделенная для самого массива. После в переменную *sentence* передается

значение функции *readSentence()*. Если в *sentence* передается *NULL* то с помощью цикла очищается память, отведенная под каждый элемент в массиве (**text*), а потом и память, отведенная под массив, функция возвращает 0. Если *sentence != NULL*, то в элемент *(*text)[i++]* передастся значение *sentence*. В итоге функция возвращает *i*.

В самой функции readSentence() посимвольно считывается предложение, до тех пор, пока не встретится один из знаков окончания предложения, а в качестве первого символа строки передается значение *delTabs()*. Динамически выделяется память под несколько знаков, которые поступают из стандартного потока ввода с помощью функции *getchar()*. Как только индекс текущего символа *ind_ch* >= *size – 2*. После завершения цикла *do while* в строчку *sentence* добавляются два символа – перенос строки и символ окончания строки, а потом функция возвращает строку.

В функции *main* в переменную *new_len* передается значение *len_text*, а затем в *new_len* передается значение функции *deleteSentence(),* которой на вход подается *text* и *new_len*. Внутри функции *deleteSentence()* в переменную *char** *sentence* передается значение *text*. С помощью цикла while до тех пор, пока не пройдет по всем предложениям данного текста, функция находит предложения, в которых находится вопросительный знак с помощью функции *strchr()*. В случае, когда функция находит такое предложение то память, выделенная под него, освобождается, длинна всего текста уменьшается на 1, а все элементы, идущие после удаленного, передвигаются на индекс меньше их индекса на единицу (в массиве) с помощью функции *memmove*. В противном случае функция переходит к следующему в тексте предложению. Возвращает измененную длину.

В функции *main* с помощью цикла for выводится текст и очищает память, отведенную под предложения. Затем печатает строку Количество предложений до %d и количество предложений после %d\n, куда передаются значения *len_text* и *new_len* со значением на 1 меньше. Освобождается память выделенная под *text*. Возвращается 0.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Sometimes we need to delete. HOW? Sentence has deleted. Dragon flew away!	Sometimes we need to delete. Sentence has deleted. Dragon flew away! Количество предложений до 3 и количество предложений после 2	
2.	Dragon flew away!	Dragon flew away! Количество предложений до 0 и количество предложений после 0	
3.	HOW? Delete? Dasds? NO. Desdwadas. asdas; WWW? Dragon flew away!	NO. Desdwadas. asdas; Dragon flew away! Количество предложений до 7 и количество предложений после 3	

Выводы

Были изучены основы работы с указателями и динамической памятью в языке С.

Разработана программа, выполняющая запись введенного текста в динамическую память, обработку в соответствии с заданным условием и вывод результата на экран.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Baykov_Egor_lb3.c

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

#define MEMORY_STEP 20
#define END_SENTENCE "Dragon flew away!\n"

char delTabs(){
    char c = getchar();
    for(c; c == ' ' || c == '\n' || c == '\t'; c = getchar()){}
    return c;
}

char* readSentence(){
    int size = MEMORY_STEP;
    char* t = malloc(size * sizeof(char));
    char temp;

    if(t != NULL){
        int ind_ch = 0;
        char* sentence = t;
        sentence[ind_ch++] = delTabs();
            do{
                if (ind_ch >= size - 2){
                    size += MEMORY_STEP;
                    t = realloc(sentence, size * sizeof(char));
                    if(t != NULL){
                        sentence = t;
                    }
                    else{
                        free(sentence);
                        return NULL;
                    }
                }
                temp = getchar();
                sentence[ind_ch++] = temp;
            } while(sentence[ind_ch-1] != '.' && sentence[ind_ch-1] !=
';' && sentence[ind_ch-1] != '?' && sentence[ind_ch-1] != '!');
        sentence[ind_ch++] = '\n';
        sentence[ind_ch] = '\0';
        return sentence;
    }
    else{
        return NULL;
    }
}

int readText(char*** text, char* end_sentence){
    int size = MEMORY_STEP;
    char** t = malloc(size * sizeof(char*));
```

```
    if (t != NULL){

        (*text) = t;
        int i = 0;
        char* sentence;

        do{
            if(i == size){
                size += MEMORY_STEP;
                t = realloc((*text), size * sizeof(char*));
                if (t != NULL){
                    (*text) = t;
                }
                else{
                    for(int j = 0; j < i; j++){
                        free((*text)[j]);
                    }
                    free(*text);
                    return 0;
                }
            }
            sentence = readSentence();
            if(sentence != NULL){
                (*text)[i++] = sentence;
            }
            else{
                for(int j = 0; j < i; j++){
                    free((*text)[j]);
                }
                free(*text);
                return 0;
            }
        } while(strcmp(sentence, end_sentence));

        return i;
    }
    else{
        return 0;
    }
}

int deleteSentence(char** text, int len){
    char** sentence = text;
    while(sentence < text + len){
        if(strchr(*sentence, '?')){
            free(*sentence);
            len--;
            memmove(sentence, sentence + 1, sizeof(char*) * (text + len
- sentence));
        }
        else{
            sentence++;
        }
    }
    return len;
}
```

```
int main(){
    char** text;
    int len_text, new_len;
    len_text = readText(&text, END_SENTENCE);
    new_len = len_text;
    new_len = deleteSentence(text, new_len);
    for(char** sentence = text; sentence < text + new_len; sentence++){
        printf("%s", *sentence);
        free(*sentence);
    }
    printf("Количество предложений до %d и количество предложений после
%d\n", len_text - 1, new_len - 1);
    free(text);
    return 0;
}
```