	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	Edgar Tista García
Asignatura:	Estructura de Datos y Algoritmos I
Grupo:	1
No de Práctica(s):	Práctica #2
Integrante(s):	Santiago Díaz Edwin Jaret
No. de Equipo de cómputo empleado:	Trabajo en casa
No. de Lista o Brigada:	42
Semestre:	2021-2
Fecha de entrega:	19/03/2021
Observaciones:	

CALIFICACIÓN: _____

APLICACIONES DE APUNTADORES

Objetivos

Objetivo: Utilizar apuntadores en lenguaje C para acceder a las localidades de memoria tanto de datos primitivos como de arreglos.
Objetivos clase: Utilizar apuntadores para modificar valores de variables indirectamente, así como para realizar el manejo de arreglos y el paso por referencia en funciones.

Desarrollo

Ejemplo 1

El programa utiliza un apuntador para acceder a la dirección de la memoria de variable de tipo carácter. La variable tiene de valor asignado de "x", por lo tanto, el apuntador equivale a ese carácter.
El programa:

- Imprime el carácter utilizando el apuntador **"*ap"** y para representarlo con el tipo de dato carácter usa **"%c"**.
- Imprime el Código ASCII del carácter, utilizando el apuntador **"*ap"** y para obtener su valor en código se utiliza **"%d"**.
- Imprime la dirección de memoria, en este caso el apuntador no lleva **"*"**, y se utiliza **"%d"** para obtener los números equivalentes a la dirección de memoria.

El Código para imprimir

```
//Imprime el carácter de la localidad a la que apunta
printf("Car%ccter: %c\n",aa,*ap);
// Imprime el codigo ASCII de la localidad a la que apunta
printf("C%cdigo ASCII: %d\n",ao,*ap);
// Imprime la dirección de memoria de la igualdad a la que apunta
printf("Direcci%cn de memoria: %d\n\n",ao,ap);
```

Lo que se imprime en la pantalla

```
Carácter: x
Código ASCII: 120
Dirección de memoria: 6487567
```

El programa no tuvo errores.
Este es un buen ejemplo en código para la introducción a los apuntadores, con la teoría vista en clase se puede entender fácilmente cómo funcionan, además, podemos observar la distinta información que tiene un solo apuntador.

Ejemplo 2

El programa tiene variables y un apuntador de tipo entero, el cual va a modificar el valor de las variables de manera indirecta.

- Se asigna el apuntador con la variable **a**. La variable tiene un valor de 5.
- Al valor del apuntador se guarda en otra variable llamada **b**.
- La variable **b** guarda el valor del apuntador + 1.
- Se le asigna un valor al apuntador de 0, por lo tanto, la variable **a** vale 0.

```
a = 5
b = 5 //*apEnt
b = 6 //*apEnt+1
a = 0 //*apEnt = 0
```

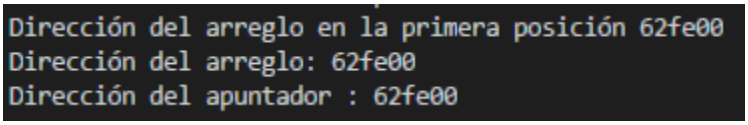
El programa no tuvo errores.
En pocas palabras, el ejemplo se relaciona con la teoría al explicar que se puede modificar, asignar valores a través de un apuntador.

Ejemplo 3

El programa utiliza un arreglo y un apuntador, ambos de tipo entero. El apuntador es asignado al arreglo, pero específicamente, se le asigna a la dirección de memoria de la primera posición del arreglo.

El arreglo imprime:

- La dirección del arreglo de la posición [0]
- La dirección del arreglo
- La dirección del apuntador

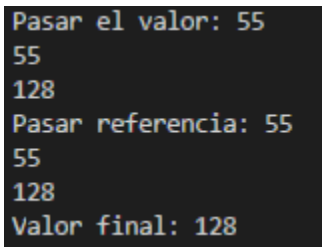


Las 3 salidas tienen el mismo valor de la memoria, por lo tanto, el apuntador está asignado a la misma dirección de memoria que del arreglo y la primera posición del arreglo es donde inicia el arreglo.
No hubo errores.
Se relaciona con la teoría para obtener las direcciones de las memorias de las variables o apuntadores.

