



Nombre de la práctica	Apuntadores		No.	15
Asignatura:	MÉTODOS NUMÉRICOS	Carrera:	ISIC	Duración de la práctica (Hrs)
				6

I. Competencia(s) específica(s):

II. Lugar de realización de la práctica (laboratorio, taller, aula u otro):

III. Material empleado:

Visual studio code

IV. Desarrollo de la práctica:

```

C apuntador_n.c C apuntador_n.a.c C estruc_frac.c
home > mary > Escritorio > c > C estruc_frac.c > main()
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 struct fraccion{
5     int a, b, c, d;
6 };
7 void main(){
8     struct fraccion ingresa;
9     int suma, division, resta, multiplicacion;
10    int suma1, division1, resta1, multiplicacion1;
11    puts("Ingrese numero a");
12    scanf("%d",&ingresa.a);
13    puts("\n Ingrese numero b");
14    scanf("%d",&ingresa.b);
15    puts("\n Ingrese numero c");
16    scanf("%d",&ingresa.c);
17    puts("\n Ingrese numero d");
18    scanf("%d",&ingresa.d);
19    suma=(ingresa.a*ingresa.d)+(ingresa.c*ingresa.b);
20    suma1=ingresa.c*ingresa.d;
21    printf("La suma= %d",suma);
22    printf("\n      / \n      %d \n",suma1);
23    resta=(ingresa.a*ingresa.d)-(ingresa.c*ingresa.b);
24    resta1=ingresa.c*ingresa.d;
25    printf("\nLa resta= %d",resta);
26    printf("\n      / \n      %d \n",resta1);
27    division=(ingresa.a*ingresa.d);
28    division1=(ingresa.c*ingresa.b);
29
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ ./fr
Ingrese numero a
1
Ingrese numero b
1
Ingrese numero c
2
Ingrese numero d
3
La suma= 5
      /
      6

La resta= 1
      /
      6

La division= 3
      /
      2

La multilicacion= 1
      /
      6
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$

```

Un puntero es un objeto que apunta a otro objeto. Es decir, una variable cuyo valor es la dirección de memoria de otra variable. Para declarar un apuntador se especifica el tipo de dato al que apunta, el operador '*', y el nombre del apuntador. Creamos un apuntador de tipo entero entero, donde a este le asignamos un valor de una variable de igual entero, imprimimos un mensaje en pantalla en el que el apuntador es 17 puesto que antes se declara el valor, después hacemos una suma del apuntador con el valor de 17 mas 3 que este resultado se guarda en y e imprimimos ese resultado en otro mensaje.

```

Welcome C estructura_CD.c C apuntador_1.c
home > mary > Escritorio > c > C apuntador_1.c > main()
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 int main(){
5     int x=17, y;
6     int * p;
7
8     p = &x;
9     printf("El valor de x es %d ", *p);
10    y=*p+3;
11    printf("El valor de y es %d \n ", y);
12 }

mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ gcc apuntador_1.c -o cd
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ ./cd
El valor de x es 17 El valor de y es 20

```

Cuando se declara un apuntador, posee un valor cualquiera que no se puede conocer con antelación. Después de que ha sido inicializado, la dirección que posee puede dejar de ser válida por que la variable asociada termina su ámbito o por que ese espacio de memoria fue reservado dinámicamente.

Declaramos el apuntador, lo que se muestra al usuario es el valor de `y` y donde es igual al apuntador de valor 23, por eso se muestra el numero 23, el apuntador de nombre `p` es 25 por la parte en como esta ubicado en el código e imprimimos la dirección en donde se aloja el valor de `p`.

```
Welcome | C estruc_medalla.c | C estruc_mul.c | C estruc_frac.c | C apuntador_3.c
home > mary > Escritorio > c > C apuntador_2.c > func()
1  #include <stdlib.h>
2  #include <string.h>
3  int*p, y;
4  void func() {
5      int x=40;
6      p=&x;
7      y=*p;
8      *p=23;
9  }
10
11 int main(void){
12     func();
13     y=*p;
14     *p=25;
15     printf (" El valor de y es %d \nEl valor de *p es %d \n El valor de p es %d\n", y, *p, p);
16 }
```

OUTPUT | **TERMINAL** | DEBUG CONSOLE | PROBLEMS

```
El valor de p es 0x7fff53d6afa4mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ ./a2
El valor de y es 23
El valor de *p es 25
El valor de p es 0x7ffefb286284mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ ./a2
```

Dado que un apuntador es una variable que apunta a otra, fácilmente se puede deducir que pueden existir apuntadores a apuntadores, y a su vez los segundos pueden apuntar a apuntadores.

Es posible declarar apuntadores a constantes. De esta manera, no se permite la modificación de la dirección almacenada en el apuntador, pero si se permite la modificación del valor al que apunta.

