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. 1304 _____ Шаврин А.П

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2021

Цель работы.

Научиться использовать указатели для решения различных задач, в частности для обработки текста. Познакомиться с динамической памятью компьютера, а также с динамическими символьными массивами. Практика работы с памятью в Си.

Основные теоретические положения.

1. *malloc(int *size)* - Выделение динамической памяти размера *size* байт.
2. *realloc(char *temp,int size)* - Увелечение или уменьшение памяти массива *temp* на *size* байт.
3. * - Разыменование элемента.
4. & - Получение адреса.
5. *strlen(char * arr)* - Вычисление размера массива *arr*.
6. *free(arr)* - освобождение памяти массива элементов.

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, 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.

1. Функция *char get_first_symbol()* возвращает первый символ предложения, не включая табуляции и т.п.
2. Функция *char *read_sentence()* возвращает указатель на считаное предложение.
3. Функция *int get_len_text(char ***text)* принимает адрес указателя на текст, записывает массив указателей(на предложения) и возвращает длину текста(кол-во предложений).
4. Функция *void get_new_text(char ***text, int *len_text)* принимает адрес указателя на текст и кол-во предложений в тексте и удаляет все предложения оканчивающиеся знаком “?”.
5. Функция *void free_text(char ***text, int len_text)* принимает адрес указателя на текст и кол-во предложений в тексте и очищает память выделенную под каждое предложение.
6. Функция *void print_new_text(char ***text, int len_text, int len_new_text)* принимает адрес указателя на текст, длину исходного текста и длину измененного текста. Выводит измененный текст и добавляет предложение с информацией о кол-ве предложений начального текста и измененного.
7. Функция *int main()* реализует программу.

Каждая функция вынесена в отдельный файл с названием соответствующим названию функции с расширением .c. У каждого файла функции есть заголовочный файл. Создан *Makefile* в котором реализована сборка программы.

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.	Adfghj fghjk fghjk. FDFGHJgh fghj? Dragon flew away!	Adfghj fghjk fghjk. Dragon flew away! Количество предложений до 2 и количество предложений после 1.	Было удалено вопросительное предложение.
2.	DFGHhjkl fghjk 5678. Dragon flew away!	DFGHhjkl fghjk 5678. Dragon flew away! Количество предложений до 1 и количество предложений после 1	Ничего не удалилось, первое предложение удовлетворяет условию задачи.
3.	FGHJ fghj! ? Dragon flew away!	FGHJ fghj! Dragon flew away! Количество предложений до 2 и количество предложений после 1	Было удалено вопросительное предложение.
4.	Dragon flew away!	Dragon flew away! Количество предложений до 0 и количество предложений после 0	Ничего не изменилось.

Выводы.

Были изучены основы работы с указателями, динамической памятью, символьными массивами. Также была разработана программа по обработке символьного массива.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: *main.c*

```
#include <stdio.h>
#include "get_len_text.h"
#include "get_new_text.h"
#include "print_new_text.h"
#include "free_text.h"

int main() {
        char **text;
int len_text = get_len_text(&text);
        int len_new_text = len_text;
        get_new_text(&text, &len_new_text);
        print_new_text(&text, len_text, len_new_text);
        free_text(&text, len_new_text);
         return 0;
}
```

Название файла: *get_first_symbol.c*

```
#include <stdio.h>

char get_first_symbol(){
        char symbol;
        for (symbol = getchar(); symbol == ' ' || symbol == '\t' || symbol == '\n';
symbol = getchar()){}
        return symbol;
}
```

Название файла: *get_first_symbol.h*

```
char get_first_symbol();
```

Название файла: *read_sentence.c*

```
#include <stdio.h>
#include <stdlib.h>
#include "get_first_symbol.h"
#define LENGTH_SENT 50

char *read_sentence(){
    int len_sent = LENGTH_SENT;
    char *temp = malloc(len_sent*sizeof(char));

    if (temp != NULL){
        char *sentence = temp;
        char symbol = get_first_symbol();
        int i;
          for (i = 0; symbol != '.' && symbol != ';' && symbol != '?' &&
symbol != '!'; symbol = getchar()){
            sentence[i++] = symbol;
            if (i >= (len_sent - 2)){
                len_sent += LENGTH_SENT;
                temp = realloc(sentence, len_sent*sizeof(char));
                if (temp != NULL){
                    sentence = temp;
                }
                else{
                    free(sentence);
                    return NULL;
                }
            }
        }
        sentence[i++] = symbol;
```

```
            sentence[i++] = '\n';
            sentence[i] = '\0';
            return sentence;
        }
        return NULL;
    }
```

