



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Adrián Ulises Mercado Martínez

Asignatura: Fundamentos de Programación

Grupo: 15

No de Práctica(s): 11: Arreglos unidimensionales y multidimensionales.

Integrante(s): Morales Velasco Brayan, Lagunas Parra Jaime,
Walls Chávez Luis Fernando

*No. de Equipo de
cómputo empleado:* 41, 42 y 3

No. de Lista o Brigada: Brigada 13

Semestre: 2020-I

Fecha de entrega: 31/10/19

Observaciones:

CALIFICACIÓN: _____

Introducción

Arreglos:

Un arreglo es un conjunto de datos con un número fijo de componentes, todos del mismo tipo, que están referenciados bajo un mismo nombre. Cada componente del arreglo se puede acceder mediante índices (0, 1, 2, 3, ...) encerradas entre corchetes []. Estos permiten manejar de forma sencilla y directa conjuntos de datos del mismo tipo, de los cuales conocemos su cantidad y con los cuales se realizarán operaciones similares.

Un arreglo unidimensional es la estructura natural para modelar listas de elementos iguales. Se caracteriza por ser un acceso directo, es decir, podemos acceder a cualquier elemento del arreglo sin tener que consultar a elementos anteriores o posteriores utilizando el índice.

Los arreglos multidimensionales tienen más de una dimensión. En C, las dimensiones se manejan por medio de corchetes, dentro de los que se escriben los valores de cada dimensión y en un par separado cada tamaño.

Apuntadores:

Cuando se declara una variable, el compilador reserva un espacio de memoria para ella y asocia el nombre de ésta a la dirección de memoria desde donde comienzan los datos de esa variable. Las direcciones de memoria se suelen describir como números en hexadecimal.

Un apuntador es una variable cuyo valor es la dirección de memoria de otra variable. Se dice que un apuntador “apunta” a la variable cuyo valor se almacena a partir de la dirección de memoria que contiene el apuntador. Por ejemplo, si un apuntador p almacena la dirección de una variable x, se dice que “p apunta a x”.

Sintaxis:

- Arreglos:

TipoDeDato nombre[tamaño]

- Apuntadores:

*TipoDeDato *apuntador, variable;*

apuntador = &variable;

Desarrollo

1.- Comenzamos la práctica retomando aspectos de la práctica pasada y, como hubo algunas instrucciones que no pudimos probar en los compiladores locales, entramos a un servidor para poder usar el compilador y depurador funcional de ese ordenador.

```
Practica11 — fp15alu51@samba:~ — ssh fp15alu51@192.168.3.200 — 101x20
Last login: Thu Oct 24 15:15:30 on console
India54:~ fp15alu51$ ls
Desktop      Downloads    Movies       Pictures
Documents    Library     Music        Public
India54:~ fp15alu51$ cd Documents
India54:Documents fp15alu51$ ls
India54:Documents fp15alu51$ mkdir Practica11
India54:Documents fp15alu51$ cd Practica11
India54:Practica11 fp15alu51$ vim Practica11.c

#include<stdio.h>
#define TAM 5
int main()
{
    int lista[TAM]={10, 8, 5, 8, 7};

    int i=0;

    printf("\tLista\n");
    while (i<5)
    {
        printf("\nCalificación del alumno %d es %d",i, lista[i]);
        i++;
    }
    printf("\n");
    return 0;
}
~
~
"Practica11.c" 17L, 228C                                     1,1      All

India54:Practica11 fp15alu51$ servidor
India54:Practica11 fp15alu51$ ssh fp15alu51@192.168.3.200
The authenticity of host '192.168.3.200 (192.168.3.200)' can't be established.
RSA key fingerprint is SHA256:jTgFsbnpV7IaIpwchV27DaUa9i2pvAVVZwZzbIneOF8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.3.200' (RSA) to the list of known hosts.
fp15alu51@192.168.3.200's password:

Samba

-bash: aviso: setlocale: LC_CTYPE: no se puede cambiar el local (UTF-8)
[fp15alu51@samba ~]$

[fp15alu51@samba ~]$ ls
Escritorio
[fp15alu51@samba ~]$ cd Escritorio
[fp15alu51@samba Escritorio]$ ls
Practica11.c  Proyector.desktop
```

```
Practica11 — fp15alu51@samba:~/Escritorio — ssh fp15alu51@192.168.3.200 — 101x20
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /users/fp15/fp15alu51/Escritorio/P11_1...done.
(gdb) quit
[fp15alu51@samba Escritorio]$ gdb P11_1
GNU gdb (GDB) Fedora (7.4.50.20120120-42.fc17)
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /users/fp15/fp15alu51/Escritorio/P11_1...done.
(gdb)
```

