	
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación

Profesor: Saavedraa Hernández Honorato

Asignatura: Fundamentos de programación

Grupo: 01

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

Integrante(s): Luis Salinas Ludwig

Semestre: 2018-1

Fecha de entrega: 13/11/2017

Observaciones:

CALIFICACIÓN: _____

salas A y B

Objetivo

Reconocer la importancia y utilidad de los arreglos, en la elaboración de programas que resuelvan problemas que requieran agrupar datos del mismo tiempo, así como trabajar con arreglos tanto unidimensionales como multidimensionales.

Desarrollo

Comenzamos por tratar de entender que es un arreglo y cuál es su función, de lo cual se vio que un arreglo es el conjunto de datos contiguos del mismo tipo con un tamaño fijo definido al crearse. Ahora a cada dato del arreglo se asociamos una dirección, es importante denotar que existen arreglos unidimensionales como multidimensionales.

La sintaxis para definir un arreglo en lenguaje C es de la siguiente manera:

tipoDeDato nombre[tamaño]

Donde nombre se refiere al indicador de nuestro arreglo, tamaño al número entero que define al máximo número de elementos que se encuentran contenidos en el arreglo.

Pasamos a ver el código (arreglo unidimensional while) con el siguiente ejemplo:

```
#include <stdio.h>

/*
    Este programa genera un arreglo unidimensional de 5 elementos y los
    accede a cada elemento del arreglo a través de un ciclo while.
*/

int main (){
    #define TAMANO 5
    int lista[TAMANO] = {10, 8, 5, 8, 7};

    int indice = 0;

    printf("\tlista\n");
    while (indice < 5 ){
        printf("\nCalificación del alumno %d es %d", indice+1, lista[indice]);
        indice += 1;          // análogo a indice = indice + 1;
    }

    printf("\n");

    return 0;
}
```

The screenshot shows the Code::Blocks IDE with a C program in the editor and its execution output in the console. The program, named `main.c`, includes `<stdio.h>` and contains a `main` function. It defines a constant `TAMANO` as 5 and declares an array `lista` of 5 integers with values {10, 8, 5, 8, 7}. A `while` loop prints each element of the array. The console output shows the execution of the program, which terminated with an error.

```
1 #include <stdio.h>
2
3 /*
4  Esta funcion genera un arreglo unidimensional de 5 elementos y los
5  accede a cada elemento del arreglo a traves de un ciclo while.
6  */
7
8 int main () {
9     #define TAMANO 5
10    int lista[TAMANO] = {10, 8, 5, 8, 7};
11
12    int indice = 0;
13
14    printf("\nlista\n");
15    while (indice < 5) {
16        printf("\nCalificacion del alumno %d es %d", indice+1, lista[indice]);
17        indice += 1; // Si el ciclo se ejecuta 5 veces, el indice sera 5.
18    }
19
20    printf("\n");
21    return 0;
22 }
```

Run: Debug in práctica 11 (compiler: GCC GCC Compiler)
Checking for existence: D:\Users\fp01alu05\práctica 11\bin\Debug\práctica 11.exe
Executing: "C:\Program Files (x86)\CodeBlocks\cb_console_runner.exe" "D:\Users\fp01alu05\práctica 11\bin\Debug\práctica 11.exe" (in D:\Users\fp01alu05\práctica 11\.)
Process terminated with status -1073741510 (0 minute(s), 2 second(s))

Analizando el programa anterior logramos entender un poco más de cuál es la función de los apuntadores ya que estos tienen la función de apartarnos memoria, pero también al apartar memoria no utilizamos realmente toda esta memoria sino más bien solo una pequeña parte y lo demás lo desperdiciamos.

Para que un programa que su necesidad sea pedir memoria, sólo podrá hacerse si utilizamos apuntadores de otra manera no se podrá realizar dicho programa.

Para declarar algún apuntador indicando la dirección lo tenemos que hacer con un carácter (*) y el & representaría la dirección de esa variable a la cual le estemos buscando su dirección, ya que cada variable tiene una dirección.

Con lo anterior dicho el carácter (*) lo leeríamos como el contenido de la dirección guardada en esa variable, al momento de ver lo que nuestro programa nos imprime en pantalla nos mostrara parte de la dirección e sistema hexadecimal.