Название файла: *read_sentence.h

```
char *read_sentence();
```

Название файла: get_len_text.c

```
#include <stdlib.h>
#include <string.h>
#include "read_sentence.h"
#include "free_text.h"
#define LENGTH_TEXT 10
#define LAST_SENT "Dragon flew away!\n"

int get_len_text(char ***text){
    int len_text = LENGTH_TEXT;
    char **temp = malloc(len_text*sizeof(char*));

    if (temp != NULL){
        *text = temp;
        char *sentence;
        int i = 0;
         for (sentence = read_sentence(); strcmp(sentence, LAST_SENT) != 0; sentence = read_sentence()){
            if (sentence == NULL){
                free_text(text, i);
                return 0;
```

```
                }
                (*text)[i++] = sentence;
                if (i >= len_text){
                        len_text += LENGTH_TEXT;
                        temp = realloc(temp, len_text*sizeof(char*));
                        if (temp != NULL){
                                (*text) = temp;
                        }
                        else{
                                free_text(text, i);
                                return 0;
                        }
                }
        }
        (*text)[i++] = sentence;
        return i;
    }
    return 0;
}
```

Название файла: *get_len_text.h*

```
int get_len_text(char ***text);
```

Название файла: *get_new_text.c*

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

void get_new_text(char ***text, int *len_text){
    char **temp = malloc(*len_text*sizeof(char*));
    if (temp != NULL){
        char **new_text = temp;
```

```
            int n = 0;
            for (int i = 0; i < *len_text; i++){
                int len_sent = strlen((*text)[i]);
                if ((*text)[i][len_sent-2] != '?'){
                    new_text[n++] = (*text)[i];
                }
                else{
                    free((*text)[i]);
                }
            }
            free(*text);
            *text = new_text;
            *len_text = n;
        }
}
```

Название файла: *get_new_text.h*

```
void get_new_text(char ***text, int *len_text);
```

Название файла: *free_text.c*

```
#include <stdlib.h>

void free_text(char ***text, int len_text){
    for (int i = 0; i < len_text; i++){
        free((*text)[i]);
    }
    free(*text);
}
```

Название файла: *free_text.h*

*void free_text(char ***text, int len_text);*

Название файла: *print_new_text.c*

Vim #include <stdio.h>

```
void print_new_text(char ***text, int len_text, int len_new_text){
    for (int i = 0; i < len_new_text; i++){
        printf("%s", (*text)[i]);
    }
      printf("Количество предложений до %d и количество предложений после %d", len_text-1, len_new_text-1);
}
```

Название файла print_new_text.*h*

*void print_new_text(char ***text, int len_text, int len_new_text);*

Название файла *Makefile*

```
CC = gcc

all: main.o get_first_symbol.o read_sentence.o get_len_text.o get_new_text.o print_new_text.o free_text.o
        $(CC) main.o get_first_symbol.o read_sentence.o get_len_text.o get_new_text.o print_new_text.o free_text.o -o main

main.o: main.c get_len_text.h get_new_text.h print_new_text.h free_text.h
    $(CC) -c main.c

get_first_symbol.o: get_first_symbol.c
    $(CC) -c get_first_symbol.c

read_sentence.o: read_sentence.c get_first_symbol.h
```

```
	$(CC) -c read_sentence.c

get_len_text.o: get_len_text.c read_sentence.h free_text.h
	$(CC) -c get_len_text.c

get_new_text.o: get_new_text.c
	$(CC) -c get_new_text.c

print_new_text.o: print_new_text.c
	$(CC) -c print_new_text.c

free_text.o: free_text.c
	$(CC) -c free_text.c

clean:
	rm -f *.o main
```