Nombre de la práctica	APUNTADORES			No.	1
Asignatura:	METODOS NUMERICOS	Carrera:	ING. SISTEMAS COMPUTACIONALES	Duración de la práctica (Hrs)	6

I. Competencia(s) específica(s):

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

Otro

III. Material empleado:

VisualStudio Code

Ubuntu

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.

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.

Declaracion de un apuntador

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
...		
...		
1502	dato	/0
1503	*punt	
1504		
...		

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

punteros.c

```c
#include <stdio.h>
#include <stdlib.h>

int*p,y;

void func(){
    int x =40;
    p=&x;
    y=*p;
    *p=23;
}

int main(void){
    func();
    y=*p;
    *p=25;
    printf("El valor de y es %d \n El valor de *p es %d \nEL valor de p es %p\n",y,*p,p);
    return 0;
}
```

Al usar un nuevo SO y un nuevo Software debemos ver como es que se compila y ejecuta, para esto debemos abrir una nueva Terminal, esto puede ser desde el menu del mismo SO o desde la barra de Herramientas de VisualStudio, que es la forma en que yo lo hice.
Una vez en la Terminal debemos escribir el comando "gcc NombreDelArchivo ".c" que es nuestra extencion en este caso, despues de eso ponemos "-o nombre" para guardarlo como un ejecutable.
Posteriormete para ejecutarlo ingresamos "./nombreDelEjecutable" y listo.

```
DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL                    46: bash

emmanuel@emmanuel-Inspiron-5523:~/Documentos/C Codes$ gcc punteros.c -o Punt0
emmanuel@emmanuel-Inspiron-5523:~/Documentos/C Codes$ ./Punt0
El valor de y es 23
El valor de *p es 25
EL valor de p es 0x7ffe7a7a8294
emmanuel@emmanuel-Inspiron-5523:~/Documentos/C Codes$
```

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 stdef.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.

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

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

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.

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

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

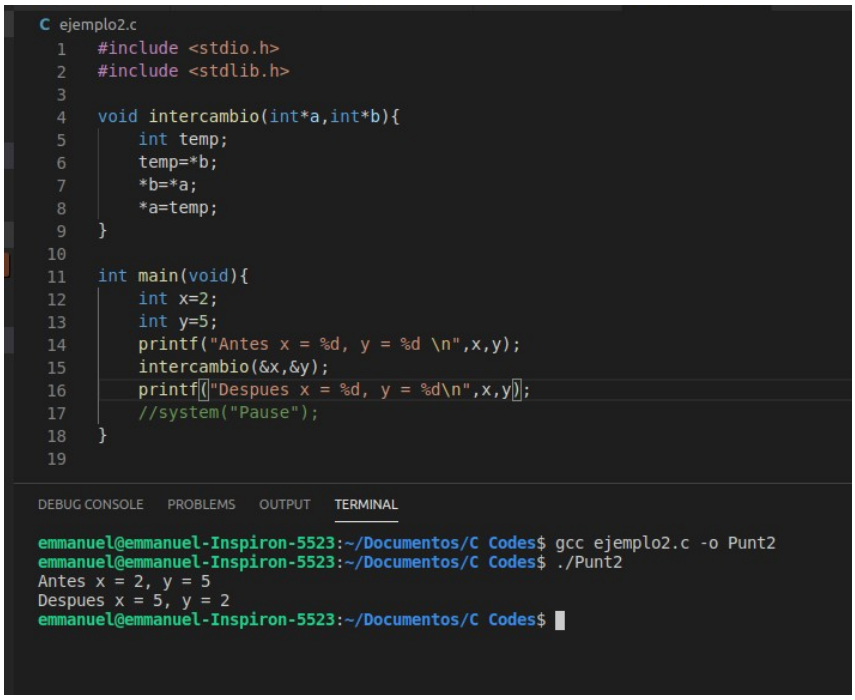
Paso de parámetros por referencia

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

Dentro de la función la dirección se utiliza para acceder al argumento real.

