

Nombre de la práctica	Ejercicios en Lenguaje C			No.	4
Asignatura:	Métodos Numéricos	Carrera:	ISIC	Duración de la práctica (Hrs)	

Nombre: **Rodrigo Javier Martínez**

Grupo: **3041**

## I. Competencia(s) específica(s):

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

- Aula y casa

## III. Material empleado:

- Dev-C++
- Word

## IV. Desarrollo de la práctica:

Esta es la continuación del curso de Lenguaje C que llevamos a cabo en Métodos Numéricos, en donde estamos realizando ejercicios en este lenguaje.

Para comenzar vamos a ver que es un apuntador, es un objeto que apunta a otro objeto. Es decir, una variable cuyo valor es la dirección de memoria de otra variable.

- ¿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>

- 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 diferenciación?

La diferenciación 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

- ¿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

- 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 por que ese espacio de memoria fue reservado dinámicamente.

Ejemplo:

```
C apun1.c > main(void)
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 }
```

OUTPUT   DEBUG CONSOLE   PROBLEMS   TERMINAL

```
rodrigo@rodrigo-Lenovo:~/Documentos/Ejercicios$ gcc apun1.c -o apun1.exe
rodrigo@rodrigo-Lenovo:~/Documentos/Ejercicios$ ./apun1.exe
El valor de y es 23
El valor de *p es 25
El valor de p es 0x7ffeed2120e4
rodrigo@rodrigo-Lenovo:~/Documentos/Ejercicios$
```

- 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.

## Paso de parámetros por referencia

Ejemplo:

```
C apun2.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     return 0;
18 }
```

OUTPUT DEBUG CONSOLE PROBLEMS TERMINAL 1: bash

```
rodrigo@rodrigo-Lenovo:~/Documentos/EjerciciosC$ gcc apun2.c -o apun2.exe
rodrigo@rodrigo-Lenovo:~/Documentos/EjerciciosC$ ./apun2.exe
Antes x = 2, y = 5
Despues x = 5, y = 2
sh: 1: pause: not found
```

- Función sizeof

Devuelve el tamaño en Bytes que ocupa un tipo de variable en memoria.

Ejemplo:

```
C apun3.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 }
14
```

OUTPUT DEBUG CONSOLE PROBLEMS TERMINAL 1: bash

```
rodrigo@rodrigo-Lenovo:~/Documentos/EjerciciosC$ gcc apun3.c -o apun3.exe
rodrigo@rodrigo-Lenovo:~/Documentos/EjerciciosC$ ./apun3.exe
Los bytes del arreglo son: 40
Cada entero tiene: 4 bytes
El arreglo tiene 10 elementos
sh: 1: Pause: not found
```

- Asignación dinámica de memoria

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_t tam_bloque)`

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

```
C apun4.c
4  int main(void){
5      int i,n;
6      char *buffer;
7
8      printf("Ingresa la longitud de la cadena: ");
9      scanf("%d",&i);
10     buffer = (char*)malloc(i+1);
11     if(buffer == NULL)exit(1);
12     for(n=0;n<i;n++)
13         buffer[n]= rand()%26+'a';
14     buffer[i]= '\0';
15     printf("Random string:%s\n",buffer);
16     free(buffer);
17     system ("pause");
18 }
19
```

OUTPUT DEBUG CONSOLE PROBLEMS TERMINAL 1: bash

```
rodrigo@rodrigo-Lenovo:~/Documentos/EjerciciosC$ gcc apun4.c -o apun4.exe
rodrigo@rodrigo-Lenovo:~/Documentos/EjerciciosC$ ./apun4.exe
Ingresa la longitud de la cadena: 10
Random string:nwlrbbmqbh
sh: 1: pause: not found
```

Ejercicio:

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 apun5.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void){
5      int i,n;
6      printf("Ingresa la cantidad del arreglo: ");
7      scanf("%d",&n);
8      int x[n];
9      for(i=0;i<n;i++){
10         printf("Ingresa el %d numero = ",(i+1));
11         scanf("%d",&x[i]);
12     }
13     for(i=0;i<n;i++){
14         printf("El %d numero es %d\n", (i+1),x[i]);
15     }
16     return 0;
17 }
```

OUTPUT

DEBUG CONSOLE

PROBLEMS

TERMINAL

1: bash

```

rodrigo@rodrigo-Lenovo:~/Documentos/EjerciciosC$ gcc apun5.c -o apun5.exe
rodrigo@rodrigo-Lenovo:~/Documentos/EjerciciosC$ ./apun5.exe
Ingresa la cantidad del arreglo: 5
Ingresa el 1 numero = 2
Ingresa el 2 numero = 9
Ingresa el 3 numero = 0
Ingresa el 4 numero = 3
Ingresa el 5 numero = 6
El 1 numero es 2
El 2 numero es 9
El 3 numero es 0
El 4 numero es 3
El 5 numero es 6
rodrigo@rodrigo-Lenovo:~/Documentos/EjerciciosC$ 1
```

- Apuntadores a arreglos

El nombre de un arreglo es simplemente un apuntador constante al inicio del arreglo.

Ejemplo:

```
C apun6.c > main(void)
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int i[10],x;
5  float f[10];
6  int main(void){
7      printf("\t\t\t\t\tEntero\t\t\tFlotante\n\n");
8      for(x=0;x<10;x++){
9          printf("Elemento %d: \t%d\t\t\t\t\t%f\n",x, &i[x], &f[x]);
10     }
11     return 0;
12 }
13
```

OUTPUT

DEBUG CONSOLE

PROBLEMS

TERMINAL

Code

	Entero	Flotante
Elemento 0:	-964018016	-964018080
Elemento 1:	-964018012	-964018076
Elemento 2:	-964018008	-964018072
Elemento 3:	-964018004	-964018068
Elemento 4:	-964018000	-964018064
Elemento 5:	-964017996	-964018060
Elemento 6:	-964017992	-964018056
Elemento 7:	-964017988	-964018052
Elemento 8:	-964017984	-964018048
Elemento 9:	-964017980	-964018044



## Ejercicio:

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 apun7.c > ...
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void){
5      int i,n;
6      printf("Ingresa la cantidad de numeros enteros que desee: ");
7      scanf("%d",&n);
8      int x[n];
9      for(i=0;i<n;i++){
10         printf("Ingresa el %d numero = ",(i+1));
11         scanf("%d",&x[i]);
12     }
13     scanf("\n");
14     for(i=0;i<n;i++)
15         printf("El %d numero es %d\n",(i+1),x[i]);
16     return 0;
17 }
```

OUTPUT DEBUG CONSOLE PROBLEMS TERMINAL 1: bash

```
rodrigo@rodrigo-Lenovo:~/Documentos/Ejercicios$ gcc apun7.c -o apun7.exe
rodrigo@rodrigo-Lenovo:~/Documentos/Ejercicios$ ./apun7.exe
Ingresa la cantidad de numeros enteros que desee: 4
Ingresa el 1 numero = 2
Ingresa el 2 numero = 9
Ingresa el 3 numero = 5
Ingresa el 4 numero = 3
El 1 numero es 2
El 2 numero es 9
El 3 numero es 5
El 4 numero es 3
rodrigo@rodrigo-Lenovo:~/Documentos/Ejercicios$
```

## Ejemplo:

```
C apun8.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void){
5      int i,n;
6      int *buffer, *p_buffer;
7
8      puts("Ingresa la longitud del arreglo");
9      scanf("%d",&n);
10     buffer = (int*) malloc((n)* sizeof(int));
11     if(buffer==NULL)exit(1);
12     p_buffer = buffer;
13
14     for(i=0;i<n;i++){
15         printf("Ingresa el valor %d \n", i);
16         scanf("%d",p_buffer);
17     }
18     p_buffer = buffer;
19     printf("\nLos valores son\n");
20     for(n=0;n<i;n++){
21         printf("arreglo[%d] = %d \n", n,*p_buffer);
22     }
23     free(buffer);
24     system("pause");
25 }
```

```
OUTPUT  DEBUG CONSOLE  PROBLEMS  TERMINAL
rodrigo@rodrigo-Lenovo:~/Documentos/EjerciciosC$ gcc apun8.c -o apun8.exe
rodrigo@rodrigo-Lenovo:~/Documentos/EjerciciosC$ ./apun8.exe
Ingresa la longitud del arreglo
4
Ingresa el valor 0
5
Ingresa el valor 1
7
Ingresa el valor 2
1
Ingresa el valor 3
3

Los valores son
arreglo[0] = 3
arreglo[1] = 3
arreglo[2] = 3
arreglo[3] = 3
sh: 1: pause: not found
rodrigo@rodrigo-Lenovo:~/Documentos/EjerciciosC$
```

## Ejercicio:

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

```
C apun9.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main (void){
5      int i,n,letra;
6      char *buffer, *p_buffer;
7      printf("Ingresa la 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("\nLa palabra es: %s\n",p_buffer);
22     puts("La 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 }
30
```



```
rodrigo@rodrigo-Lenovo:~/Documentos/EjerciciosC$ gcc apun9.c -o apun9.exe
rodrigo@rodrigo-Lenovo:~/Documentos/EjerciciosC$ ./apun9.exe
Ingresa la cantidad de letras: 4

Ingresa una letra 1
h

Ingresa una letra 2
o

Ingresa una letra 3
l

Ingresa una letra 4
a

La palabra es: hola
La palabra invertida es:
aloh
```

## V. Conclusiones:

Para concluir podemos decir que el lenguaje de programación C es uno de los más completos y complejos de utilizar ya que cuenta con muchas funciones las cuales nos pueden ayudar a resolver distintos problemas ya que los puedes implementar en la programación y sacarle provecho a esta. También agregar que el lenguaje C es la base casi toda la programación que su estructura es muy similar a las de los demás lenguajes.