Dentro de la función la dirección se utiliza para acceder al argumento real. En las llamadas por referencia cualquier cambio en la función tiene efecto sobre la variable cuya dirección se pasó como argumento.

```
Welcome | C estruc_mul.c | C estruc_frac.c | C apuntador_4.c | C apuntador_5.c
home > mary > Escritorio > c > C apuntador_4.c > main(void)
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void){
4      int x = 2;
5      int y = 5;
6      printf ("Antes x = %d, y = %d \n", x, y);
7      intercambio(&x,&y);
8      printf ("Despues x = %d, y = %d \n",x, y);
9      system("Pause");
10 }
11 void intercambio( int *a, int *b){
12     int temp;
13     temp = *b;
14     *b= *a;
15     *a = temp;
16 }
17 }
```

OUTPUT | **TERMINAL** | DEBUG CONSOLE | PROBLEMS

```
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ gcc apuntador_4.c -o a4
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ ./a4
Antes x = 2, y = 5
Despues x = 5, y = 2
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ gcc apuntador_4.c -o a4
```

La función `sizeof()` devuelve el tamaño en bytes que ocupa un tipo o variable en memoria. Por ello declaramos un array con los valores declarados, tomamos la longitud en bytes del array entre los bytes que es un entero imprimimos los bytes del array, cuantos enteros tiene un byte, cuantos elementos tiene un array.

```
Welcome | C estruc_mul.c | C estruc_frac.c | C apuntador_2.c | C apuntador_5.c
home > mary > Escritorio > c > C apuntador_5.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  int main(){
5      int array[10]={1,2,3,4,5,6,7,8,9,0};
6      int len=sizeof(array)/sizeof(int);
7      printf("Los bytes del arreglo son: %ld\n", sizeof(array));
8      printf("Cada entero tiene: %ld bytes\n", sizeof(int));
9      printf("El arreglo tiene: %d elementos\n", len);
10     return 0;
11
12 }

OUTPUT | TERMINAL | DEBUG CONSOLE | PROBLEMS
Cada entero tiene: 4 bytes
El arreglo tiene: 10 elementos
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ ./a5
Los bytes del arreglo son: 40
Cada entero tiene: 4 bytes
El arreglo tiene: 10 elementos
```

Los programas pueden crear variables globales o locales. Las variables declaradas globales en sus programas se almacenan en posiciones fijas de memoria (segmento de datos) y todas las funciones pueden utilizar estas variables.

La función `free()` permite liberar la memoria reservada a través de un apuntador.

`void free (void* ptr);`

La función `malloc()` reserva memoria y retorna su dirección, o retorna NULL en caso de no haber conseguido suficiente memoria.

`Void *malloc(size_t tam_bloque)`

```
Welcome | C estruc_mul.c | C estruc_frac.c | C apuntador_6.c
home > mary > Escritorio > c > C apuntador_6.c > main(void)
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  int main(void){
5      int i,n;
6      char * buffer;
7      printf (" Teclea la longitud de la cadena? ");
8      scanf ("%d", &i);
9      buffer = (char*) malloc (i+1);
10     if (buffer==NULL) exit (1);
11     for (n=0; n<i; n++)
12         buffer[n]=rand()%26+'a';
13     buffer[i]='\0';
14     printf ("Random string: %s\n",buffer);
15     free (buffer);
16 }

OUTPUT | TERMINAL | DEBUG CONSOLE | PROBLEMS
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ ./a6
Teclea la longitud de la cadena? 14
Random string: nwlrbmqbhcdar
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ ./a6
```

Crea un arreglo entero de tamaño x, en donde x es ingresado por teclado.
Llena todos los elementos del arreglo con datos ingresados por el usuario.
Muestra los valores

```
home > mary > Escritorio > c > C apuntador_n.c > main()
1  #include <stdio.h>
2  int main(){
3      int i,n;
4      puts("Ingresa el arreglo");
5      scanf("%d", &n);
6      int x[n];
7
8      for ( i = 0; i < n; i++){
9          printf("ingresa el %d = ",(i+1));
10         scanf("%d",&x[i]);
11     }
12     for(i=0; i<n; i++){
13         printf("\n el %d numero es: %d\n", (i+1),x[i]);
14         printf("\n Fin \n");
15     }
16 }
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ gcc apuntador_n.c -o an
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ ./an
Ingresa el arreglo
3
ingresa el 1 = 43
ingresa el 2 = 98
ingresa el 3 = 45

el 1 numero es: 43

el 2 numero es: 98

el 3 numero es: 45

Fin
```

En este código tenemos una muestra donde se declara un arreglo de tipo entero en el que se imprime hasta que se 10, ya que muestra su tipo de dato entero y flotante.
Por ello el mensaje lo muestra recordemos que para ello los valores deben de estar de acuerdo a su tipo de dato %d de tipo entero, %f tipo flotante.

```
home > mary > Escritorio > c > C apuntador6.c > main(void)
1  #include <stdio.h>
2  #include <stdlib.h>
3  int i[10],x;
4  float f[10];
5
6  int main (void){
7
8      printf("\t\tEntero\t\t\tFlotante\n\n");
9
10     for(x=0;x<10;x++){
11         printf("Elemento %d:\t %d\t\t%f\n",x,i[x],f[x]);
12     }
13     system("pause");
14 }
15
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ gcc apuntador6.c -o aaa
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ ./aaa
Entero      Flotante

Elemento 0:  0      0.000000
Elemento 1:  0      0.000000
Elemento 2:  0      0.000000
Elemento 3:  0      0.000000
Elemento 4:  0      0.000000
Elemento 5:  0      0.000000
Elemento 6:  0      0.000000
Elemento 7:  0      0.000000
Elemento 8:  0      0.000000
Elemento 9:  0      0.000000
sh: 1: pause: not found
```