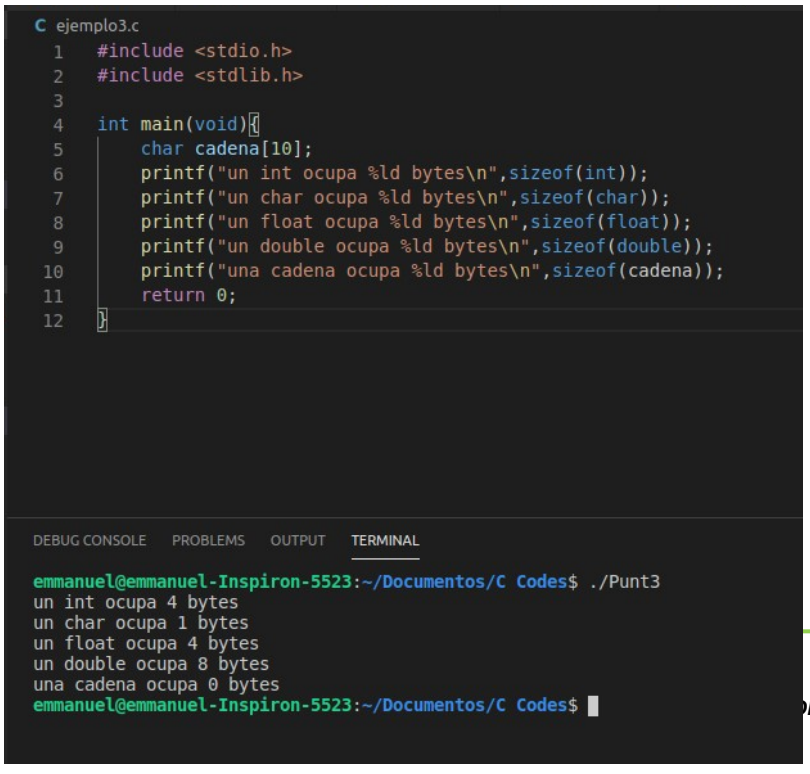
En las llamadas por referencia cualquier cambio en la función tiene efecto sobre la variable cuya dirección se pasó como argumento.

Ejemplo 2:

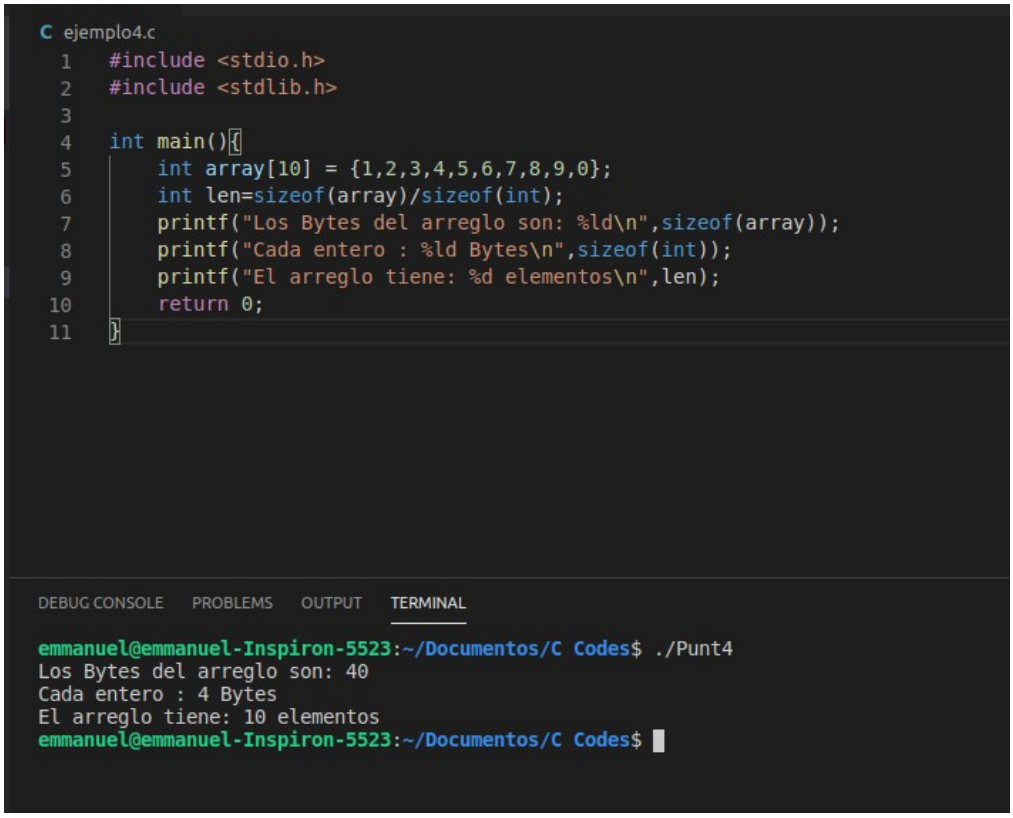


La función sizeof()

