Unidad III
Arreglos en C

1. Arreglos de una dimensión (vectores).
2. Cadenas de texto (strings).
3. Arreglos y funciones.
4. Arreglos de dos dimensiones (matrices).
5. Inicialización de arreglos.
6. Arreglos multidimensionales.
7. Aplicaciones de vectores y matrices.
8. Ejercicios.

UPB
UNIVERSIDAD PRIVADA BOLIVIANA
Compromiso de Excelencia

Hasta ahora, ha podido desarrollar diversas aplicaciones que utilizan solo unas cuantas variables.

Ej. Implemente una aplicación que reciba tres números enteros desde el teclado y muestre la suma de dichos números.

```
#include <stdio.h>

void main( void )
{
      int Num1, Num2, Num3, Res;

      printf("Introduzca un número: ");
      scanf("%i", &Num1 );
      printf("Introduzca otro número: ");
      scanf("%i", &Num2 );
      printf("Introduzca un último número: ");
      scanf("%i", &Num3 );

      Res = Num1 + Num2 + Num3;

      printf("%i + %i + %i = %i", Num1, Num2, Num3, Res );
}
```

Un programador novato podría haber resuelto el anterior problema con tres o cuatro variables. Pero, ¿Qué pasa cuando se desean utilizar muchas variables, almacenarlas temporalmente en la memoria para luego procesarlas y finalmente mostrar en pantalla el resultado obtenido?

Ej. Implemente una aplicación que reciba **N** números enteros desde el teclado y muestre la suma de dichos números.

```
#include <stdio.h>

void main( void )
{
      int Numero, Resultado, Cantidad, Contador;

      printf("Cuantos números desea sumar?: ");
      scanf("%i", &Cantidad );

      Contador = 0;
      Resultado = 0;

      while( Contador < Cantidad )
      {
            printf("Introduzca un número: ");
            scanf("%i", &Numero );

            Resultado = Resultado + Numero;
            Contador++;
      }

      printf("La sumatoria de los números es %i", Resultado );
}
```

Ciertamente un programador de nivel intermedio puede resolver el anterior problema con cuatro variables pero apoyándose en un bucle o ciclo (estructura de control de flujo repetitivo como for, while, do-while).

El nuevo problema que surge, radica en que el ingreso de datos y el procesamiento están obligatoriamente unidos dentro del bucle. Si bien esto funciona y es factible, rompe con el esquema de programación estructurada donde se debería primero realizar solo la lectura de todos los datos. En segunda instancia, procesar los datos y finalmente mostrar en pantalla el resultado obtenido. Es en este contexto, donde el uso de arreglos es particularmente útil.

Ej. Implemente una aplicación que reciba **N** números enteros desde el teclado y muestre la suma de dichos números.

```
#include <stdio.h>

void main( void )
{
      int Numeros[100];       // Declara 100 números enteros
      int Resultado, Cantidad, Contador;

      printf("Cuantos números desea sumar?: ");
      scanf("%i", &Cantidad );

      // Solo ingreso de datos
      for(Contador=0 ; Contador<Cantidad ; Contador++)
      {
            printf("Introduzca un número: ");
            scanf("%i", &Numeros[Contador] );
      }

      // Solo procesamiento de datos
      for(Contador=0, Resultado=0 ; Contador<Cantidad ; Contador++)
            Resultado = Resultado + Numeros[Contador];

      // Muestra los resultados obtenidos
      printf("La sumatoria de los números es %i", Resultado );
}
```

Ahora se ha utilizado un arreglo para almacenar los datos en variables y esto ha permitido separar todos los procesos (ingreso de datos, procesamiento y muestra de resultados). Cada uno de estos procesos puede ser implementado como una función ó un procedimiento (en **C** una función de tipo void). Donde, cada función se especializa en un trabajo específico.

1. Arreglos de una dimensión (vectores)

Un arreglo es un conjunto de variables del mismo tipo. Todas tienen el mismo nombre, se diferencian unas de otras por un índice y son contiguas (están una tras de la otra).