Para entender mejor lo anterior recapitulamos. Un apuntador es una variable que contiene la dirección de otra variable, ya que los apuntadores trabajan directo con la memoria. La sintaxis para verlo de una manera más clara al apuntador es la siguiente:

TipoDeDato*apuntador,variable;

Apuntador = &variable

Como anteriormente lo comentamos la declaración de una variable apuntador inicia con el carácter (*). Así mismo cuando una variable inicia con un ampersand (&) de esta manera accede a la dirección de la memoria que es lo mismo cuando leemos un dato con scanf.

Continuamos con el Código apuntadores.

```
#include <stdio.h>

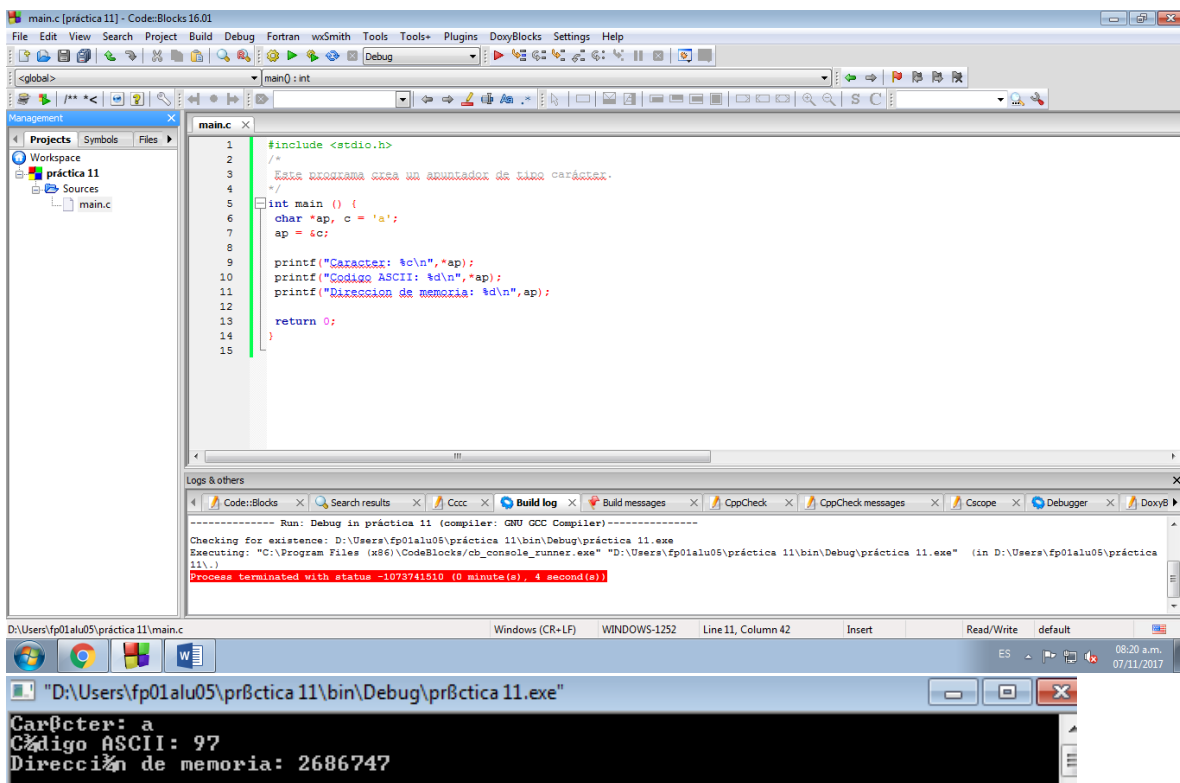
/*
    Este programa crea un apuntador de tipo carácter.
*/

int main () {
    char *ap, c = 'a';
    ap = &c;

    printf("Carácter: %c\n",*ap);
    printf("Código ASCII: %d\n",*ap);
    printf("Dirección de memoria: %d\n",ap);

    return 0;
}
```

En este programa vemos que el carácter 'a' se guardó en alguna parte de la memoria.

The screenshot shows a C code editor window titled 'main.c [práctica 11] - Code::Blocks 16.01'. The code is the same as shown in the previous block. The 'Log messages' window at the bottom shows the execution output: 'Carácter: a', 'Código ASCII: 97', and 'Dirección de memoria: 2686747'. The status bar at the bottom indicates the file path 'D:\Users\fp01alu05\práctica 11\main.c', the window title 'Windows (CR+LF)', the file encoding 'WINDOWS-1252', the current line and column 'Line 11, Column 42', and the editor mode 'Insert'. The system tray at the bottom right shows the date and time '08:20 a.m. 07/11/2017'.

