

Nombre de la práctica	APUNTADORES Y ARITMETICA DE OPERADORES			No.	15
Asignatura:	MÉTODOS NUMÉRICOS	Carrera:	ING. SISTEMAS COMPUTACIONALES	Duración de la práctica (Hrs)	10

I. Competencia(s) específica(s):

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

Otro

III. Material empleado:

Dev C++

IV. Desarrollo de la práctica:

APUNTADORES:

° Un puntero es un objeto que apunta a otro objeto.

Es decir, una variable cuyo valor es la dirección de memoria de otra variable.

Dirección	Etiqueta	Contenido
...		
...		
1502	x	25
1503		
1504		
...		
...		

En C no se debe indicar numéricamente la dirección de la memoria, si no que se usa una etiqueta que conocemos como **variable**.

Las direcciones de memoria dependen de la Arquitectura del ordenador y de la gestión que el Sistema operativo haga de ella.

¿Cómo se declaran los apuntadores?

Para declarar un apuntador se especifica el tipo de dato al que apunta, el operador '*', y el nombre del apuntador.

Un puntero tiene su propia dirección de memoria.

La sintaxis es la siguiente:

<tipo de dato apuntador> * <identificador del apuntador>

```
int* punt;
char* car;
float* num;
```

Al igual que el resto de las variables, los apuntadores se enlazan a tipos de datos específicos, de manera que a un apuntador sólo se le puede asignar direcciones de variables del tipo especificado en la declaración.

```
int* punt;
char* car;
float* num;
```

Tipos de apuntadores

Hay tantos tipos de apuntadores como tipos de datos.

Se puede también declarar apuntadores a estructuras más complejas.

Funciones
Struct
Ficheros

Se pueden declarar punteros vacíos o nulos.

¿Qué es la referenciación?

La referenciación es obtener la dirección de una variable.

Se hace a través del operador '&', aplicado a la variable a la cual se desea saber su dirección

```
&x ; //La dirección de la variable
```

No hay que confundir una dirección de memoria con el contenido de esa dirección de memoria.

```
x = 25;           //El contenido de la variable
&x = 1502;       //La dirección de la variable
```

Fragmento de código –referenciación.

```
int dato;           //variable que almacenará un carácter.
int *punt;         //declaración de puntero a carácter.
punt = &dato;      //en la variable punt guardamos la dirección
                  //de memoria de la variable dato;
                  // punt apunta a dato.
```

Dirección	Etiqueta	Contenido	Dirección	Etiqueta	Contenido
...			...		
...			...		
1502	dato	/0	1502	dato	/0
1503	*punt		1503	*punt	1502
1504			1504		
...			...		

¿Qué es la desreferenciación?

Es la obtención del valor almacenado en el espacio de memoria donde apunta un apuntador.

Se hace a través del operador "*", aplicado al apuntador que contiene la dirección del valor.

*p ; //El contenido de p

Fragmento de código –referenciación

```
Int x=17, y;  
Int * p;  
p = &x;  
printf ("El valor de x es %d", *p);  
y=*p+3;  
printf ("El valor de y es %d", *p);
```

Dirección	Etiqueta	Contenido
...		
...		
1502	x	17
1503	y	
1504		
...		

Dirección	Etiqueta	Contenido
...		
...		
1502	x	17
1503	y	
1504	*p	
...		

Dirección	Etiqueta	Contenido
...		
...		
1502	x	17
1503	y	
1504	*p	1502
...		

Dirección	Etiqueta	Contenido
...		
...		
1502	x	17
1503	y	
1504	*p	1502
...		

El valor de x es 17
Presione cualquier tecla para continuar...

Dirección	Etiqueta	Contenido
...		
...		
1502	x	17
1503	y	20
1504	*p	1502
...		

El valor de x es 17
El valor de y es 20
Presione cualquier tecla para continuar...

Asignación de apuntadores

Aun apuntador se pueden asignar direcciones de variables a través del operador de referenciación

('&') o direcciones almacenadas en otros apuntadores.

Direcciones inválidas

Un apuntador puede contener una dirección inválida por:

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 porque ese espacio de memoria fue reservado dinámicamente.

