Nombre de la práctica	**APUNTADORES**			**No.**	**15**
Asignatura:	**METODOS NUMERICOS**	**Carrera:**	**ING. SISTEMAS COMPUTACIONALES**	**Duración de la práctica (Hrs)**	**10**

Nombre del alumno: Lissette Garcia Nolasco

I. Competencia(s) específica(s):

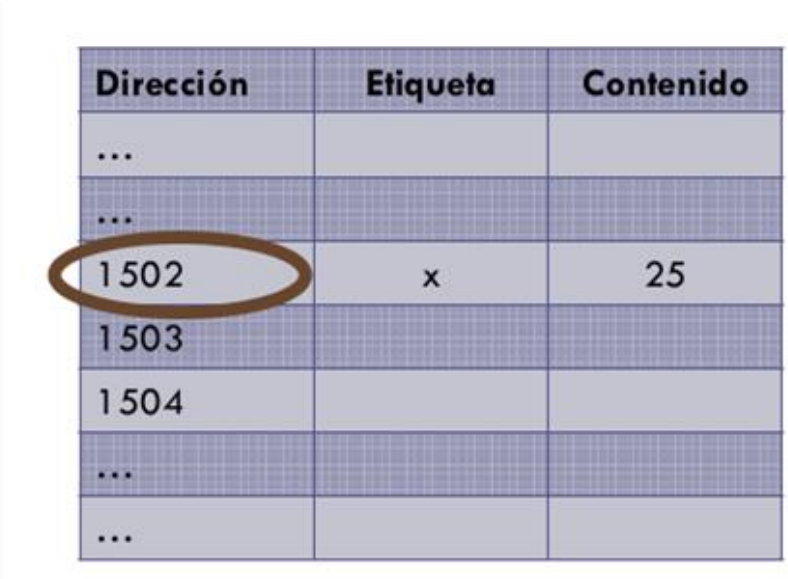
II. Lugar de realización de la práctica (laboratorio, taller, aula u otro):
Otro

III. Material empleado:
VisualStudio Code
Ubuntu

IV. Desarrollo de la práctica:

¿Qué es un apuntador?

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> *<indentificador 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.

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

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

EJEMPLO 1

```
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", y);
```

```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;
}
```

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.

```
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
...		
...		

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.

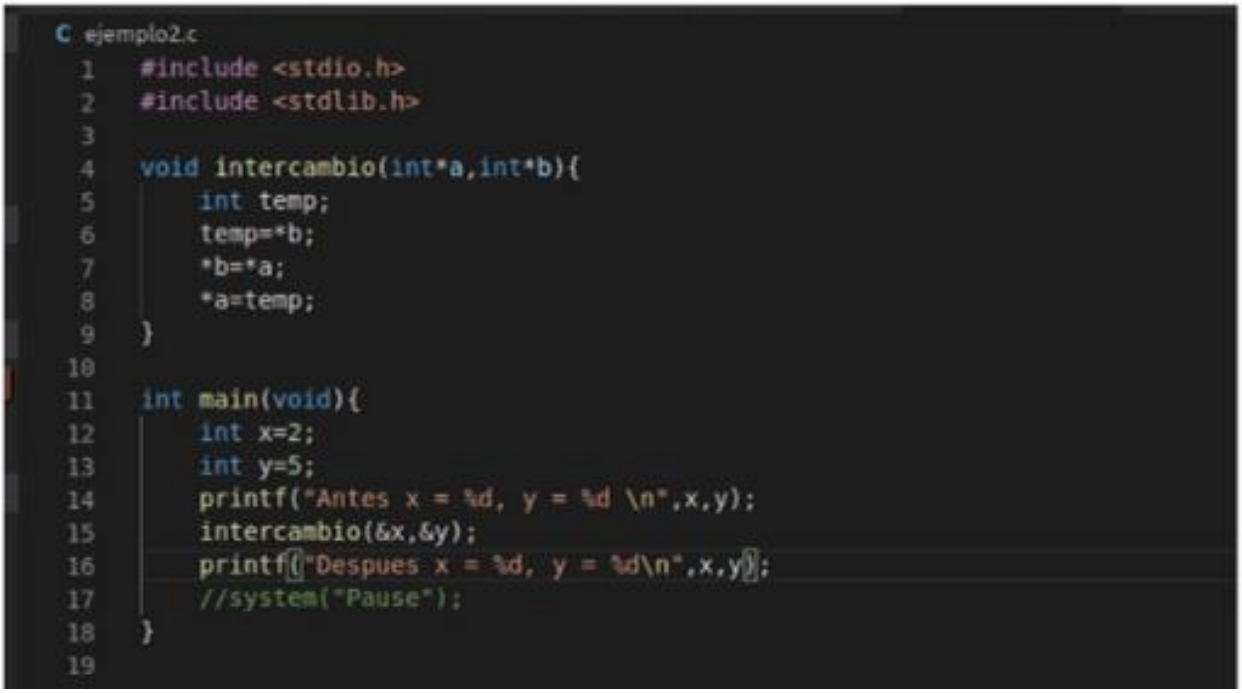
En las llamadas por referencia cualquier cambio en la función tiene efecto sobre la variable cuya dirección se pasó como argumento.