La sintaxis para declarar un arreglo de una dimensión es:

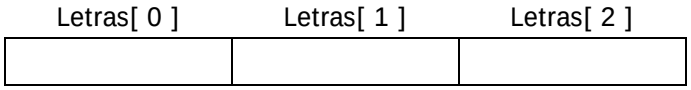
TipoDeDato NombreDelArreglo [Tamaño];

Importante:

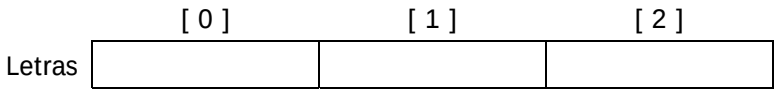
- Los **corchetes son obligatorios**.
- El Tamaño especifica la cantidad de elementos del arreglo.
- Los índices o posiciones disponibles van desde **0** hasta **Tamaño-1**.

```
Ej.    char Letras[3];   // Declara un arreglo para almacenar tres
                         // caracteres. Los índices van desde 0 a 2
```

La representación gráfica del arreglo `Letras` es:



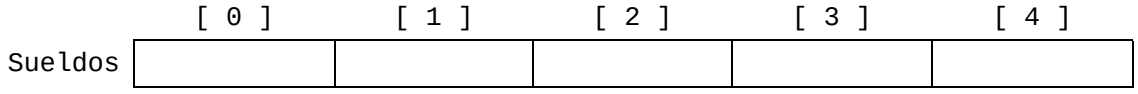
Ó también:



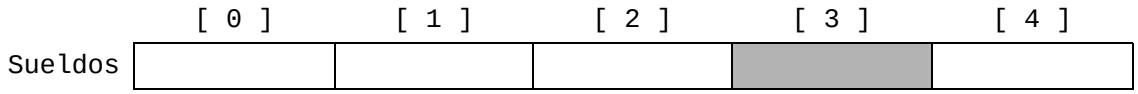
Recuerde que un arreglo es un conjunto de variables del mismo tipo. Todas tienen el mismo nombre, se diferencian unas de otras por un índice y son contiguas.

```
Ej.    float Sueldos[5]; // Declara un arreglo para almacenar cinco
                         // flotantes. Los índices van desde 0 a 4
```

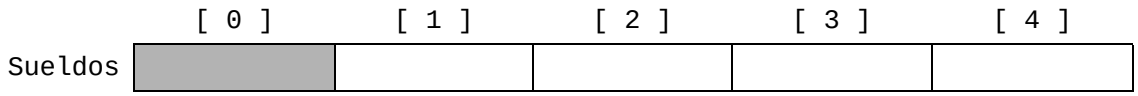
La representación gráfica del arreglo `Sueldos` es:



Recuerde que si desea mostrar uno de los elementos del arreglo habrá que escribir su nombre y el índice. Por ejemplo la instrucción `printf("%f", Sueldos[3]);` mostrará el contenido de la variable flotante que está en gris:



La instrucción `scanf("%f", &Sueldos[0]);` permitirá que se ingrese desde teclado un número flotante dentro de la variable que está en gris:



Ej. Escriba un programa en **C** para introducir 10 enteros, desplegarlos por pantalla y hallar el menor de todos ellos.

```
#include <stdio.h>

void main( void )
{
      int Numeros[10];  // Reserva 10 enteros (posiciones desde el
                        // 0 hasta 9)
      int Pos, Menor;

      // Lectura de los datos
      for( Pos = 0 ; Pos < 10 ; Pos++ )
      {
            printf("Introduzca el número %i: ", Pos + 1);
            scanf("%i", &Numeros[Pos] );
      }

      // Muestra los datos
      printf("\nLos números son:\n");
      for( Pos = 0 ; Pos < 10 ; Pos++ )
            printf("%i ", Numeros[Pos] );

      // Encuentra el menor de los datos introducidos
      Menor = Numeros[0];
      for( Pos = 1 ; Pos < 10 ; Pos++ )
            if ( Numeros[Pos] < Menor )
                  Menor = Numeros[Pos];

      printf("\nEl menor de los números es %i", Menor );
}
```

2. Cadenas de texto (strings)

El tipo de dato cadena de texto ó string NO existe en el lenguaje **C**, sin embargo puede ser emulado por medio de un arreglo de caracteres de una dimensión. El final de la cadena de texto está delimitado por el carácter nulo **'\0'**.