Ejemplo 1:

```
Welcome  C apuntadores1.c X
home > yleroy > Visual Code > C apuntadores1.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int *p,y;
5  void func(){
6      int x=40;
7      p=&x; //correcto
8      y=*p; //correcto
9      *p=23;
10 }
11 int main (void){
12     func();
13     y=*p; //incorrecto
14     *p=25; //incorrecto
15     printf("EL VALOR DE y ES: %d \n",y);
16     printf("EL VALOR DE *p ES: %d \n",*p);
17     printf("EL VALOR DE p ES: %p\n",p);
18 }

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
yleroy@yleroy-Latitude-7280:~/Visual Code$ gcc apuntadores1.c -o uno
yleroy@yleroy-Latitude-7280:~/Visual Code$ ./uno
EL VALOR DE y ES: 23
EL VALOR DE *p ES: 25
EL VALOR DE p ES: 0x7ffc411ed384
yleroy@yleroy-Latitude-7280:~/Visual Code$
```

La dirección NULL

Cuando no se desea que el apuntador apunte a algo, se le suele asignar el valor de NULL, en cuyo caso se dice que el apuntador es nulo (no apunta a nada).

NULL es una macro típicamente definida en archivos de cabecera como `stddef.h` y `stdlib.h`.

Se utiliza para proporcionar a un programa un medio de conocer cuándo un apuntador contiene una dirección inválida.

Apuntadores a apuntadores

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.

```
char c = 'z';  
char *pc = &c;  
char **ppc = &pc;  
char ***pppc = &ppc;  
***pppc = 'm'
```

Dirección	Etiqueta	Contenido
...		
1502	c	z
1503	pc	1502
1504	ppc	1503
1505	pppc	1504
...		
...		

Apuntadores constantes

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.

```
intx = 5, y = 7;  
int*constp = &x;  
*p=3;  
p=&y;
```

Dirección	Etiqueta	Contenido
...		
1502	x	5
1503	y	7
1504	*p	1502
1505		
...		
...		

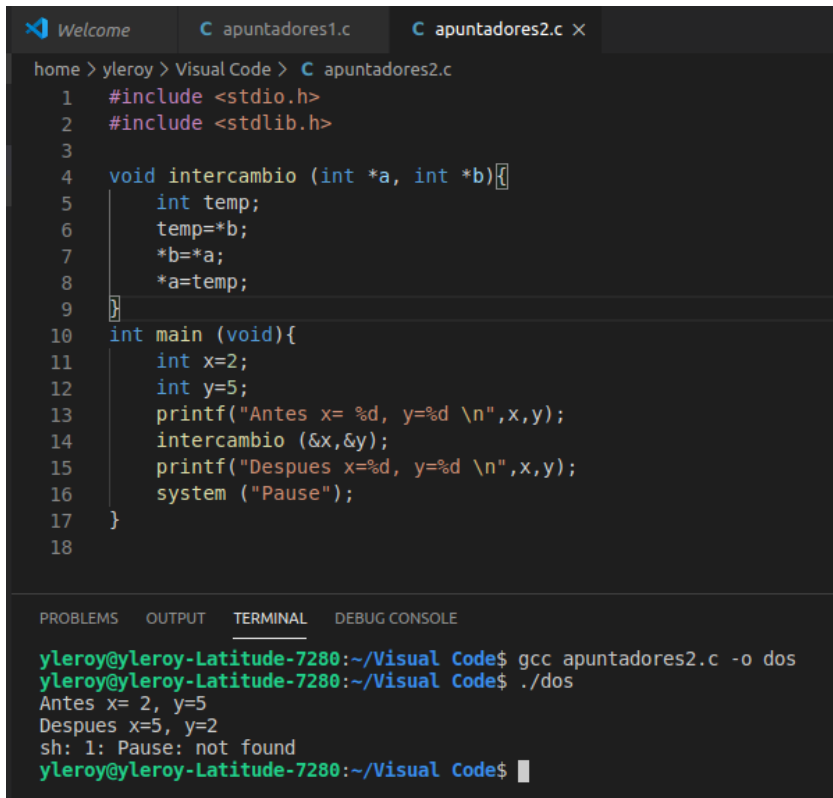
Paso de parámetros por referencia

En este tipo de llamadas los argumentos contienen direcciones de variables.