Ejemplo 4

El programa utiliza 2 funciones, para paso por valor y paso por referencia. Se trabaja con un apuntador y con un arreglo, el apuntador está asignado a la dirección de memoria del arreglo.
El arreglo trabaja:

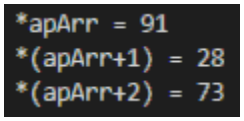
- Con el apuntador, imprime el primer valor del arreglo.
- Ese valor se pasa por la primera función y se guarda en otra variable, el cual, imprime el valor y después, a la variable se cambia de valor. Al final se imprime el nuevo valor. El apuntador no cambia de valor, es paso por valor
- Se manda a llamar la segunda función pasando un valor y utilizando el apuntador. Se imprime el valor que se "pasa" y después se modifica. Se imprime el valor modificado. El valor del apuntador fue cambiado.
- Se imprime el valor final después de recorrer las funciones. Se obtiene un nuevo valor por el paso anterior



No hubo errores.
Se relaciona con la teoría al utilizar 2 tipos de funciones, paso por valor y paso por referencia, además, aplicando apuntadores.

Ejemplo 5

El programa utiliza un apuntador y un arreglo de 5 columnas. El apuntador es asignado al arreglo para poder imprimir los datos del arreglo desde el apuntador.



- Imprime el valor inicial del arreglo desde el apuntador
- Imprime el Segundo valor del arreglo sumando “1” a la localidad inicial del apuntador
- Imprime el tercer valor del arreglo sumándole “2” a la localidad inicial del apuntador.

```
printf("*apArr = %i\n", *apArr);
printf("*(apArr+1) = %i\n",*(apArr+1));
printf("*(apArr+2) = %i\n\n",*(apArr+2));
```

El programa no tuvo errores. El ejemplo es una introducción para utilizar los apuntadores sumándole una cifra a la dirección de memoria, esta teoría será fundamental.

Ejemplo 6

El programa utiliza un arreglo de 5 columnas y un apuntador, el cual, está asignado al arreglo. El programa imprime la dirección de la memoria en donde se encuentra cada elemento del arreglo, utilizando un *ciclo for.*

```
62fe00
62fe02
62fe04
62fe06
62fe08
```

Ejemplo 7

El funcionamiento del programa es, imprimir la dirección de la memoria de los elementos del arreglo, el cual es asignado a un apuntador y este es recorrido con un *ciclo for.*

```
62fde0  62fde4  62fde8
62fdec  62fdf0  62fdf4
62fdf8  62fdfc  62fe00
```

- En el *ciclo for,* a la dirección de memoria del apuntador se le suman el valor del ciclo (desde 0 a 9).

El programa tenía un error en el momento de asignarle al apuntador la dirección de memoria del arreglo, le faltaba el símbolo amperson “&”.

Antes	**Después**
`ap = nums;`	`ap = *nums;`

Relación

Ejemplo 8

El programa es un cifrado César en el cual, trabaja con arreglos y funciones. El programa puede cifrar y descifrar las palabras que el usuario desee ingresar. Para cifrar:

- El programa recorre “hacia Adelante” la palabra y va cambiando letra por la letra, la cual es la que sigue después de 4 espacios. Si dentro de la palabra hay una letra “E” ésta es cambiada por la letra “H” ya que 4 espacios delante de la E se encuentra la letra H

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
DEFGHIJKLMNOPQRSTUVWXYZABC
```

```
      *** CIFRADO CÉSAR ***
ABCDEFGHIJKLMNOPQRSTUVWXYZ
DEFGHIJKLMNOPQRSTUVWXYZABC
Elegir una opción:
1) Cifrar
2) Descifrar.
3) Salir
1
Ingresar la palabra a cifrar (en mayúsculas): RETIRADA