Ej. La representación gráfica de una cadena de texto con el carácter nulo es:

	[0]	[1]	[2]	[3]	[4]
Cadena	'h'	'o'	'l'	'a'	'\0'

Sin embargo la siguiente NO es una cadena de texto:

	[0]	[1]	[2]	[3]	[4]
Texto	'a'	'd'	'i'	'o'	's'

Para utilizar las funciones de manejo de cadenas debe incluirse el archivo de cabecera **string.h**.

Función	Descripción	Ejemplo
printf("%s", cad)	Muestra en la pantalla la cadena **cad**. (%s es por string)	printf("%s", Cadena); printf("%s", "que facil");

scanf("%s", &cad)	Lee desde el teclado la cadena **cad**. (<enter>, <space bar> y <tab> finalizan la introducción de la cadena que no puede tener espacios en blanco)	scanf("%s", Cadena);
		
		// Ojo Cadena va a cambiar pero no hay que enviarla por referencia

strlen(cad)	**str**ing **len**gth Retorna la longitud de la cadena **cad**.	Tamanio = strlen("hola"); Tam = strlen(MiCadena);
strcpy(cadDes, cadOr)	**str**ing **c**o**py** Copia la cadena **cadOr** (constante o variable) en **cadDes** (variable).	strcpy(Destino, "Prueba"); strcpy(Final, Destino);
strcat(cadDes, cadOr)	**str**ing con**cat**enate Concatena **cadOr** al final de **cadDes**. El resultado se guarda en **cadDes**.	strcpy(Destino, "Prueba"); strcat(Destino, " de texto");
strcmp(cadA, cadB)	**str**ing **c**o**mp**are Compara las cadenas **cadA** y **cadB**. Retorna un número: **–** si cadA < cadB **+** si cadA > cadB **0** si cadA == cadB	Res = strcmp("Hole", "Hola"); // Esta función es muy útil para ordenar alfabéticamente las cadenas de texto que se comparan.
puts(cad)	**put s**tring Muestra en la pantalla la cadena **cad**.	puts(cadena);

gets(cad)	**get s**tring
	Lee desde el teclado la cadena **cad**. (<enter> finaliza la introducción de la cadena que puede tener espacios en blanco)
		gets(cadena);
		
		// Ojo Cadena va a cambiar pero no hay que enviarla por referencia

Ej. Introduzca dos cadenas de texto, concaténelas y despliegue el resultado en pantalla.

```
#include <stdio.h>
#include <string.h>

void main( void )
{
      char Cadena1[30], Cadena2[30];

      printf("Introduzca una cadena de texto: ");
      scanf("%s", Cadena1 );        // Ojo: scanf() sin &
      printf("Introduzca otra cadena de texto: ");
      scanf("%s", Cadena2 );

      strcat(Cadena1, " ");
      strcat(Cadena1, Cadena2);

      printf("Uniendo las cadenas: %s", Cadena1 );
}
```

Ej. Introduzca dos cadenas de texto y despliéguelas en orden alfabético en la pantalla.

```
#include <stdio.h>
#include <string.h>

void main( void )
{
      char CadA[30], CadB[30];

      printf("Introduzca una cadena de texto: ");
      gets( CadA );
      printf("Introduzca otra cadena de texto: ");
      gets( CadB );

      if ( strcmp(CadA, CadB) < 0 )
            printf("Ordenadas: %s y %s", CadA, CadB );
      else
            printf("Ordenadas: %s y %s", CadB, CadB );
}
```

Importante: Recuerde que una cadena de texto es un arreglo de caracteres delimitado por un carácter nulo, entonces es posible recorrer cada una de las posiciones del arreglo e ir mostrando el carácter almacenado ahí hasta que se llegue al carácter nulo.

Ej.

```
#include <stdio.h>
#include <string.h>

void main( void )
{
      char Cadena[100];
      int Pos;
```

```
        printf("Introduzca un texto con espacios en blanco:\n");
        gets( Cadena );

        printf("\nLa cadena que introdujo es:\n");
        for( Pos = 0 ; Cadena[Pos] != '\0' ; Pos++ )
              printf("%c", Cadena[Pos] );
}
```

3. Arreglos y funciones

En el anterior capítulo se expuso la forma en la que el lenguaje **C** utiliza el paso de parámetros por valor y el paso de parámetros por referencia como medio para enviar y recibir datos entre funciones.