```
int main(void)
{
    int x = 2;
    int y = 5;
    printf ("Antes x = %d, y = %d \n", x, y);
    intercambio (&x, &y);
    printf ("Despues x = %d, y = %d \n", x, y
    system("Pause");
}

void intercambio(int *a, int *b){
    int temp;
    temp = *b;
    *b = *a;
    *a = temp;
}
```

Dirección	Etiqueta	Contenido
...		
1502	x	2
1503		
1504		
1505		
...		
...		

EJEMPLO 2



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

void intercambio(int*a,int*b){
    int temp;
    temp=*b;
    *b=*a;
    *a=temp;
}

int main(void){
    int x=2;
    int y=5;
    printf("Antes x = %d, y = %d \n",x,y);
    intercambio(&x,&y);
    printf("Despues x = %d, y = %d\n",x,y);
    //system("Pause");
}
```

La función sizeof()

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

char cadena [10];

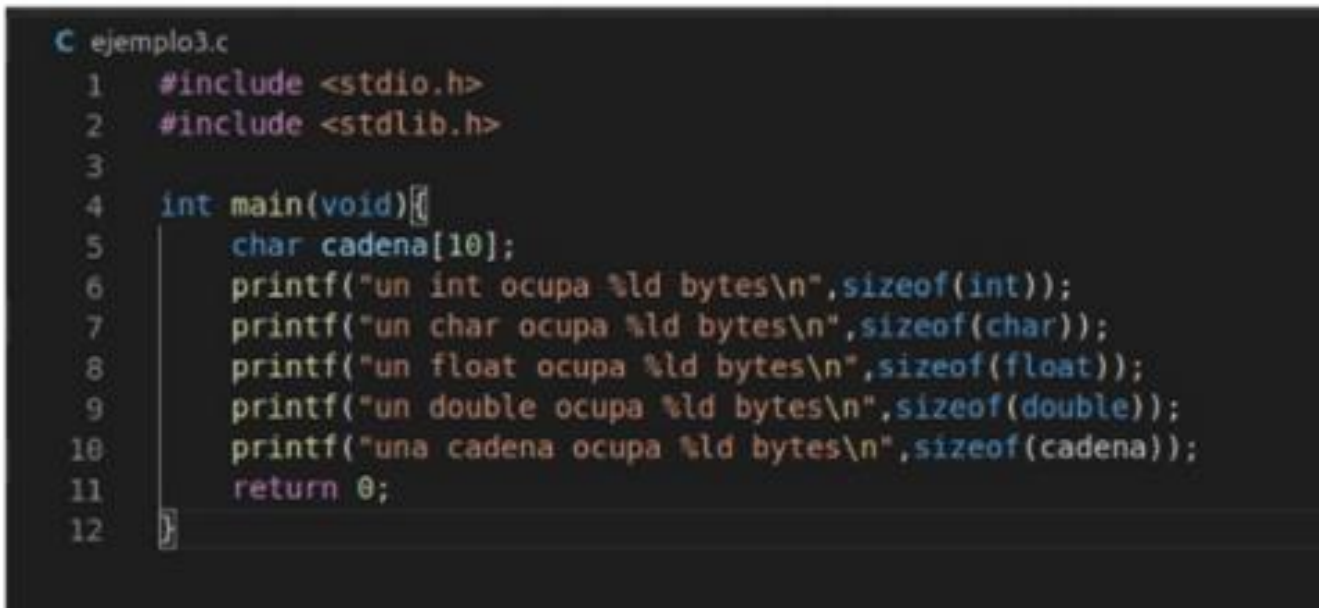
printf (“un int ocupa %d bytes”, sizeof(int));

printf (“un char ocupa %d bytes”, sizeof(char));

printf (“un float ocupa %d bytes”, sizeof(float));

printf (“un double ocupa %d bytes”, sizeof(double));

printf (“cadena ocupa %d bytes”, sizeof(cadena));



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()

57

La función free() permite liberar la memoria

reservada a través de un apuntador.

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

EJEMPLO

```
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);
    system("Pause");
}
```

Dirección	Etiqueta	Contenido
...		
1502	i	
1503	n	
1504		
1505		
1506		
...		