Continuamos con otro ejemplo.

```
#include<stdio.h>

/*
    Este programa accede a las localidades de memoria de distintas variables a
    través de un apuntador.
*/

int main () {
    int a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0};
    int *apEnt;
    apEnt = &a;

    printf("a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0}\n");
    printf("apEnt = &a\n");

    b = *apEnt;
    printf("b = *apEnt \t-> b = %i\n", b);

    b = *apEnt +1;
    printf("b = *apEnt + 1 \t-> b = %i\n", b);

    *apEnt = 0;
    printf("**apEnt = 0 \t-> a = %i\n", a);

    apEnt = &c[0];
    printf("apEnt = &c[0] \t-> apEnt = %i\n", *apEnt);

    return 0;
}
```

The screenshot shows the Code::Blocks IDE with the following components:

- Editor:** Displays the C program code from the previous block, with line numbers 1 through 21. The code is color-coded (keywords in blue, strings in red, etc.).
- Management Panel:** On the left, showing the project structure with 'práctica 11' and its source file 'main.c'.
- Logs & others:** At the bottom, showing the execution output:


```
----- Run: Debug in práctica 11 (compiler: GNU GCC Compiler)-----
Checking for existence: D:\Users\fp01alu05\práctica 11\bin\Debug\práctica 11.exe
Executing: "C:\Program Files (x86)\CodeBlocks\cb_console_runner.exe" "D:\Users\fp01alu05\práctica 11\bin\Debug\práctica 11.exe" (in D:\Users\fp01alu05\práctica 11\.)
```
- Taskbar:** At the very bottom, showing the Windows taskbar with the Start button, taskbar buttons for Chrome, Word, and other applications, and a system clock showing 08:32 a.m. on 07/11/2017.

Donde apEnt ahora lo veremos como un alias que sigue siendo un apuntador. Todo este programa es lo mismo que lo siguiente:

```
apEnt = &c[0];  
apEnt = c;
```

Continuamos con el código apuntadores en cadena, dónde observamos el siguiente programa:

```
#include <stdio.h>  
  
/*  
    Este programa muestra el manejo de cadenas en lenguaje C.  
*/  
  
int main(){  
    char palabra[20];  
    int i=0;  
  
    printf("Ingrese una palabra: ");  
    scanf("%s", palabra);  
    printf("La palabra ingresada es: %s\n", palabra);  
  
    for (i = 0 ; i < 20 ; i++){  
        printf("%c\n", palabra[i]);  
    }  
  
    return 0;  
}
```

The screenshot shows the Code::Blocks IDE with a C program named `main.c` open. The program includes `<stdio.h>` and contains a `main` function. It prompts the user to enter a word, reads it using `scanf`, and then prints each character of the word in a loop. The execution logs at the bottom show the program running successfully, with the message "Process terminated with status -1073741510 (0 minute(s), 13 second(s))".

```
1 #include <stdio.h>
2 /*
3  * Esta programa muestra el manejo de cadenas en lenguaje C.
4  */
5 int main(){
6     char palabra[20];
7     int i=0;
8     printf("Ingrese una palabra: ");
9     scanf("%s", palabra);
10    printf("La palabra ingresada es: %s\n", palabra);
11    for (i = 0 ; i < 20 ; i++){
12        printf("%c\n", palabra[i]);
13    }
14    return 0;
15 }
16
17
```

----- Run: Debug in 232 (compiler: GNU GCC Compiler)-----
Checking for existence: D:\Users\fp01alu05\232\bin\Debug\232.exe
Executing: "C:\Program Files (x86)\CodeBlocks\cb_console_runner.exe" "D:\Users\fp01alu05\232\bin\Debug\232.exe" (in D:\Users\fp01alu05\232\.)
Process terminated with status -1073741510 (0 minute(s), 13 second(s))

Retomando lo comentado anteriormente sobre el `scanf` ahora lo podemos ver que es la dirección de una palabra para el caso de este programa.

Conclusiones.

Se entendió que un apuntador es una variable que contiene la dirección de otra variable y que `*ap` hay que leerlo como la dirección de una variable que se guarda en otra variable ya que si no logramos entender lo anterior llega a pasar que comienza la confusión. Los apuntadores nos ayudan para poder ver la dirección de algo dentro de la memoria y que estos apuntadores solo pueden apuntar a la dirección de memoria del mismo tipo de dato con el que se declararon.