Estos mecanismos también se utilizan con los arreglos, pero tienen algunas diferencias:

- Para enviar el arreglo completo como argumento en la llamada a la función, no se utiliza **&** ni tampoco se especifica el tamaño del arreglo. Basta con especificar el nombre del arreglo que se va a enviar (notación similar al paso de parámetros por valor).
- Para recibir el arreglo como parámetro en la implementación de la función, se debe especificar el tipo de dato y el tamaño del arreglo.
- El envío de un arreglo entre dos funciones es un caso especial y en realidad es similar al paso de parámetros por referencia donde se envían los datos originales. Sin embargo, **C** tiene una manera especial de envío, pues solo manda a la función la dirección de inicio del arreglo. Todo este trabajo se realiza por medio de punteros.

Ej. Escriba un programa en **C** para introducir 10 enteros, desplegarlos por pantalla y hallar el menor de todos ellos. Utilice la programación estructurada con prototipos de funciones para resolver este ejercicio.

```
#include <stdio.h>

// Prototipos de funciones
void Lee( int Numeros[10] );
void Muestra( int Numeros[10] );
int Encuentra_Menor( int Numeros[10] );

// Función principal
void main( void )
{
      int Numeros[10];  // Variables locales que reservan 10 enteros
                        // (posiciones desde el 0 hasta 9)
      int Menor;

      Lee( Numeros );

      Muestra( Numeros );

      Menor = Encuentra_Menor( Numeros );
      printf("\nEl menor de los números es %i", Menor );
}
```

```
// Implementación de las funciones
void Lee( int Numeros[10] )
{
      int Pos;

      // Lectura de los datos
      for( Pos = 0 ; Pos < 10 ; Pos++ )
      {
            printf("Introduzca el número %i: ", Pos + 1);
            scanf("%i", &Numeros[Pos] );
      }
}

void Muestra( int Numeros[10] )
{
      int Pos;

      // Muestra los datos
      printf("\nLos números son:\n");
      for( Pos = 0 ; Pos < 10 ; Pos++ )
            printf("%i ", Numeros[Pos] );
}

int Encuentra_Menor( int Numeros[10] )
{
      int Pos, Menor;

      // Encuentra el menor de los datos introducidos
      Menor = Numeros[0];
      for( Pos = 1 ; Pos < 10 ; Pos++ )
            if ( Numeros[Pos] < Menor )
                  Menor = Numeros[Pos];

      return Menor;
}
```

Importante: Recuerde que cuando se llama a una función y se desea enviarle un arreglo como argumento, basta con poner el nombre del arreglo. Es importante tener en cuenta que el mecanismo de envío del arreglo funciona de manera similar al paso de parámetros por referencia (es como si se enviaran los datos originales). Cualquier modificación a los datos del arreglo provocará que se retornen los datos modificados a la función donde se hizo la llamada.

Importante: Si bien la codificación se ha mejorado mucho por medio de buenas prácticas de programación, aún se puede mejorar el trabajo con arreglos por medio de constantes para los tamaños. Imagine que le piden modificar el código para que funcione ahora con 50 números. Entonces, tendrá que sustituir la ocurrencia de la constante numérica 10 por 50. Si tiene muchas funciones donde aparece el 10, deberá realizar el cambio en cada una de ellas. Esto es moroso y pueden surgir errores de programación. Para optimizar la codificación se puede utilizar:

```
#define TAM 10
```

Y especificar todos los tamaños del arreglo por medio de esta constante. Si en algún momento se desea cambiar el tamaño del arreglo, basta con modificar esta línea.

Ej. Introducción de 10 números y muestra del menor de ellos.

```
#include <stdio.h>

#define TAM 10

// Prototipos de funciones
void Lee( int Numeros[TAM] );
void Muestra( int Numeros[TAM] );
int Encuentra_Menor( int Numeros[TAM] );

// Función principal
void main( void )
{
      int Numeros[TAM], Menor;

      Lee( Numeros );
      Muestra( Numeros );
      Menor = Encuentra_Menor( Numeros );
      printf("\nEl menor de los números es %i", Menor );
}

// Implementación de las funciones
void Lee( int Numeros[TAM] )
{
      int Pos;

      for( Pos = 0 ; Pos < TAM ; Pos++ )  // Lectura de datos

      {
            printf("Introduzca el número %i: ", Pos + 1);
            scanf("%i", &Numeros[Pos] );
      }
}

void Muestra( int Numeros[TAM] )
{
      int Pos;

      printf("\nLos números son:\n");
      for( Pos = 0 ; Pos < TAM ; Pos++ )  // Muestra los datos
            printf("%i ", Numeros[Pos] );
}

int Encuentra_Menor( int Numeros[TAM] )
{
      int Pos, Menor;

      Menor = Numeros[0];
      for( Pos = 1 ; Pos < TAM ; Pos++ )  // Encuentra el menor
            if ( Numeros[Pos] < Menor )
                  Menor = Numeros[Pos];

      return Menor;
}
```