Crea un arreglo entero de tamaño x, en donde x es ingresado por teclado.
Llena todos los elementos del arreglo con datos ingresados por el usuario usando apuntadores.

```
C apuntador_n.c  C apuntador6.c  C apuntador_nA.c X  C apun
home > mary > Escritorio > c > C apuntador_nA.c > main()
1  #include <stdio.h>
2  int main()
3  {
4      int i,n;
5      puts("Ingresa el arreglo");
6      scanf("%d", &n);
7      int x[n];
8      int *a;
9      a=x;
10
11     for ( i = 0; i < n; i++){
12         printf("ingresa el %d = ",(i+1));
13         scanf("%d",&x[i]);
14     }
15     for(i=0; i<n; i++)
16         printf("\n el %d numero es: %d\n", (i+1),x[i]);
17     printf("\n Fin \n");
18     return 0 ;
19 }
```

OUTPUT TERMINAL DEBUG CONSOLE ... 1: bash

```
Ingresa el arreglo
3
ingresa el 1 = 12
ingresa el 2 = 32
ingresa el 3 = 11

el 1 numero es: 12
el 2 numero es: 32
el 3 numero es: 11

Fin
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$
```

De todo lo visto anterior se crea un código donde hacemos uso de apuntadores, además de `buffer free()` permite liberar la memoria reservando a través de un apuntador además la función `malloc()` reserva memoria y retorna su dirección, el usuario ingresa la longitud del arreglo, se ocupan los valores anteriores, ingresa cada valor y esto se imprimen después.

```
Welcome  C estruc_frac.c  C apuntador_8.c
home > mary > Escritorio > c > C apuntador_8.c > ...
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main (void){
4      int i,n;
5      char *buffer,*p_buffer;
6
7      printf("Teclea la longitud del arreglo ");
8      scanf("%d",&n);
9
10     buffer=malloc((n)*sizeof(int));
11     if(buffer==NULL)exit(1);
12     p_buffer=buffer;
13     for(i=0;i<n;i++){
14         printf("Ingresa el valor %d\n",i);
15         scanf("%s",p_buffer++);
16     }
17     p_buffer=buffer;
18     printf("\nLos valores son\n");
19     for(n=0;n<i;n++){
20         printf("arreglo[%d]=%d\n",n,*p_buffer);
21     }
22     free(buffer);
23     system("pause");
24 }
```

```
14     printf("Ingresa el valor %d\n",i);
15     scanf("%s",p_buffer++);
16 }
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ ./a8
Teclea la longitud del arreglo 4
Ingresa el valor 0
12
Ingresa el valor 1
11
Ingresa el valor 2
43
Ingresa el valor 3
54

Los valores son
arreglo[0]=49
arreglo[1]=49
arreglo[2]=49
arreglo[3]=49
```

Crea un arreglo de tipo char de tamaño x, en donde x es ingresado por teclado.
Llena elemento por elemento del arreglo con letras ingresados por el usuario.
Muestra el arreglo impreso en forma inversa.
Todo debe ser manejado con apuntadores.

```
C apuntador_nC.c  C apuntador_nA.c X
home > mary > Escritorio > c > C apuntador_nA.c > main(void)
2  #include <stdlib.h>
3  int main(void){
4      int i,n;
5      int *buffer,* x_buffer;
6      puts("Ingrese el tamaño de la cadena");
7      scanf("%d",&n);
8
9      char x[n];
10     buffer=(int*)malloc((n)*sizeof(int));
11     if(buffer==NULL)exit (1);
12     x_buffer=buffer;
13     for ( i = 0; i <n; i++){
14         printf("Ingrese el espacio %d:",i);
15         scanf("%s",&x[i]);
16     }
17     for ( i = n; i >=0; i--){
18         printf("%c",x[i]);
19     }
20     printf("\n Palabra en orden= %s \n",x);
21     free(buffer);
22 }
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
mary@mary-HP-240-G3-Notebook-PC:~/Escritorio/c$ ./nA
Ingrese el tamaño de la cadena
4
Ingrese el espacio 0:h
Ingrese el espacio 1:o
Ingrese el espacio 2:l
Ingrese el espacio 3:a
aloh
Palabra en orden= hola
```

V. Conclusiones:

Los apuntadores funciona de una estructura totalmente diferente en el cual podemos seguir usando el código por partes para una mejor estructura ademas de que los datos se pueden pasar por parámetros, los apuntadores e declaran por un *p con el nombre del apuntador, cuentan con su propia dirección de memoria, antes esta el tipo de dato.

Se enlazan a los datos en específicos con su declaración, existe apuntadores de estructuras mas complejas como struct, ficheros, la dirección se hace por medio de &.

La función free() permute liberar espacio en memoria a través de un apuntador malloc() reserva memoria sin importar el tipo de datos que almacenará en ella.