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

ОТЧЕТ

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

Студент гр. 0382 _____ Павлов С.Р.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Целью работы является освоение работы 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, 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

ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ.

Указатель – 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.

Функция *malloc* (для ее использования следует подключить заголовочный файл stdlib.h): *void * malloc(size_t sizemem)*.

Функция выделяет из памяти sizemem и возвращает указатель на выделенную память, который следует привести к требуемому типу.

Для изменения размера выделенной ранее динамически области памяти используется функция *realloc*: void * realloc(void * ptrmem, size_t size) Она получает указатель на выделенную ранее область памяти и новый ее размер (он может быть как увеличен, так и уменьшен) и возвращает указатель на область памяти измененного размера (при изменении размера области памяти ее начальный адрес может измениться).

Следовательно, функция *realloc* может выполняться в некоторых случаях довольно долго, поэтому не следует использовать ее слишком часто без явной необходимости.

Если выделенная память больше не требуется, следует обязательно высвобождать ее с помощью оператора *free*.

Формально в языке С нет специального типа данных для строк, но представление их довольно естественно.

Строки в языке С – 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.

Выполнение работы.

В начале программы используются необходимые для ее выполнения, заголовочные файлы. *stdio.h , stdlib.h , string.h.*

Описание используемых функций:

1. Функция *char* get_sent().*

Данная функция получает и собирает символы из потока, в предложения. В ней используется цикл с терминальным условием внутри, что *ch* (символ) не должен быть знаком конца предложения. Так же в данной функции реализована выделение динамической памяти. Изначально указателю *tmp_sent*, выделяется 100 байт памяти типа char, но если кол-во элементов превосходит это кол-во, то с помощью функции *realloc()* , память расширяется на 100 байтов.

2. Функция *char* sentence_edit_1(char* tmp_sent).*

Функция убирает символы пробела, табуляции и перевода строки. И возвращает отредактированое предложение с помощью *return.*

3. Функция *int main().*

Задача главной функции заключается в вызове всех предыдущих функций и выполнении поставленных задач. В начале описания функции инициализируются двумерный массив *text,* обладающий 100 байтами памяти. Затем главный цикл, с терминальным условием если обрабатываемое предложение станет, «Dragons flew away!», то цикл завершится. Эта проверка реализована с помощью функции *strcmp()*. На каждой новой иттерации цикл получает новое предложение, которое записывается в соотв. элемент *text.* Так же для *text* реализовано динамическое расширение памяти. Так же в цикле производится проверка на вопросительный знак в конце предложения (если он есть, то функция *get_sent()* вернет null и цикл передет на след. Иттерацию).

Затем после цикла, идет новый цикл который распечатывает элементы *text,*и так же освобождая память соотв. элемента с помощью функции *free().*

Разработанный программный код см. в приложении А.

Выводы.

Была изучена работа массивов и указателей в языке Си. Разработана программа, которая считывает введенный с клавиатуры текст и записывает его в динамический массив. Также в функциях производится обработка каждого предложения текста, для этого используются указатели и функции для работы с динамической памятью

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

char* get_sent(){
    char ch;
    char *tmp_sent;
    char len = 0;
    int mem_add = 100;
    tmp_sent = malloc(100*sizeof(char));


    while (1){

        ch = getchar();
        tmp_sent[len] = ch;
        len += 1;
        if (ch == '?') return NULL;
        if ((ch == '.')||(ch == ';')||(ch == '?')||(ch == '!')) break;
        if (len > mem_add){
            mem_add += 100;
            tmp_sent = realloc(tmp_sent, mem_add);
        }

    }

    tmp_sent[len] = '\0';
    return tmp_sent;
}



char* sentence_edit_1(char* tmp_sent){
    int i = 0;
    while(tmp_sent[i] == ' ' || tmp_sent[i] == '\t' || tmp_sent[i] ==
'\n')
        {
            int j;
            for (j = 0; j < strlen(tmp_sent) - 1; j ++){
                tmp_sent[j] = tmp_sent[j+1];
        }

    tmp_sent[j] = '\0';

    }
    return tmp_sent;
}
```

```
int main(){

    char stop_sent[] = "Dragon flew away!";
    char **text = malloc(100*sizeof(char*));
    char *sentence;
    int i=0;
    int origin_str_count=0;
    int main_mem_add = 100;

    while (1){

        sentence = get_sent();
        origin_str_count += 1;
        if (sentence != NULL){

            sentence = sentence_edit_1(sentence);

            text[i] = sentence;
            i += 1;

            if (i == main_mem_add){
                main_mem_add += 100;
                text = realloc(text, (main_mem_add*sizeof(char*)));

            }

            if (!strcmp(stop_sent,sentence))
                break;
        }
    }
    for (int k=0;k<i; k++){
        printf("%s\n", text[k]);
        free(text[i]);
    }
    free(text);
    printf("Количество предложений до %d и количество предложений после
%d\n", origin_str_count-1, i-1);

    return 0;
}
```