El envío y recepción de cadenas de texto se somete a las mismas reglas de los arreglos.

Ej. Introduzca dos cadenas de texto, concaténelas y despliegue el resultado en pantalla.

```
#include <stdio.h>
#include <string.h>

#define TAM 10

void Introduce( char Cadena[TAM] )
{
      puts("Introduzca una cadena de texto: ");
      gets( Cadena );
}

void Concatena( char CadA[TAM], char CadB[TAM], char CadC[TAM] )
{
      strcpy(CadC, "");             // Limpia la cadena destino
      strcpy(CadC, CadA);           // Copia la primera cadena
      strcat(CadC, " ");            // Concatena un espacio blanco
      strcat(CadC, CadB);           // Une la segunda cadena
}

void main( void )
{
      char Cad1[TAM], Cad2[TAM], Cad3[TAM];

      Introduce( Cad1 );
      Introduce( Cad2 );

      Concatena( Cad1, Cad2, Cad3 );

      printf("Uniendo las cadenas: %s", Cad3 );
}
```

Importante: Observe que la función `Introduce( )` se ha implementado una sola vez, pero puede ser utilizada muchas veces y con distintas cadenas de texto. Esta es una ventaja de la programación estructurada.

4. Arreglos de dos dimensiones (matrices)

Un arreglo de dos dimensiones es un conjunto de variables del mismo tipo. Todas tienen el mismo nombre, se diferencian unas de otras por dos índices y son contiguas.

La sintaxis para declarar un arreglo de dos dimensiones es:

TipoDeDato NombreDelArreglo [Tam1][Tam2];

Importante:

- Los **corchetes son obligatorios**.
- Tam1 especifica la cantidad de filas del arreglo.
- Los índices o posiciones disponibles para las filas van desde **0** hasta **Tam1-1**.
- Tam2 especifica la cantidad de columnas del arreglo.
- Los índices o posiciones disponibles para las columnas van desde **0** hasta **Tam2-1**.
- La cantidad de elementos del arreglo bidimensional es **Tam1 * Tam2**.

Ej.
```
char Datos[5][3]; // Declara un arreglo de dos dimensiones
                  // que almacena 15 elementos.
                  // Los índices de filas van desde 0 a 4.
                  // Los índices de columnas van desde 0 a 2.
```

La representación gráfica del arreglo `Datos` es:

		Columnas		
	Datos	[0]	[1]	[2]
	[0]			
	[1]			
Filas	[2]			
	[3]			
	[4]			

Para acceder al elemento que está en gris habrá que escribir su nombre y especificar la fila y columna en la que se encuentra:

- Para mostrar su contenido se usa: `printf("%c", Datos[1][2]);`.
- Para introducir un dato se utiliza: `scanf("%c", &Datos[1][2]);`.

Ej. Escriba un programa para introducir 9 caracteres en una matriz y despliéguelos en pantalla.

```
#include <stdio.h>

#define FILAS 3
#define COLUMNAS 3

void main( void )
{
      char Datos[FILAS][COLUMNAS];
      int fil, col;

      // Introduce los datos desde teclado
      for( fil = 0 ; fil < FILAS ; fil++ )
            for( col = 0 ; col < COLUMNAS ; col++ )
            {
                  printf("Introduzca un carácter: ");
                  scanf("%c", &Datos[fil][col]);
                  fflush( stdin );        // Limpia el buffer,
                                          // evita el <ENTER>
            }
```

```
        // Muestra los datos en pantalla
        printf("\nLos caracteres son:\n");

        for( fil = 0 ; fil < FILAS ; fil++ )
              for( col = 0 ; col < COLUMNAS ; col++ )
                    printf("%c  ", Datos[fil][col]);
}
```

Después de haber copiado y ejecutado el anterior código, es probable que se haya llevado una decepción, pues los datos no se mostraron de forma matricial. Para solucionar esto, solo se debe modificar la manera como los datos de despliegan en la pantalla. Recuerde que los datos en la memoria se almacenan de forma contigua pero pueden ser presentados en pantalla de diversas maneras.