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);
}
```

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.

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);
}
```

Apuntadores a arreglos

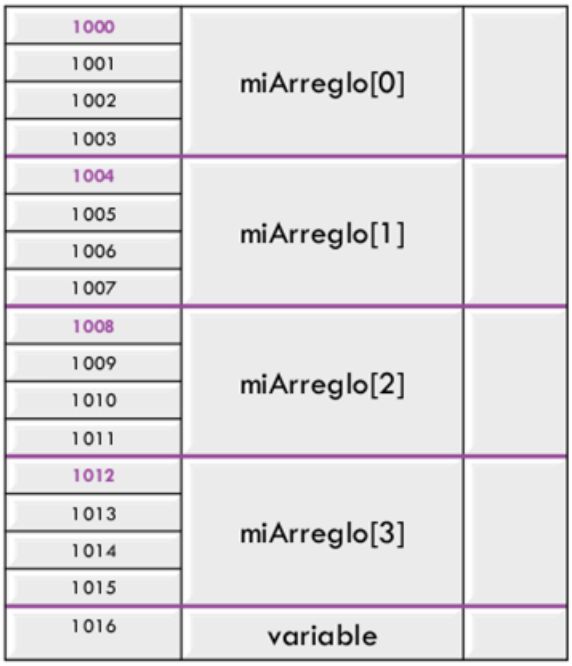
El nombre de un arreglo es simplemente un apuntador constante al inicio del arreglo

int lista_arr[3]={10,20,30};

int *lista_ptr;

lista_ptr = lista_arr;

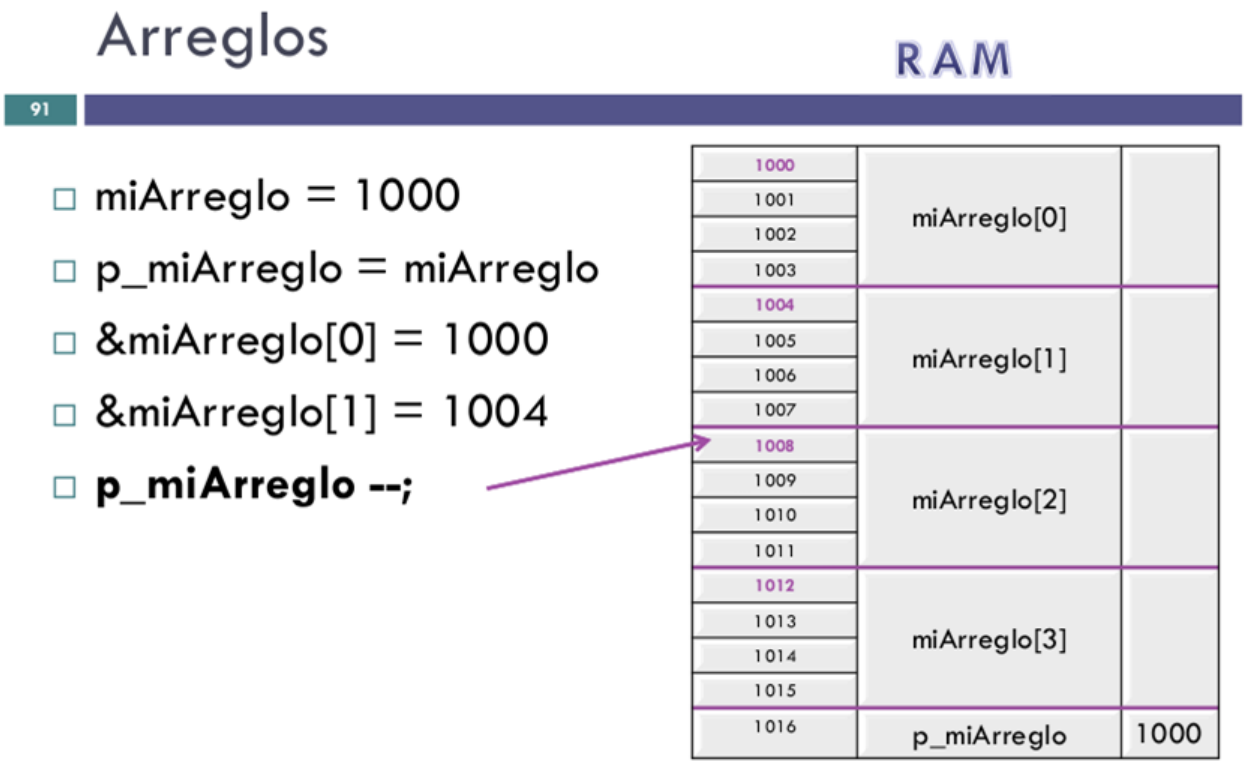
- miArreglo = 1000
- &miArreglo[0] = 1000
- &miArreglo[1] = 1004



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.



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.

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;
}
```

EJEMPLO

124

Ejemplo

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

      printf ("Teclea la longitud del arreglo");
      scanf ("%d", &n);

      buffer = (int*) malloc((n)* sizeof(int));
      if (buffer==NULL) exit (1);

      p_buffer = buffer;
      for (i=0;i<n; i++){
          printf("Ingresa el valor %d \n", i);
          scanf("%d", p_buffer++);
      }

      p_buffer = buffer;
      printf("\nLos valores son\n");
      for (n=0; n<i; n++){
          printf("arreglo[%d] = %d \n", n, *p_buffer++);
      }

      free (buffer);
      system("Pause");
}
```

1000	i	3
1004	n	3
1008	*buffer	1016
1012	*p_buffer	1028
1016	buffer[0]	5
1020	buffer[1]	6
1024	buffer[2]	7
1028		
1032		
1036		
1040		
1044		
1048		
1013		
1014		
1015		
1016		

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;
}
```