El texto RETIRADA cifrado es:
U
H
W
L
U
D
G
D
```

Para descifrar:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
DEFGHIJKLMNOPQRSTUVWXYZABC
```

- El programa recorre “hacia atrás” y cambiando letra por letra de la palabra por la letra que se encuentra está 4 espacios antes. Si dentro de la palabra hay “U” la letra que está 4 espacios antes es la “R”.

```
       *** CIFRADO CSAR ***
ABCDEFGHIJKLMNOPQRSTUVWXYZ
DEFGHIJKLMNOPQRSTUVWXYZABC
Elegir una opción:
1) Cifrar
2) Descifrar.
3) Salir
2
Ingresar la palabra a descifrar (en mayúsculas) :UHWLUDGD

El texto UHWLUDGD descifrado es:
RETIRADA
```

Además, se puede terminar la ejecución del programa ingresando el número 3

```
       *** CIFRADO CSAR ***
ABCDEFGHIJKLMNOPQRSTUVWXYZ
DEFGHIJKLMNOPQRSTUVWXYZABC
Elegir una opción:
1) Cifrar
2) Descifrar.
3) Salir
3
```

El programa no tuvo errores. El programa solo usa funciones y arreglos, pero gracias a eso, el programa fue fácil de implementar.

Ejercicio 1

El programa utiliza variables y apuntadores de tipo entero. Su función es asignar a los apuntadores a la dirección de memoria de las variables modificarlas de manera indirecta, estas variables ya tenían un valor asignado.

```
int w = 10, x = 20, y = 30, z = 40, r1, r2, r3;
int *p1, *p2, *p3, *p4;
p1 = &w; //10
p2 = &x; //20
p3 = &y; //30
p4 = p1; //10
r1 = *p1 + *p2; // 10 + 20 = 30
r2 = *p3 * *p4; // 30 * 10 = 300
r3 = *p2 + *p3; // 20 + 30 = 50
printf("Los resultados son: %d, %d, y %d", r1, r2, r3);
```

- P1 equivale a 10 por la asignación de la variable “w”.
- P2 equivale a 20 por la variable “x”.
- P3 equivale 30 por el valor de “y”.
- P4 equivale a 10 por la asignación de la dirección de memoria de con “P1”.
- En la variable r1 se guarda la suma de p1 con p2, dando de resultado 30.
- En la variable r2 se guarda la multiplicación entre p3 y p4, el resultado es 300.
- En la variable r3 se guarda la suma de p2 con p3, el resultado es 50.

El resultado de las operaciones aritméticas con apuntadores da:

```
Los resultados son: 30, 300, y 50
```

El programa tenía errores.

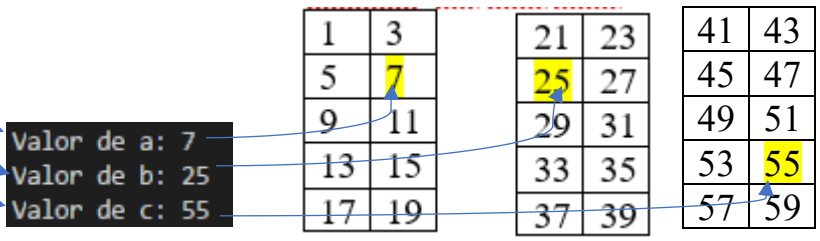
- No estaban bien asignados los apuntadores con la dirección de memoria de las variables

`` int w = 10, x = 20, y = 30, z = 40, r1, r2, r3; int *p1, *p2, p3, *p4; p1 = &w; *p2 = &x; *p3 = &y; p4 = *p1; r1 = p1 + *p2; r2 = p3 * p4; / *r3 = *p2 + *p3; printf("Los resultados son: %d, %d, y %d", r1, r2, r3); ```	Al asignar la dirección de memoria: • No deberían de tener "*" los apuntadores **p2** y **p3** • Al asignar en **p4** el **p1** tampoco debería de tener "*" Las operaciones aritméticas no eran las adecuadas en: • **R1**: faltaba "*" en p1. • **R2**: en ambos apuntadores les hacía faltaba "*" • **R3:** No debería de tener "*" en la variable r3 porque no es un apuntador.
`` int w = 10, x = 20, y = 30, z = 40, r1, r2, r3; int *p1, *p2, *p3, *p4; p1 = &w; p2 = &x; p3 = &y; p4 = p1; r1 = *p1 + *p2; r2 = *p3 * *p4; r3 = *p2 + *p3; printf("Los resultados son: %d, %d, y %d", r1, r2, r3); ```	Se corrigieron los errores y el Código queda de esta manera.