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



Ejemplo 3:



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:

ejermplo5.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(void){
    int i,n;
    char*buffer;
    printf("Teclea la longitud de la cadena: ");
    scanf("%d",&i);

    buffer=(char*)malloc(i+1);
    if(buffer==NULL)exit(1);

    for(n=0;n<i;n++)
        buffer[n]=rand()%26+'a';
    buffer[i]='\0';

    printf("Random string: %s\n",buffer);
        free(buffer);
}
```

```
DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL
emmanuel@emmanuel-Inspiron-5523:~/Documentos/C Codes$ ./Punt5
Teclea la longitud de la cadena: 5
Random string: nwlrb
emmanuel@emmanuel-Inspiron-5523:~/Documentos/C Codes$
```

Ejercicio 1

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

ejercicio1_1.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(void){
    int i,n,j,a;
    char*buffer;
    printf("Teclea la longitud de la cadena: ");
    scanf("%d",&i);

    buffer=(char*)malloc(i+1);
    if(buffer==NULL){
        exit(1);
    }

    for(n=0;n<i;n++){
        printf("Ingresa un valor\n");
        scanf("%d",&j);
        buffer[n]=j;
        buffer[i]='\0';
    }

    for(a=0;a<i;a++){
        printf(" %d ",buffer[a]);
    }

    free(buffer);
}
```

```
DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL

emmanuel@emmanuel-Inspiron-5523:~/Documentos/C Codes$ gcc ejercicio1_1.c -o EPunt1_1
emmanuel@emmanuel-Inspiron-5523:~/Documentos/C Codes$ ./EPunt1_1
Teclea la longitud de la cadena: 3
Ingresa un valor
45
Ingresa un valor
23
Ingresa un valor
12
 45  23  12 emmanuel@emmanuel-Inspiron-5523:~/Documentos/C Codes$
```

Apuntadores a arreglos

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

Int lista_arr[3]={10,20,30};
Int *lista_ptr;
lista_ptr= lista_arr;

Dirección	Etiqueta	Contenido
...		
1502	arr[0]	10
1503	arr[1]	20
1504	arr[2]	30
1505	*list_arr	1502
...		
...		

Se crea el apuntador lista_ptrpara poder ir modificándola dirección a donde apunta.

ARIMETICA DE OPERADORES

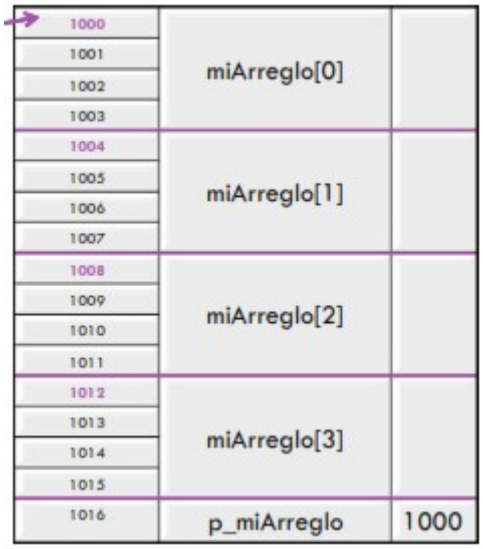
Incremento de operadores

Cuando se incrementa un apuntador se está incrementando su valor.

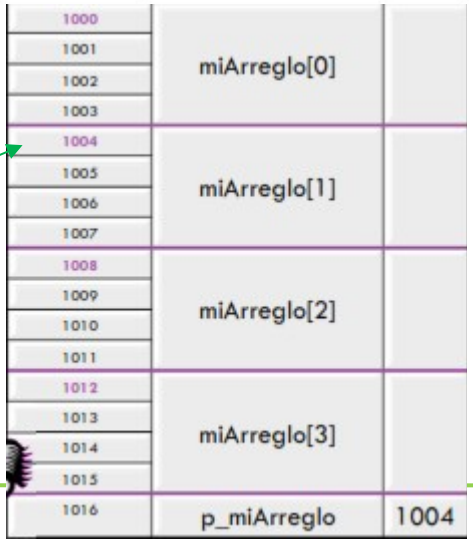
Si incrementamos en 1 el valor del apuntador, C sabe el tipo de dato al que apunta e incrementa la dirección guardada en el apuntador en el tamaño del tipo de dato.