Ej. Escriba un programa para introducir 9 caracteres en una matriz y despliéguelos de forma matricial en la pantalla.

```
#include <stdio.h>

#define FILAS 3
#define COLUMNAS 3

void main( void )
{
      char Datos[FILAS][COLUMNAS];
      int fil, col;

      // Introduce los datos desde teclado
      for( fil = 0 ; fil < FILAS ; fil++ )
            for( col = 0 ; col < COLUMNAS ; col++ )
            {
                  printf("Introduzca un carácter: ");
                  scanf("%c", &Datos[fil][col]);
                  fflush( stdin );        // Limpia el buffer,
                                          // evita el <ENTER>
            }


      // Muestra los datos en pantalla de forma matricial
      printf("\nLos caracteres son:\n");

      for( fil = 0 ; fil < FILAS ; fil++ )
      {
            for( col = 0 ; col < COLUMNAS ; col++ )
                  printf("%c  ", Datos[fil][col]);

            printf("\n");     // Fuera del for de columnas
      }
}
```

Ej. Utilizando programación estructurada, implemente un programa para introducir 10 enteros en una matriz, despliéguelos de forma matricial en la pantalla y obtenga el promedio de los números.

```
#include <stdio.h>

#define FILAS 5
#define COLUMNAS 2

void Introduce(int Enteros[FILAS][COLUMNAS])
{
      int fil, col;

      for( fil = 0 ; fil < FILAS ; fil++ )
            for( col = 0 ; col < COLUMNAS ; col++ )
            {
                  printf("Introduzca un entero: ");
                  scanf("%i", &Enteros[fil][col]);
            }
}

void Muestra(int Enteros[FILAS][COLUMNAS])
{
      int fil, col;

      printf("\nLos numeros son:\n");

      for( fil = 0 ; fil < FILAS ; fil++ )
      {
            for( col = 0 ; col < COLUMNAS ; col++ )
                  printf("%3i", Enteros[fil][col]);

            printf("\n");     // Fuera del for de columnas
      }
}

float Promedio(int Enteros[FILAS][COLUMNAS])
{
      int fil, col, Sumatoria = 0;

      for( fil = 0 ; fil < FILAS ; fil++ )
            for( col = 0 ; col < COLUMNAS ; col++ )
                  Sumatoria += Enteros[fil][col];

      return (Sumatoria / (FILAS * COLUMNAS));
}

void main( void )
{
      int Datos[FILAS][COLUMNAS];

      Introduce( Datos );
      Muestra( Datos );
      printf("\nEl promedio de los números es: %5.2f",
                                                Promedio(Datos) );
}
```

5. Inicialización de arreglos

Los arreglos pueden ser inicializados al momento de ser declarados:

Ej. `char Letras[3] = {'a', 'b', 'c'};`

La representación gráfica del arreglo `Letras` es:

	[0]	[1]	[2]
Letras	'a'	'b'	'c'

Ej. `float Sueldos[4] = {7700.2, 5230.0, 4890.88, 4901.33};`

La representación gráfica del arreglo `Sueldos` es:

	[0]	[1]	[2]	[2]
Letras	7700.2	5230.0	4890.88	4901.33

Ej. `int Datos[3][2] = { {2, 4}, {6, 8}, {0, 1} };`

La representación gráfica del arreglo `Datos` es:

		Columnas	
Datos		[0]	[1]
Filas	[0]	2	4
	[1]	6	8
	[2]	0	1

Una manera equivalente de inicializar un arreglo de dos dimensiones es:

```
int Datos[3][2] = { 2, 4, 6, 8, 0, 1 };
```

Ej.

```
#include <stdio.h>

void main( void )
{
      int Datos[3][2] = { 2, 4, 6, 8, 0, 1 }, fil, col;

      for( fil = 0 ; fil < 3 ; fil++ )
      {
            for( col = 0 ; col < 2 ; col++ )
                  printf("%3i", Datos[fil][col]);
            printf("\n");
      }
}
```

En **C** se puede inicializar un arreglo entero con una sola constante.