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.

Ejemplo 2:



```
home > yleroy > Visual Code > C apuntadores2.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void intercambio (int *a, int *b){
5      int temp;
6      temp=*b;
7      *b=*a;
8      *a=temp;
9  }
10 int main (void){
11     int x=2;
12     int y=5;
13     printf("Antes x= %d, y=%d \n",x,y);
14     intercambio (&x,&y);
15     printf("Despues x=%d, y=%d \n",x,y);
16     system ("Pause");
17 }
18
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
yleroy@yleroy-Latitude-7280:~/Visual Code$ gcc apuntadores2.c -o dos
yleroy@yleroy-Latitude-7280:~/Visual Code$ ./dos
Antes x= 2, y=5
Despues x=5, y=2
sh: 1: Pause: not found
yleroy@yleroy-Latitude-7280:~/Visual Code$
```

La función sizeof()

Devuelve el tamaño en bytes que ocupa un tipo o variable en memoria.

```
char cadena [10];
```

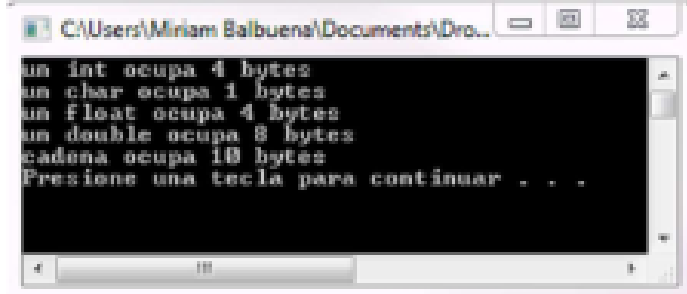
```
printf ("un int ocupa %d bytes", sizeof(int));
```

```
printf ("un char ocupa %d bytes", sizeof(char));
```

```
printf ("un float ocupa %d bytes", sizeof(float));
```

```
printf ("un doublé ocupa %d bytes", sizeof(double));
```

```
printf ("cadena ocupa %d bytes", sizeof(cadena));
```



```
C:\Users\Miniam Balbuena\Documents\Doc...
un int ocupa 4 bytes
un char ocupa 1 bytes
un float ocupa 4 bytes
un double ocupa 8 bytes
cadena ocupa 10 bytes
Presione una tecla para continuar . . .
```

Ejemplo 3:

```
Visual Studio Code
C apuntadores1.c  C apuntadores2.c  C apuntadores3.c x

home > yleroy > Visual Code > C apuntadores3.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
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
8      printf("Los bytes del arreglo son: %ld \n",sizeof(array));
9      printf("Cada entero tiene: %ld bytes\n",sizeof(int));
10     printf("El arreglo tiene %d elementos \n",len);
11     system ("pause");
12     return 0;
13 }
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
yleroy@yleroy-Latitude-7280:~/Visual Code$ gcc apuntadores3.c -o tres
yleroy@yleroy-Latitude-7280:~/Visual Code$ ./tres
Los bytes del arreglo son: 40
Cada entero tiene: 4 bytes
El arreglo tiene 10 elementos
sh: 1: pause: not found
yleroy@yleroy-Latitude-7280:~/Visual Code$
```

Asignación dinámica de memoria

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.

Las variables locales se almacenan en la pila (stack) y existen solo mientras están activas las funciones donde están declaradas.

En ambos casos el espacio de almacenamiento se reserva en el momento de compilación del programa.

Para asignar memoria dinámicamente se utilizan las funciones malloc() y free(), definidas típicamente en el archivo stdlib.h.



free()

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

void free (void* ptr);

ptr es un puntero de cualquier tipo que apunta a un área de memoria reservada previamente con **malloc**.

malloc()

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

void*malloc(size_ttam_bloque)

malloc() reserva memoria sin importar el tipo de datos que almacenará en ella.

