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: Указатели и массивы

Студент гр.0382 _____ Диденко Д.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Изучение принципов работы с указателя и массивами.

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 (\t, ' ') в начале предложения должна быть удалена. Все предложения, в которых есть число 555, 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 переменная, содержащая адрес другой переменной.

Синтаксис объявления указателя:

<тип_переменной_на_которую_ссылается_указатель> <название переменной>;*

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. В то же время, у указателя есть унарная операция разыменования *, которая позволяет получить значение ячейки, на которую ссылается указатель.

Любую операцию, которую можно выполнить с помощью индексов массива, можно сделать и с помощью указателей.

Пусть у нас объявлен массив из 10 элементов типа int:

int array[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

и указатель на int:

int p_array;*

Тогда запись вида:

p_array = &array[0];

означает, что *p_array* указывает на нулевой элемент массива *array*; т.е. *p_array* содержит адрес элемента *array[0]*. Если *p_array* указывает на некоторый определенный (*array[i])* элемент массива *array*, то *p_array+1* указывает на следующий элемент после *p_array*, т.е. на *array[i+1]*. Тогда как *p_array-1* - на предыдущий *(array[i-1]).*

Память можно представить в виде некоторой ленты идущих друг за другом пронумерованных ячеек. Если с одномерным массивом кажется очевидным как он хранится в памяти, то работа с многомерными массивами может вызывать сложности.

Можно смотреть на двумерный массив (хотя это справедливо и для многомерных) как на массив массивов. Это означает, что массив int arr[4][3] - это массив из 4 элементов, каждый из которых является тоже массивом из трех элементов. Аналогично хранятся и массивы большей размерности. Так, массив arr[n][m][k] - это массив из n элементов, каждый из которых тоже массив из m элементов, элементы которого - массивы длины k.

Если указатели p и q указывают на элементы одного массива, то к ним можно применять операторы отношения ==, !=, <, >= и т. д. Например, отношение вида p < q истинно, если p указывает на более ранний элемент массива, чем q. Любой указатель всегда можно сравнить на равенство и

неравенство с нулем. А вот для указателей, не указывающих на элементы одного массива, результат арифметических операций или сравнений не определен.

Могут возникнуть ситуации, когда объем требуемой памяти в момент написания программы неизвестен и требуется, чтобы выделенная память существовала вне функций, в которых она выделена (вспомним про область жизни локальных переменных) или требуется выделить очень большой объем памяти.

Важно помнить, что в программах на языке С нет механизма, автоматически освобождающего динамически выделенную память, поэтому такая задача ложится на плечи программиста. Если забыть про это, то объем доступной свободной памяти будет уменьшаться (так как выделенная ранее память освобождена не будет) и это может привести к исчерпанию всей доступной памяти. Такие ситуации называются "утечками памяти".

Для работы с динамической памятью используются следующие функции:

- malloc (void* malloc (size_t size)) - выделяет блок из size байт и возвращает указатель на начало этого блока
- calloc (void* calloc (size_t num, size_t size)) - выделяет блок для num элементов, каждый из которых занимает size байт и инициализирует все биты выделеного блока нулями
- realloc (void* realloc (void* ptr, size_t size)) - изменяет размер ранее выделенной области памяти на которую ссылается указатель ptr. Возвращает указатель на область памяти, измененного размера.
- free (void free (void* ptr)) - высвобождает выделенную ранее память.

Технически, динамических двумерных массивов в языке С не предусмотрено. Однако, ничего не мешает нам создать динамический одномерный массив, каждым элементом которого будет являться указатель на другой динамический одномерный массив.

Двумерный массив NxM можно представить в виде N одномерных массивов длинны M. Таким образом, мы имеем N указателей типа int (каждый

их которых ссылается на первый элемент массивов-строк). Для их хранения, используем массив типа int* длины N. Осталось только динамически выделить память под этот массив. Указатель на массив элементов типа int* будет иметь типа int** (указатель на указатель).

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.

Строки считываются с помощью функции fgets():

char* fgets(char *str, int num, FILE *stream)

Преимущества:

- Безопасный способ (явно указывается размер буфера)
- Считывает до символа переноса строки
- Помещает символ переноса строки в строку-буфер (!)

Выполнение работы.

Исходный код решения задачи см.в приложении А.

В главной функции main создается двумерный массив text, которому динамически выделяется память с помощью функции malloc на size_txt элементов типа char*. В теле цикла проверяется условие хватает ли места для нового элемента массива, если нет, то с помощью функции realloc дополнительно выделяется еще 20*typeof(char*) байт памяти. Объявляется массив sent, хранящий вводимые символы и являющийся элементом массива text. Sent получает значение ,обращаясь к функции get_sentence(). В следующем условии проводится проверка, является ли sent валидным предложением (без числа 555) с помощью функции is_sent_valid, если является, то sent добавляется к массиву text, в следующей строке проверяем, равно ли sent стоп-предложению stop_sent с помощью функции is_equal(), если равно, ввод прекращается, если нет - переменные count_sent и count_sent_b увеличиваются на один; если же sent не валидно, то оно не добавляется к text и count_sent_b увеличивается на один (count_sent – считает количество валидных предложений, которые в итоге будут выведены на консоль, count_sent_b считает количество введенных предложений).

После цикла выводим предложения из массива text на консоль, каждое с новой строки.

Очищаем последовательно память с помощью функции free().

Выводим предложение с информацией о количестве введенных предложений и количестве выведенных.