Ej. Declare un arreglo de dos dimensiones e inicialícelo por medio de la constante numérica cero.

```
#include <stdio.h>

#define FILAS 3
#define COLUMNAS 2

void main( void )
{
      int Datos[FILAS][COLUMNAS] = { 0 };
      int fil, col;

      for( fil = 0 ; fil < FILAS ; fil++ )
      {
            for( col = 0 ; col < COLUMNAS ; col++ )
                  printf("%3i", Datos[fil][col]);

            printf("\n");
      }
}
```

Importante: Algunas excepciones en la inicialización de arreglos son las cadenas de texto. Donde se puede utilizar la manera tradicional de inicialización (sin olvidar el carácter nulo), ó también se dispone de una manera alterna de inicialización:

```
char Texto[4] = { 'Q', 'u', 'e', '\0'};   // Forma tradicional

char Cadena[20] = "fácil es C";           // Forma alterna
```

Ej.

```
#include <stdio.h>

void main( void )
{
      char Texto[4] = { 'Q', 'u', 'e', '\0'};
      char Cadena[20] = "fácil es C";

      printf("%s %s", Texto, Cadena);
}
```

Importante: Cuando se declara un arreglo sin tamaño es necesario inicializarlo. De esta manera, el compilador se encarga de contar la cantidad de elementos en la inicialización y reserva dicho espacio en la memoria para el arreglo.

Ej.

```
int Numeros[ ] = { 22, 33, 44, 55 };      // Arreglo de 4 elementos

char Cadena[ ] = "Computación";           // Arreglo de 12 elementos
                                          // se incluye el '\0'.
```

6. Arreglos multidimensionales

Un arreglo de multidimensional no es más que un arreglo de múltiples dimensiones. La sintaxis para declarar un arreglo multidimensional es:

TipoDeDato NombreDelArreglo [Tam1][Tam2]...[TamN];

Importante:

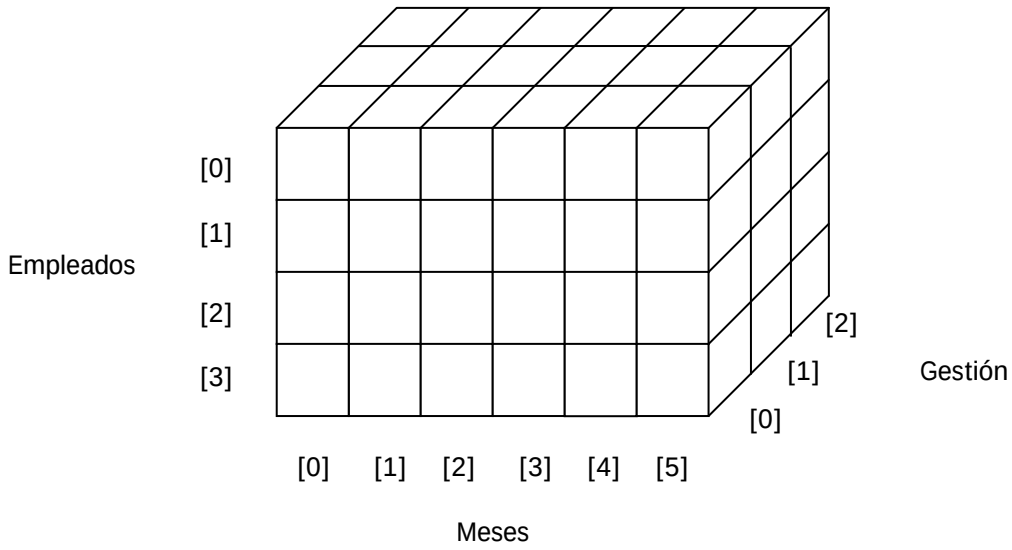
- Los **corchetes son obligatorios**.
- Tam1 especifica la cantidad de filas del arreglo.
- Los índices o posiciones disponibles para las filas van desde **0** hasta **Tam1-1**.
- Tam2 especifica la cantidad de columnas del arreglo.
- Los índices o posiciones disponibles para las columnas van desde **0** hasta **Tam2-1**.
- Los puntos suspensivos representan "y así consecutivamente".
- La cantidad de elementos del arreglo multidimensional es la productoria de tamaños de cada una de las dimensiones: **Tam1 * Tam2 * ... * TamN**.