1a. Así que compilamos el código antes mostrado junto con información de depuración y al ejecutarlo probamos el comando “l” que muestra las líneas de código de 10 en 10.

```
Practica11 — fp15alu51@samba:~/Escritorio — ssh fp15alu51@192.168.3.200 — 101x20
(gdb) l
1      #include<stdio.h>
2      #define TAM 5
3      int main()
4      {
5          int lista[TAM]={10, 8, 5, 8, 7};
6
7          int i=0;
8
9          printf("\tLista\n");
10         while (i<5)
(gdb) l
11         {
12             printf("\nCalificación del alumno %d es %d",i, lista[i]);
13             i++;
14         }
15         printf("\n");
16         return 0;
17     }
(gdb)
```

1b. Después con el comando “b” insertamos puntos de quiebre para detener el programa justo en la línea que queramos o necesitemos analizar. Para borrar estos, utilizamos “d” y el no. de línea.

```
Practica11 — fp15alu51@samba:~/Escritorio — ssh fp15alu51@192.168.3.200 — 101x20
14         }
15         printf("\n");
16         return 0;
17     }
(gdb) b 9
Breakpoint 1 at 0x40059e: file Practica11.c, line 9.
(gdb) b 10
Breakpoint 2 at 0x4005a8: file Practica11.c, line 10.
(gdb) b 13
Breakpoint 3 at 0x4005c7: file Practica11.c, line 13.
(gdb) b 15
Breakpoint 4 at 0x4005d1: file Practica11.c, line 15.
(gdb) b b 16
Function "b 16" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) b 16
Breakpoint 5 at 0x4005db: file Practica11.c, line 16.
(gdb) d 16
No breakpoint number 16.
(gdb)
```

1c. Después de ejecutar y que el programa sea detenido con el comando “c” el depurador continúa la ejecución. Y con “p” antes del nombre de una variable el depurador imprime el valor actual de la misma.

```

15         printf("\n");
(gdb) ./P11_1
Undefined command: ".". Try "help".
(gdb) run
Starting program: /users/fp15/fp15alu51/Escritorio/P11_1

Breakpoint 1, main () at Practica11.c:9
9         printf("\tLista\n");
Missing separate debuginfos, use: debuginfo-install glibc-2.15-37.fc17.x86_64
(gdb) c
Continuing.
        Lista

Breakpoint 2, main () at Practica11.c:10
10        while (i<5)
(gdb) p i
$1 = 0
(gdb) p i
$2 = 0
(gdb) c
Continuing.

Breakpoint 3, main () at Practica11.c:13
13        i++;
(gdb) p i
$3 = 0
(gdb) c
Continuing.
Calificación del alumno 0 es 10

Breakpoint 3, main () at Practica11.c:13
13        i++;
(gdb) p i
$4 = 1
(gdb) █

```

1d. También, con “l” al lado de un rango, podemos imprimir un no. específico de líneas.

```

Practica11 — fp15alu51@samba:~/Escritorio — ssh fp15alu51@192.168.3.200 — 101x20
Type "apropos word" to search for commands related to "word".
---Type <return> to continue, or q <return> to quit---q
Quit
(gdb) l 1,15
1      #include<stdio.h>
2      #define TAM 5
3      int main()
4      {
5          int lista[TAM]={10, 8, 5, 8, 7};
6
7          int i=0;
8
9          printf("\tLista\n");
10         while (i<5)
11         {
12             printf("\nCalificación del alumno %d es %d",i, lista[i]);
13             i++;
14         }
15         printf("\n");
(gdb) █

```

1e. Con “quit” salimos del depurador y posteriormente con “exit” cerramos la sesión en el servidor.

```
Breakpoint 5, main () at Practica11.c:16
16          return 0;
(gdb) c
Continuing.
[Inferior 1 (process 17838) exited normally]
(gdb) c
The program is not being run.
(gdb) quit
[[fp15alu51@samba Escritorio]$ exit
logout
Connection to 192.168.3.200 closed.
India54:Practica11 fp15alu51$
```

