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. 7381 _____ Лукашев Р.С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург
2017

Цель работы.

Изучить указатели в языке Си и научиться основам работы с ними.
Научиться работать с динамической памятью в языке Си.
Создать проект на основе полученных знаний.

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, конструкции вида &(x-1) и &3 являются незаконными. 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.

Если p_array — указатель, указывающий 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 указывает на предыдущий элемент, т.е. на array[i-1].
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.

Если нам необходимо изменить значение некоторой внешней переменной внутри функции, необходимо использовать не саму переменную, а указатель на нее (поскольку иначе функция работала бы с копией нашей переменной, изменение переменной внутри функции не повлияло бы на оригинальную). При вызове функции в таком случае следует передать в качестве аргумента адрес нашей переменной.

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

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.

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.
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 можно считать испульзуя следующие стандартные функции:

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

Ход работы.

Был написан проект программы, которая форматирует некоторый текст и выводит результат на консоль.

На вход программе подается текст который заканчивается предложением "Dragon flew away!".

Предложения (кроме последнего) могут заканчиваться на:

. (точка)

; (точка с запятой)

? (вопросительный знак)

Программа изменяет и выводит текст следующим образом:

Каждое предложение начинается с новой строки.

Табуляция в начале предложения удалена.

Все предложения, которые заканчиваются на '?' удалены.

Текст заканчивается 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 меняется
* Статическое выделение памяти под текст запрещено
* Пробел между предложениями является разделителем, а не частью какого-то предложения

Для проекта написан Makefile.

Выводы.

Были изучены основы работы с указателями в языке Си, динамической памятью. Были усовершенствованы старые навыки написания программ на языке Си, создан проект на основе полученных знаний.

Исходный код проекта.

- Файл «Makefile»:

```
All: main.o
	gcc main.o
```

```
main.o: main.c
	gcc -c main.c
```

- Файл «main.c»:

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

#define N 16
#define term "Dragon flew away!"

int main(){
 char* sen = (char*)calloc(1,sizeof(char));
  int s = 0;
   int n = 0, m = 0;
    char c;
   while(strcmp(sen,term)){
        c = getchar();
        if((s==0) && ((c=='\t') || (c==' ')))
        continue;
        if ((s % N) == 0)
         sen = (char*)realloc(sen, (strlen(sen)+N+1)*sizeof(char));
        sen[s++] = c;
        if ((c == '.') || (c == ';')){
         sen[s]='\0';
          printf("%s\n", sen);
           free(sen);
            sen = (char*)calloc(1,sizeof(char));
           n++;
          m++;
         s = 0;
        }
    if (c=='?'){
        n++;
         s = 0;
         free(sen);
        sen = (char*)calloc(1,sizeof(char));
    }
  }
 printf("%s\n", sen);
 printf("Количество предложений до %d и количество предложений после %d",
n, m);
 free(sen);
return 0;
```

```
}
```