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 _____ Ефремов А.А.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2021

Цель работы.

Изучить основы работы с указателями, динамической памятью, написать программу, которая форматирует некоторый текст и выводит результат на консоль.

Задание.

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.

В процессе разработки программы были использованы следующие функци:

int get_size(char*** text) — функция выделяет память под хранение текста, считывает его и возвращает количество предложений.

int new_size(char*** text, int size) — функция удаляет из текста все вопросительные предложения и возвращает новое количество предложений.

char* get_sentence() — функция считывает предложение из текста, введенного в консоль.

char last_ch(char* sentence) — функция возвращает последний символ предложения (исключая «\n» и «\0»).

void free_text(char*** text, int size) — функция очищает выделенную память.

Переменные функции main:

text — массив типа char**, в котором хранится введенный текст.

amount_of_sentences — переменная типа int, которая хранит в себе количество предложений исходного текста без учета терминального предложения.

new_amount_of_sentences — переменная типа int, которая хранит в себе количество предложений после обработки текста.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 2.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии

1	It. Works? Dragon flew away!	It. Dragon flew away! Количество предложений до 2 и количество предложений после 1	Успешно
2	It. Works; Dragon flew away!	It. Works; Dragon flew away! Количество предложений до 2 и количество предложений после 2	Успешно

Выводы.

Изучены основы работы с указателями, динамической памятью, была реализована программа, которая принимает некоторый текст из консольного ввода, форматирует его и выводит результат на консоль. В процессе работы программы текст хранился в динамической памяти в виде массива символов. Для работы с массивами были использованы указатели. После вывода текста на консоль была проведена очистка динамической памяти.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: pr_lb3.c

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

int get_size(char*** text);
int new_size(char*** text, int size);
char* get_sentence();
char last_ch(char* sentence);
void free_text(char*** text, int size);

int main() {
    setlocale(LC_ALL, "ru");
    char** text;
    int amount_of_sentences = get_size(&text);
   int new_amount_of_sentences = new_size(&text, amount_of_sentences);
    for (int i = 0; i < new_amount_of_sentences; i++)
        printf("%s", text[i]);
    printf("Количество предложений до %i и количество предложений
после %i\n", amount_of_sentences, new_amount_of_sentences-1);
    free_text(&text, amount_of_sentences);
    return 0;
}

int get_size(char*** text) {
    int l = 50;
    char** t = (char**) malloc(l*sizeof(char*));
    if (t != NULL) {
        *text = t;
        char* sentence;
        int amount_of_sentences = 0;
        const char* last_sentence = "Dragon flew away!\n";
       for (sentence = get_sentence(); strcmp(sentence, last_sentence)
!= 0; sentence = get_sentence()) {
            if (sentence == NULL) {
                free_text(text, amount_of_sentences);
                return 0;
            }
            (*text)[amount_of_sentences++] = sentence;
            if (amount_of_sentences == l - 1) {
                l += l;
                t = (char**) realloc(t, l*sizeof(char*));
                if (t != NULL)
                    *text = t;
                else {
                    free_text(text, amount_of_sentences);
                    return 0;
                }
```

```
            }
        }
        (*text)[amount_of_sentences] = sentence;
        return amount_of_sentences;
    }
    return 0;
}

char* get_sentence() {
    int l = 50;
    char* t = (char*) malloc(l*sizeof(char));
    if (t != NULL) {
        char* sentence = t;
        char ch;
        int i = 0;
        do {
            ch = getchar();
        } while ((ch == ' ') || (ch == '\n') || (ch == '\t'));
        while ((ch != '.') && (ch != ';') && (ch != '?') && (ch !=
'!')) {
            if (i == l - 3) {
                l += l;
                t = (char*)realloc(t, l * sizeof(char));
                if (t != NULL)
                    sentence = t;
                else {
                    free(sentence);
                    return NULL;
                }
            }
            sentence[i++] = ch;
            ch = getchar();
        }
        sentence[i++] = ch;
        sentence[i++] = '\n';
        sentence[i] = '\0';
        return sentence;
    }
    return NULL;
}

int new_size(char*** text, int size) {
    char** t = (char**) malloc(size * sizeof(char*));
    if (t != NULL) {
        char** new_text = t;
        int new_amount_of_sentences = 0;
        for (int i = 0; i < size+1; i++) {
            if (last_ch((*text)[i]) != '?')
                new_text[new_amount_of_sentences++] = (*text)[i];
            else
                free((*text)[i]);
        }
        free(*text);
        *text = new_text;
        return new_amount_of_sentences;
    }
```

```
    return 0;
}

char last_ch(char* sentence) {
    char last_ch = sentence[strlen(sentence) - 2];
    return last_ch;
}

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