2. Pasamos a escribir nuestro segundo código y a enfocarnos más en el tema de la práctica, arreglos, específicamente unidimensionales.

```
Practica11 — fp15alu51@samba:~/Escritorio — vim Practica11.c — 101×36
#include<stdio.h>
#define TAM 5
int main()
{
    int lista[TAM]={10, 8, 5, 8, 7};

    int i=0;

    printf("\tLista\n");
    for(i=0;i<5;i++)
    {
        printf("\nCalificación del alumno %d es %d",i, lista[i]);
    }
    printf("\n");
    return 0;
}
~
```

En este programa recorreremos los datos del arreglo con un ciclo for, a diferencia del código anterior donde lo hacíamos con un ciclo while.

3. Ahora pasando a los apuntadores, en el siguiente código declaramos un apuntador y hacemos que apunte a una variable “c” de tipo char. Con esto, podemos imprimir su valor, así como su dirección de memoria.


```
#include <stdio.h>

int main(){
    char *ap, c= 'a';
    ap = &c;
    printf("Caracter: %c\n",*ap);
    printf("Código ASCII: %d\n",*ap);
    printf("Dirección de memoria: %p\n",ap);
    return 0;
}
```

```
India54:Practica11 fp15alu51$ gcc P11_3.c -o P11_3
India54:Practica11 fp15alu51$ ./P11_3
Caracter: a
Código ASCII: 97
Dirección de memoria: 0x7ffee2de1adf
India54:Practica11 fp15alu51$
```

Esta dirección cambia cada vez que el programa se ejecute ya que no siempre se guardan las variables en el mismo espacio.

4. En este paso, el código siguiente declaramos un arreglo sin tamaño ya que estamos definiendo sus valores uno por uno. También se declara un apuntador y arrojam los valores del arreglo con ayuda de una tercera variable “x”.



```
Practica11 — fp15alu51@samba:~/Escritorio — vim P11_5.c — 77x36
#include <stdio.h>

int main ()
{
    int arr[]={4,5,3};
    int *ap;
    ap=arr;

    int x = *ap;
    printf("%d\\n",x);

    x=*(ap+1);
    printf("%d\\n",x);
    x=*(ap+2);
    printf("%d\\n",x);
    return 0;
}
```

```
India54:Practica11 fp15alu51$ gcc P11_5.c
India54:Practica11 fp15alu51$ mv a.out P11_5
India54:Practica11 fp15alu51$ ./P11_5
4
5
3
India54:Practica11 fp15alu51$
```

5. Continuamos modificando un poco el código anterior agregando otro apuntador que también imprima el arreglo, pero con otros parámetros.

```
Practica11 — fp15alu51@samba:~/Escritorio — vim P11_6.c — 77x36
#include <stdio.h>

int main ()
{
    int arr[]={4,5,3};
    int *ap, *a2;
    ap=arr;

    int x = *ap;
    printf("%d\n",x);

    x=*(ap+1);
    printf("%d\n",x);
    x=*(ap+2);
    printf("%d\n",x);

    a2 = arr+1;
    x= *(a2-1);
    printf("%d\n", x);
    return 0;
}
~
[India54:Practica11 fp15alu51$ vim P11_6.c
[India54:Practica11 fp15alu51$ gcc P11_6.c -o P11_6
[India54:Practica11 fp15alu51$ ./P11_6
4
5
3
4
India54:Practica11 fp15alu51$
```

6. En el siguiente código ahorra recorreremos el arreglo, sumándole valores a *ap con un ciclo for.

```
India54:Practica11 fp15alu51$ vim P11_7.c
India54:Practica11 fp15alu51$ gcc P11_7.c -o P11_7
India54:Practica11 fp15alu51$ ./P11_7
0 4
1 5
2 3
India54:Practica11 fp15alu51$
```

16 lines (13 sloc) | 170 Bytes

```
1  #include <stdio.h>
2
3  int main ()
4  {
5      int arr[]={4,5,3};
6      int *ap, *a2;
7      ap=arr;
8
9      int i;
10     for(i=0; i<3; i++){
11         printf("%d %d\n",i,*(ap+i));
12     }
13     printf("\n");
14     return 0;
15 }
```


