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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Программирование»
Тема: Указатели и массивы

Студентка гр. 1304 ____ Нго Тхи Йен

Преподаватель ____ Чайка К. В.

Санкт-Петербург
2021

Цель работы.

Научиться использовать указатели, массивы и динамическую память

Основные теоретические положения.

Указатели

Указатель – это переменная, содержащая адрес другой переменной. (Керниган и Ритчи, 5 раздел).

Синтаксис объявления указателя:

<тип_переменной_на_которую_ссылается_указатель>* <название переменной>;

Связь указателей и массивов:

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 A; 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]).

Адресная арифметика

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.

Динамическая память

Вы ознакомились со статическим выделением памяти в программах на языке С. Однако, 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.

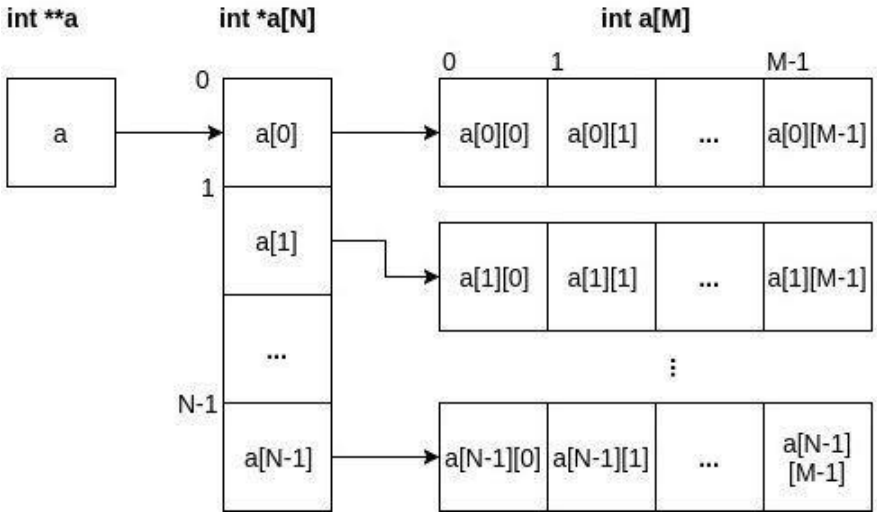
Статические двумерные массивы

В языке С предусмотрены статические многомерные массивы, хотя на практике чаще находят применение динамические. Простейший многомерный массив - двумерный. Его объявление похоже на объявление одномерного массива, только теперь в своих квадратных скобках указывается еще и вторая

размерность. Первая размерность задает количество строк, вторая - количество столбцов (размер каждой строки). Индексируются многомерные массивы также с нуля.

int arr[10][5]; //объявление двумерного массива из 50 элементов

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. Визуально в общем виде это можно представить следующим образом:



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).

Строки

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.
Как считать строку?

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

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

int scanf(const char * format, arg1, arg2, ...argN);

- %s в форматной строке для ввода строки
- Считывает символы до первого символа табуляции (не помещая его в строку)
- Не контролирует размер буфера
- Потенциально опасна

char* gets(char* str);

- Не контролирует размер буфера
- Потенциально опасна

Массивы строк

Как вы уже могли догадаться, если строка в Си - массив символов, то массив строк это двумерный массив символов, где каждая строка - массив, хранящий очередную символьную строку.

Статический массив строк может быть также инициализирован при объявлении.

Лабораторная работа №3: Использование указателей

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.

Тестирование.

Результаты тестирования представлены в таблице 1. Все результаты соответствуют ожидаемым.

Таблица 1 – Результаты тестирования.

№ п/п	Входные данные	Выходные данные
1.	\tab\tc. b3*cg; \t\tg? \t\tThg. Fgh? Dragon flew away! Dragon flew away! absc.	ab\tc. b3*cg; Thg. Dragon flew away! Количество предложений до 5 и количество предложений после 3
2.	Dragon flew away!	Dragon flew away! Количество предложений до 0 и количество предложений после 0

Ход работы

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

 int in_word(char*** word, char* fisrt_str, char* end_sep) {
 int i_str, i_ch;
*word = NULL;
 i_str = 0;
do{
    *word= realloc(*word, sizeof(char*)*(i_str+1));
   (*word)[i_str] = NULL;
    i_ch = 0;

    scanf(" ");
    do {
        (*word)[i_str] = realloc((*word)[i_str], sizeof(char)*(i_ch+1));
        scanf("%c", (*word)[i_str]+i_ch);
        i_ch++;
    } while (!strchr(fisrt_str, (*word)[i_str][i_ch-1]));
    (*word)[i_str] = realloc((*word)[i_str], sizeof(char)*(i_ch+1));
    (*word)[i_str][i_ch] = '\0';
    i_str++;
    } while (strcmp((*word)[i_str-1], end_sep));
    return i_str;
 }
 int remove_questions(char** word, int count){
 char** ans;
```

```
    ans = word;
    while (ans < word + count)
        if (strchr(*ans, '?')){
            free(*ans);
            count--;
            memmove(ans, ans+1, sizeof(char*)*(word+count-ans));
        } else ans++;
      return count;
    }

    void out_word(char** word, int count, char* str){
    char** ans;
    for (ans = word; ans < word+count; ans++)
        printf("%s%s", *ans, str);
    }

   void free_word(char** word,  int count){
        char** ans;
        for (ans = word; ans < word+count; ans++)
            free(*ans);
        free(word);
   }

   int main(){
       char** word; int result, new_result;
       result = in_word(&word, ";.?!", "Dragon flew away!");
       new_result = remove_questions(word, result);
       out_word(word, new_result, "\n");
       printf("Количество предложений до %d и количество предложений после %d", result-1, new_result-
1);
       free_word(word, new_result-1);
       return 0;
   }
```

Выводы.

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

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