Arreglos

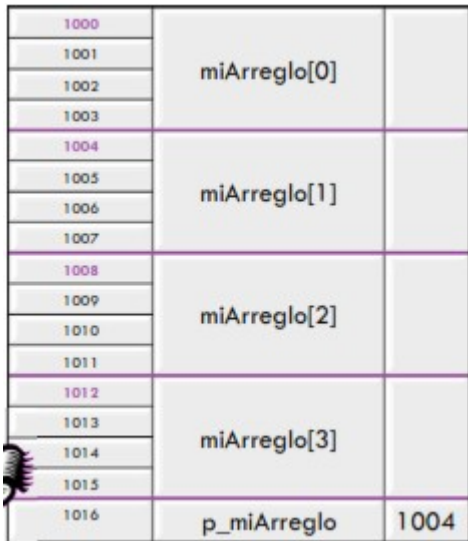
miArreglo= 1000
p_miArreglo= miArreglo
&miArreglo[0] = 1000
&miArreglo[1] = 1004



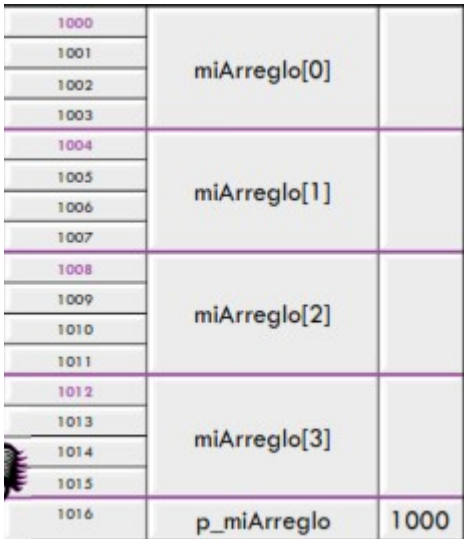
miArreglo= 1000
p_miArreglo= miArreglo
&miArreglo[0] = 1000
&miArreglo[1] = 1004
p_miArreglo++;



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



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



Ejercicio 2:
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 ejercicio1.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(void){
    int i,n;
    int*buffer,*apBuffer;

    printf("Ingresa el tamanio de la cadena: ");
    scanf("%d",&n);
    buffer=(int*)malloc((n)*sizeof(int));
    if(buffer==NULL)exit(1);

    apBuffer=buffer;
    for(i=0;i<n;i++){
        printf("Ingresa un valor\n");
        scanf("%d",apBuffer++);
    }
    printf("\n");
    free(buffer);
    return 0;
}
```

```
emmanuel@emmanuel-Inspiron-5523:~/Documentos/C Codes$ gcc ejercicio1.c -o EPunt1
emmanuel@emmanuel-Inspiron-5523:~/Documentos/C Codes$ ./EPunt1
Ingresa el tamanio de la cadena: 3
Ingresa un valor
56
Ingresa un valor
43
Ingresa un valor
23

emmanuel@emmanuel-Inspiron-5523:~/Documentos/C Codes$
```

Ejercicio 3:
- 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.

ejercicio3.c

```c
int main(){
    int i,n,palabra;
    char*buffer,*ptrBuffer;
    puts("Ingresa el numero de letras de una palabra");
    scanf("%d",&n);

    buffer=(char*)malloc(i+1);

    if(buffer==NULL){
        exit(1);
    }

    ptrBuffer=buffer;
    for(i=0;i<n;i++){
        puts("    ");
        scanf("%c",ptrBuffer);
        printf("Ingresa la palabra %d \n",i+1);
        scanf("%c",ptrBuffer++);
    }
    ptrBuffer=buffer;
    printf("La palabra que escribiste es: %s \n",ptrBuffer);
    puts("Al invertirla queda de la siguiente manera");
    for(i=(n-1);i>=0;i--){
        printf("%c",ptrBuffer[i]);
    }
    printf("\n\n");
    free(buffer);
    return 0;
}
```

```
emmanuel@emmanuel-Inspiron-5523:~/Documentos/C Codes$ gcc ejercicio3.c -o EPunt3
emmanuel@emmanuel-Inspiron-5523:~/Documentos/C Codes$ ./EPunt3
Ingresa el numero de letras de una palabra
7

Ingresa la palabra 1
m

Ingresa la palabra 2
r

Ingresa la palabra 3
m

Ingresa la palabra 4
a

Ingresa la palabra 5
m

Ingresa la palabra 6
a

Ingresa la palabra 7
n
La palabra que escribiste es: mrmaman
Al invertirla queda de la siguiente manera
namamrm

emmanuel@emmanuel-Inspiron-5523:~/Documentos/C Codes$
```