7. En el programa 8, declaramos ahora un arreglo de tipo char, que sería una cadena de caracteres con un tamaño definido de 30. Y con ayuda de la función strlen, y un ciclo for podemos imprimir la palabra letra por letra.

```
Practica11 — fp15alu51@samba:~/Escritorio — vim P11_8.c — 77x36
#include <stdio.h>
#include <string.h>

int main ()
{
    char word[30];
    int i=0;

    printf("Ingresa una palabra\n");
    scanf("%s", word);
    printf("La palabra es: %s\n", word);
    for(i=0; i<strlen(word); i++)
        printf("%c\n", word[i]);
    return 0;
}

~
India54:Practica11 fp15alu51$ vim P11_8.c
India54:Practica11 fp15alu51$ gcc P11_8.c -o P11_8
India54:Practica11 fp15alu51$ ./P11_8
Ingresa una palabra
Holaaaaaaaa
La palabra es: Holaaaaaaaa
H
o
l
a
a
a
a
a
a
a
a
India54:Practica11 fp15alu51$
```

8. En el penúltimo programa manejamos un arreglo multidimensional (2), el cual recorreremos con la ayuda de for y 2 variables.

```
Practica11 — fp15alu51@samba:~/Escritorio — vim P11_9.c — 77x36
#include <stdio.h>
int main ()
{
    int m[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int i,j;
    for(i=0;i<3;i++){
        for(j=0;j<3;j++){
            printf("%d", m[i][j]);
        }
    }
    return 0;
}

India54:Practica11 fp15alu51$ gcc P11_9.c -o P11_9
India54:Practica11 fp15alu51$ ./P11_9
123456789India54:Practica11 fp15alu51$
```

9. Y para finalizar, modificamos un tanto el código anterior para que con 1 apuntador podamos recorrerlo y tabularlo.

```
Practica11 — fp15alu51@samba:~/Escritorio — vim P11_10.c — 77x36
#include <stdio.h>
int main ()
{
    int m[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int *a, cont;
    int i ,j;
    a=m;

    for(i=0;i<9;i++){
        if (cont ==3){
            printf("\n");
            cont=0;
        }
        printf("%d\t", *(a+i));
        cont ++;
    }
    printf("\n");
    return 0;
}
~
```

```
[India54:Practica11 fp15alu51$ vim P11_10.c
[India54:Practica11 fp15alu51$ gcc P11_10.c -o P11_10
P11_10.c:7:3: warning: incompatible pointer types assigning to 'int *' from
'int [3][3]' [-Wincompatible-pointer-types]
    a=m;
    ^~
1 warning generated.
[India54:Practica11 fp15alu51$ ./P11_10
1      2
3      4      5
6      7      8
9
India54:Practica11 fp15alu51$
```

Conclusiones

Desde que vimos el tema de apuntadores en teoría no me quedó muy claro. Los arreglos sí; es una forma de acomodar varios datos por un mismo nombre y si es de varias dimensiones muy parecido a una forma matricial. Así que como en esta práctica usamos y combinamos el uso de ambos, resulta más fácil comprender el uso de un apuntador, el cual nos sirve para tomar como referencia, en vez de la variable en sí, a su dirección de memoria. Con lo cual podemos hacer códigos más versátiles y ahorrativos, aunque al costo de sí se escapó un error, tal vez sea más difícil solucionarlo; claro, no sin antes haber probado todas las herramientas de un depurador.

-Walls Chávez Luis Fernando

En ésta práctica se ha logrado conseguir un mayor entendimiento referente a los apuntadores y aun más para los arreglos.

Con esto queda mejor definido su funcionamiento en conjunto, la optimización de memoria, código, y mayor sencillez a la hora de realizar diversos programas sin necesidad de tener una gran extensión innecesaria tanto de uso de memoria, como de código.

También con ayuda del depurador y la conexión al servidor del laboratorio, el como hacerlo y la forma de usarlo fue de gran ayuda para futuras prácticas y para ésta misma en sí.

-Morales Velasco Brayan

Entender el tema de apuntadores y arreglos es bastante complicado, incluso algo tedioso como comprobamos en esta práctica y en las últimas clases. Los ejercicios nos han servido para mejorar el entendimiento del tema, pero en realidad me cuesta un poco entenderlo. En los ejercicios de esta práctica intentamos combinar el uso de ambos utilizando también conocimientos previos de la práctica pasado como lo de depuración. Espero y con futuros ejercicios, pueda quedarme más claro para que utilizamos cada concepto.

-Lagunas Parra Jaime Rodrigo