Описание функций:

get_sentence() – ничего не принимает, возвращает массив. В теле функции динамически выделяется память под массив sent размером под size элементов типа char. Объявляются переменные count_let, являющаяся счетчиком вводимых символов, и is_begin, показывающая были ли введены непустые символы. В теле цикла, проверяем хватает ли места в массиве для записи очередного символа, если нет – выделяем в массиве дополнительную память с помощью функции realloc. Новый символ получаем с помощью функции ввода getchar(). Если вновь введенный символ пустой и еще не было значащих символов, то записываем следующий символ на этот же индекс,

таким образом удаляя табуляцию из начала предложения. Далее следует проверка на окончание предложения. После цикла добавляем в конец массива sent символ \0. Возвращаем из функции указатель sent.

is_equal(char* str1, char* str2) – принимает на вход два указателя на начало массива. Объявляется переменная flag ,равная 1, которая изменится на 0, если в цикле while (посимвольно сравнивает элементы массивов) символы массивов окажутся разными. Функция возвращает значение flag.

is_sent_valid(char* sent) – принимает на вход указатель на начало массива. Обработка массива основывается на определении, какие символы стоят по краям у 555. Для определения символов используются коды таблицы ASCII. В теле цикла проверяем: если символ 5, то увеличиваем счетчик пятерок на один(переменная count_5), если символ «непустой»(влияющий на определение вида 555), то счетчик пятерок сбрасывается и начинается следующая итерация цикла, иначе проверяется количество подряд идущих пятярок и если их 3, цикл завершается, и функция возвращает результат – 0(ложь). Исключительная ситуация получается в конце цикла. Поэтому за циклом вводится еще одна проверка на количество пятерок. Если 555 не найдено, возвращается 1, иначе – 0.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	kjhsbdkjvhb; iuwervburewhv jwbhe?555 srvbr;ergr.Dragon flew away!	kjhsbdkjvhb; iuwervburewhv jwbhe? ergr. Dragon flew away! Количество предложений до 4 и количество предложений после 3	Программа работает верно

2.	hhhh?555 f; h555.5555;jrthb 555; 555 ;jhjh555isbv;jhbjh 555 555.jjhub 555,jhrbvjhb? uhbruhvb. dktjhb ,555{jhwegew.Dragon flew away!	hhhh? 5555; jhjh555isbv; uhbruhvb. Dragon flew away! Количество предложений до 11 и количество предложений после 4	Программа работает верно
3.	Dr; Dragon; Dragon flew!; Dragon flew away. Dragon flew away? Dragon flewaway!Dragon flew away!	Dr; Dragon; Dragon flew! ; Dragon flew away. Dragon flew away? Dragon flewaway! Dragon flew away! Количество предложений до 7 и количество предложений после 7	Программа работает верно

Выводы.

Были изучены принципы работы с указателя и массивами.

Разработана программа, которая форматирует некоторый введенный текст и выводит результат на консоль. При вводе предложения записываются в массивы, для которых была динамически выделена память. Текст хранится в виде двумерного динамического массива, элементы которого – указатели на массивы предложений.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

//Функция для получение предложения в массив предложений
char* get_sentence(){
    int size = 10;
    char* sent = (char*)malloc(size *sizeof(char));
    int count_let = 0;
    int is_begin = 1;//Начало ли предложения
    while(1){
        if (count_let == size-1){
            char* tmp = (char*)realloc(sent,(size+10)*sizeof(char));
            sent = tmp;
            size+=10;
        }
        sent[count_let] = (char)getchar();//Запись букв в предложение

        //Исключение табуляции и пробелов
        if (sent[count_let] == ' ' || sent[count_let] == '\t' ||
sent[count_let] == '\n'){
            if (is_begin){count_let--;}//Если запрещенные символы и нача
ло, то остаемся на том же индексе(дальше будет++)
        }else{is_begin = 0;}//Иначе отмечаем, что уже не начало предложен
ия


        //Условие окончания предложения
        if (sent[count_let] == '.'||
            sent[count_let] == ';'||
            sent[count_let] == '?'||
            sent[count_let] == '!' ){break;}
        count_let++;

    }
    sent[count_let+1] = '\0';
    return sent;
}

int is_equal(char* str1, char* str2){
    int flag = 1;
    for (int i = 0;i<strlen(str2);i++){
        if (str1[i] != str2[i]){
            flag = 0;
        }
    }
    return flag;
}
```

```
int is_sent_valid(char* sent){
    int i = 0;
    int count_5 = 0;
    while (sent[i] != '\0'){
        if(sent[i] == 53){
            count_5++;
            i++;
        }else if ((sent[i] > 47 && sent[i] < 58) || (sent[i]>64 && sent[i]
< 91) || (sent[i]>96 && sent[i] < 123)){
            count_5 = 0;
            i++;
        }else {
            if(count_5 == 3){
                return 0;
            }else{
                count_5 = 0;
                i++;
            }
        }
    }
    if(count_5 == 3){return 0;}else{return 1;}
}

int main() {
    int size_txt = 20;
    char** text = malloc(size_txt*sizeof(char*));
    char* stop_sent = "Dragon flew away!";
    int count_sent = 0;//Конечное количество предложений
    int count_sent_b = 0;//Начальное количество предложений
    while(1){
        if (count_sent == size_txt){
            char**                              tmp                              =
(char**)realloc(text,(size_txt+20)*sizeof(char*));
            text = tmp;
            size_txt+=20;
        }

        char* sent = get_sentence();
        if (is_sent_valid(sent)){
            text[count_sent] = sent;
            if (is_equal(text[count_sent],stop_sent)){
                break;}
            count_sent++;
            count_sent_b++;
        }else{
            count_sent_b++;
        }
    }

    for(int i = 0;i<=count_sent;i++){
        printf("%s\n",text[i]);
    }

    for (int i=0; i < count_sent;i++){
        free(text[i]);
    }
    free(text);
```

```
    printf("Количество предложений до %d и количество предложений посл
e %d",count_sent_b,count_sent);
    return 0;
}
```