Ejemplo 4:

```
C apuntadores4.c x
home > yleroy > Visual Code > C apuntadores4.c

4  int main (void){
5      int i,n;
6      char *buffer;
7
8      printf("Teclea la longitud de la cadena");
9      scanf("%d",&i);
10
11     buffer=(char*)malloc(i+1);
12     if(buffer==NULL)exit(1);
13     for (n=0;n<i;n++)
14         buffer[n]=rand()%26+'a';
15         buffer[i]='\0';
16     printf("Random string:%s\n",buffer);
17     free(buffer);
18     system ("pause");
19 }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
yleroy@yleroy-Latitude-7280:~/Visual Code$ gcc apuntadores4.c -o cuatro
yleroy@yleroy-Latitude-7280:~/Visual Code$ ./cuatro
Teclea la longitud de la cadena2
Random string:nw
sh: 1: pause: not found
yleroy@yleroy-Latitude-7280:~/Visual Code$
```

Ejercicio 5:

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.

```
C apuntadores4.c  C apuntadores5.c ●
home > yleroy > Visual Code > C apuntadores5.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main (void){
4      int i,n;
5      printf("INGRESA UNA CANTIDAD DEL ARREGLO: ");
6      scanf("%d",&n);
7      int x[n];
8      for (i=0;i<n;i++){
9          printf("INGRESA EL  %d NUMERO = ",(i+1));
10         scanf("%d",&x[i]);
11     }
12     for(i=0;i<n;i++)
13         printf("\nEL %d NUMERO ES: %d\n",(i+1),x[i]);
14     return 0;
15 }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
yleroy@yleroy-Latitude-7280:~/Visual Code$ gcc apuntadores5.c -o cinco
yleroy@yleroy-Latitude-7280:~/Visual Code$ ./cinco
INGRESA UNA CANTIDAD DEL ARREGLO: 4
INGRESA EL  1 NUMERO = 3
INGRESA EL  2 NUMERO = 4
INGRESA EL  3 NUMERO = 5
INGRESA EL  4 NUMERO = 6

EL 1 NUMERO ES: 3

EL 2 NUMERO ES: 4

EL 3 NUMERO ES: 5

EL 4 NUMERO ES: 6
yleroy@yleroy-Latitude-7280:~/Visual Code$
```


ARITMETICA DE OPERADORES

Incremento de operadores

Cuando se incrementa un apuntador se está incrementando su valor.

Si incrementamos en 1 el valor del apuntador, C sabe el tipo de dato al que apunta e incrementa la dirección guardada en el apuntador en el tamaño del tipo de dato.

Arreglos

```
miArreglo= 1000
p_miArreglo= miArreglo
&miArreglo[0] = 1000
&miArreglo[1] = 1004
```

1000		
1001	miArreglo[0]	
1002		
1003		
1004		
1005	miArreglo[1]	
1006		
1007		
1008		
1009	miArreglo[2]	
1010		
1011		
1012		
1013	miArreglo[3]	
1014		
1015		
1016		
	p_miArreglo	1000

```
miArreglo= 1000
p_miArreglo= miArreglo
&miArreglo[0] = 1000
&miArreglo[1] = 1004
p_miArreglo++;
```

1000		
1001	miArreglo[0]	
1002		
1003		
1004		
1005	miArreglo[1]	
1006		
1007		
1008		
1009	miArreglo[2]	
1010		
1011		
1012		
1013	miArreglo[3]	
1014		
1015		
1016		
	p_miArreglo	1004

```
miArreglo= 1000
p_miArreglo= miArreglo
&miArreglo[0] = 1000
&miArreglo[1] = 1004
p_miArreglo+= 2;
```

1000	miArreglo[0]	
1001		
1002		
1003		
1004	miArreglo[1]	
1005		
1006		
1007		
1008	miArreglo[2]	
1009		
1010		
1011		
1012	miArreglo[3]	
1013		
1014		
1015		
1016	p_miArreglo	1004

```
miArreglo= 1000
p_miArreglo= miArreglo
&miArreglo[0] = 1000
&miArreglo[1] = 1004
p_miArreglo--;
```

1000	miArreglo[0]	
1001		
1002		
1003		
1004	miArreglo[1]	
1005		
1006		
1007		
1008	miArreglo[2]	
1009		
1010		
1011		
1012	miArreglo[3]	
1013		
1014		
1015		
1016	p_miArreglo	1000

Ejercicio 7:

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 apuntadores6.c  C apuntadores7.c X
home > yleroy > Visual Code > C apuntadores7.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main (void){
5      int i,n;
6
7      printf("INGRESA UNA CANTIDAD CON NUMEROS ENTEROS: ");
8      scanf("%d",&n);
9      int x[n];
10
11     for (i=0;i<n;i++){
12         printf("INGRESA UN NUMERO %d=", (i+1));
13         scanf("%d",&x[i]);
14     }
15     for(i=0;i<n;i++)
16         printf("\nEL %d NUMERO ES:%d\n", (i+1),x[i]);
17     return 0;
18 }
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
yleroy@yleroy-Latitude-7280:~/Visual Code$ gcc apuntadores7.c -o siete
yleroy@yleroy-Latitude-7280:~/Visual Code$ ./siete
INGRESA UNA CANTIDAD CON NUMEROS ENTEROS: 4
INGRESA UN NUMERO 1=3
INGRESA UN NUMERO 2=4
INGRESA UN NUMERO 3=5
INGRESA UN NUMERO 4=6

EL 1 NUMERO ES:3

EL 2 NUMERO ES:4

EL 3 NUMERO ES:5

EL 4 NUMERO ES:6
yleroy@yleroy-Latitude-7280:~/Visual Code$
```

Ejemplo 8:

```
apuntadores6.c  apuntadores7.c  apuntadores8.c  Extension:
home > yleroy > Visual Code > C apuntadores8.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main (void){
4      int i,n;
5      char *buffer,*p_buffer;
6      printf("Teclea la longitud del arreglo: ");
7      scanf("%d",&n);
8      buffer=(int*)malloc((n)* sizeof(int));
9      if(buffer==NULL) exit(1);
10     p_buffer=buffer;
11     for (i=0;i<n;i++){
12         printf("INGRESA EL VALOR %d\n",i);
13         scanf("%d",p_buffer++);
14     }
15     p_buffer=buffer;
16     printf("\nLos valores son\n");
17     for(n=0;n<i;n++){
18         printf("\narreglo[%d]=%d\n",n,*p_buffer++);
19     }
20     free(buffer);
21     system ("pause");
22 }
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
yleroy@yleroy-Latitude-7280:~/Visual Code$ ./ocho
Teclea la longitud del arreglo: 3
INGRESA EL VALOR 0
4
INGRESA EL VALOR 1
5
INGRESA EL VALOR 2
6

Los valores son
arreglo[0]=4
arreglo[1]=5
arreglo[2]=6
sh: 1: pause: not found
yleroy@yleroy-Latitude-7280:~/Visual Code$
```

Ejercicio 9:

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.

```
apuntadores6.c  apuntadores7.c  apuntadores8.c  apuntadores9.c
home > yleroy > Visual Code > C apuntadores9.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main ()
4  {
5      int i,n,letra;
6      char *buffer,*p_buffer;
7      puts("INGRESA UNA CANTIDAD DE LETRAS");
8      scanf("%d",&n);
9      buffer=(char*)malloc((n)* sizeof(char));
10     if(buffer==NULL){
11         exit(1);
12     }
13     p_buffer=buffer;
14     for (i=0;i<n;i++){
15         puts(" ");
16         scanf("%c",p_buffer);
17         printf("INGRESA UNA LETRA %d\n",i+1);
18         scanf("%c",p_buffer++);
19     }
20     p_buffer=buffer;
21     printf("\nTU PALABRA ES:%s\n",p_buffer);
22     puts("TU PALABRA INVERTIDA ES:");
23     for(i=(n-1);i>=0;i--){
24         printf("%c",p_buffer[i]);
25     }
26     puts("\n");
27     free(buffer);
28     return 0;
29 }
```

```
yleroy@yleroy-Latitude-7280:~/Visual Code$ ./nuevo
INGRESA UNA CANTIDAD DE LETRAS
3

INGRESA UNA LETRA 1
o

INGRESA UNA LETRA 2
l

INGRESA UNA LETRA 3
a

TU PALABRA ES:ola
TU PALABRA INVERTIDA ES:
alo

yleroy@yleroy-Latitude-7280:~/Visual Code$
```