El ejercicio con se relaciona con la teoría al poder modificar de manera indirecta los valores de las variables, al utilizar la direcciones de memorias de las variables asignadas a los apuntadores.

Ejercicio 2

a) El programa le hacía falta aplicar un 4to *ciclo for* para la 4ta dimensión y agregar esa dimensión en la variable **arr**, ya que solo estaba usando 3 dimensiones.
b) EL programa al utilizar apuntador con un arreglo, Podemos acceder a los valores de mejor manera. En este caso con el ejemplo de las variables a, b y c, tenemos que al acceder con apuntadores, lo que obtenemos es:
a = *(point+3); == arr[0][0][1][1]
b= *(point+12); == arr[0][1][1][0]
c= *(point+27) == arr[0][2][3][1]



c) Utilizamos un ciclo for que empiece desde 0 y termine hasta 121 con varios secuencias if. Esto es para que la variable del for vaya recorriendo los espacios del arreglo y con el apuntador, será sumada a la dirección de memoria del apuntador. Con los ciclos for podemos obtener la información acerca de si se encuentra en un rango entre 0-11, 11-21, 31-41… así hasta llegar al 121, se le multiplicará los valores de 5, 6, 7 y 4 respectivamente.

- Entre los elementos de 0 a 41, tengo que encontrar un valor que con la resta me de 1 y 10, así al multiplicarlo me da los primeros números de la secuencia.

- Entre los elementos de 41 a 81 tengo que encontrar un valor que al restar me de los valores de 10 a 21, para multiplicarlos me den la secuencia del superplano 0.
- Entre los elementos de 81 a 121 tengo que encontrar un valor que al restar me de los valores de 20 a 31, para multiplicarlos me den la continuidad del superplano 1.

El resultado queda de la siguiente manera:

```
Superplano 0
[
En el plano 0 queda de la siguiente manera:
 [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]

En el plano 1 queda de la siguiente manera:
 [6, 12, 18, 24, 30, 36, 42, 48, 54, 60]

En el plano 2 queda de la siguiente manera:
 [7, 14, 21, 28, 35, 42, 49, 56, 63, 70]

En el plano 3 queda de la siguiente manera:
 [3, 6, 9, 12, 15, 18, 21, 24, 27, 30]

]
```

```
Superplano 1
[
En el plano 0 queda de la siguiente manera:
 [55, 60, 65, 70, 75, 80, 85, 90, 95, 100]

En el plano 1 queda de la siguiente manera:
 [66, 72, 78, 84, 90, 96, 102, 108, 114, 120]

En el plano 2 queda de la siguiente manera:
 [77, 84, 91, 98, 105, 112, 119, 126, 133, 140]

En el plano 3 queda de la siguiente manera:
 [33, 36, 39, 42, 45, 48, 51, 54, 57, 60]

]
```

```
Superplano 2
[
En el plano 0 queda de la siguiente manera:
 [105, 110, 115, 120, 125, 130, 135, 140, 145, 150]

En el plano 1 queda de la siguiente manera:
 [126, 132, 138, 144, 150, 156, 162, 168, 174, 180]

En el plano 2 queda de la siguiente manera:
 [147, 154, 161, 168, 175, 182, 189, 196, 203, 210]

En el plano 3 queda de la siguiente manera:
 [63, 66, 69, 72, 75, 78, 81, 84, 87, 3]

]
```

En el inciso C se creó un programa, para tener programas distintos entre los incisos A , B con el C.

Ejercicio 3

El programa solicita al usuario 2 valores (almacenadas en 2 variables) y con apuntadores, el programa realiza 2 operaciones matemáticas.

1. Con esos valores el programa elevará el primer valor a la potencia del segundo valor
2. El resultado será dividido entre el primer valor.

La salida de un programa al ingresar los valores de 8 y 4 da:

```
Ingrese el primer valor: 8
Ingrese el segundo valor: 4
Resultado de elevar el primero a la potencia del número del segundo: 4096
La división entre el valor obtenido anteriormente y el primero valor ingresado  da: 512
```

Y el funcionamiento con apuntadores es de la siguiente manera:

- Se pasará por una función 2 parámetros. El primero es el primer valor ingresado y el Segundo es el Segundo valor ingresado.
- En la dirección de memoria del Segundo valor se guarda el resultado de la operación de elevar el primer valor por el Segundo valor ingresado.
- En la dirección de memoria del primer valor se guarda el resultado de dividir el primer valor ingresado entre el valor de la operación anterior.

```
void valor(int *a, int *b) {
    *b = pow(*a, *b);
    *a = *b / *a;
}
```

No hubo errores en el programa. Al modificar valores de manera indirecta por los apuntadores, se aplica la teoría aplicada.

Ejercicio 4

El programa le solicita al usuario que ingrese valores de tipo entero para llenar una matriz de [5, 4] (5 renglones por 4 columnas). Al final, el programa regresará:

- Los renglones pares multiplicado por 2 (0, 2, 4)
- Los renglones impares multiplicados por 3 (1, 3)

```
Ingrese los valores de [3] y [3]:
1
Ingrese los valores de [4] y [0]:
1
Ingrese los valores de [4] y [1]:
1
Ingrese los valores de [4] y [2]:
1
Ingrese los valores de [4] y [3]:
1

El arreglo multiplicado por 2 del número del renglón divisibles entre 2 y
multiplicado por 3 del número de renglón divisible entre 3 el resultado es:
[2,2,2,2]

[3,3,3,3]

[2,2,2,2]

[3,3,3,3]

[2,2,2,2]
```

```
Ingrese los valores de [0] y [0]:
1
Ingrese los valores de [0] y [1]:
1
Ingrese los valores de [0] y [2]:
1
Ingrese los valores de [0] y [3]:
1
Ingrese los valores de [1] y [0]:
1
Ingrese los valores de [1] y [1]:
1
Ingrese los valores de [1] y [2]:
1
Ingrese los valores de [1] y [3]:
1
Ingrese los valores de [2] y [0]:
1
Ingrese los valores de [2] y [1]:
1
Ingrese los valores de [2] y [2]:
1
Ingrese los valores de [2] y [3]:
1
Ingrese los valores de [3] y [0]:
1
Ingrese los valores de [3] y [1]:
1
Ingrese los valores de [3] y [2]:
1
```

El programa funciona de la siguiente manera:

- Obtener los datos con un *cilclo for* y guardaros en un arreglo.
- Con un *ciclo for* de 0 a 5 representando los renglones, se divide entre 2 y si se obtiene un residuo de 0 es un renglón par, si no da 0, es un renglón impar.
- Si es un par, con un *ciclo for* se recorre los elementos del arreglo y se multiplican por 2.
- Si es impar, se recorre los elementos del arreglo y se multiplican por 3

El programa no tuvo errores.
El programa con relación a la teoría está enfocado en la modificación de arreglos con *ciclos for* y *secuencias if*. Para este ejercicio no se ocupó apuntadores, pero si se pueden aplicar.

Conclusiones

Todos los programas abarcaron la teoría aprendida, aplicando los arreglos, los apuntadores, las funciones de paso por valor y pasos por referencia, así mismo, los objetivos fueron cumplidos.
En la práctica se abarcó el 100% de los ejercicios y ejemplos de la guía, corrigiendo aquellos que tuvieron errores.
Los ejercicios fueron los adecuados para desarrollar una lógica de programación al aplicar lo nuevo aprendido y arreglar problemas. En todos lados estuvo la teoría.