Ej.
```
char Letras[10][5][20][3][5];       // Arreglo de 5 dimensiones que
                                    // contiene 15000 elementos.
```

No es frecuente utilizar arreglos multidimensionales porque requieren una gran cantidad de espacio en memoria y la memoria es uno de los recursos más preciados en una computadora. Lo más común con los arreglos multidimensionales es utilizar arreglos de hasta tres dimensiones para almacenar y analizar datos (cubos de datos).

Un arreglo tridimensional podría utilizarse para almacenar los pagos mensuales a empleados en diferentes gestiones:

Ej.
```
float Sueldos[4][6][3];
```



7. Aplicaciones de vectores y matrices

Los vectores y las matrices son de gran utilidad en muchas áreas de la ciencia. Se los puede utilizar para resolver los siguientes problemas:

Rama	Problema
Física	Trabajo con vectores. Estática. Dinámica.
Algebra	Trabajo con vectores y matrices.
Métodos numéricos	Encontrar raíces, trabajo con polinomios, resolución de sistemas de ecuaciones lineales.
Investigación operativa	Resolución de problemas de maximización y minimización (métodos de las M y de las dos fases), problema del comerciante viajero, trabajo con grafos.
Informática	Conjuntos de registros que son el resultado de consultas a Bases de Datos. Problemas de ordenamiento.
Administración	Planillas de sueldos. Manejo de inventarios. Cálculos de depreciaciones.
Finanzas	Cálculo de VAN, TIR, VA, VF.
Estadística	Encuestas. Proyecciones. Interpolaciones. Extrapolaciones. Ajuste de curvas.

La lista de aplicaciones donde se pueden utilizar los vectores y las matrices es inmensa. Especialmente en todas las ingenierías y en las ciencias administrativas y financieras.

8. Ejercicios

Resuelva los siguientes problemas:

1. Llene un arreglo de una dimensión con 5 números leídos desde el teclado (evite los negativos y ceros). Obtenga el promedio de los números que componen el arreglo.

   Ej. con los números

0	1	2	3	4
5	7	2	1	5

   el promedio es 4

2. Llene un arreglo de una dimensión con 5 números flotantes que corresponden a los salarios de los empleados y han sido introducidos desde el teclado (evite los negativos y ceros). Muestre por pantalla cuál es el mayor y el cuál es el menor de todos los salarios almacenados.

3. Llene un arreglo de una dimensión con caracteres generados al azar. Cuente la cantidad de vocales, consonantes, la cantidad de dígitos y otros signos ó símbolos de puntuación que se puedan haber generado (desde el carácter con valor ASCII 40 hasta el 140). Muestre en la pantalla los resultados obtenidos.

4. Declare una matriz sin datos. Luego llene la matriz por medio de un bucle y despliéguela en la pantalla de manera matricial:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

5. Declare matrices sin datos. Luego genere los números para llenar las matrices y finalmente despliéguelas:

A

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

B

1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1

C

1	0	1	0
0	1	0	1
1	0	1	0
0	1	0	1

6. Declare una matriz sin datos y que luego se llene por medio de bucles de la siguiente forma:

16	12	8	4
15	11	7	3
14	10	6	2
13	9	5	1

7. Lea un carácter desde el teclado (el rango válido es desde la ‘a’ hasta la ‘q’) y luego llene un arreglo bidimensional con los siguientes caracteres al carácter introducido. Ej. Si ‘d’ es el carácter introducido desde teclado entonces los caracteres siguientes son ‘e’, ‘f’, ‘g’, ‘h’, ‘i’, ‘j’, ‘k’, ‘l’, ‘m’.

‘ g ’	‘ f ’	‘ e ’
‘ j ’	‘ i ’	‘ h ’
‘ m ’	‘ l ’	‘ k ’

8. Cargue un arreglo de tres filas un carácter a la vez (utilice getch( ) y considere que cada fila puede almacenar hasta 50 caracteres). Se utilizará el retorno para saltar a la siguiente fila. Finalmente, muestre completamente la matriz de caracteres.

0	1	2	3		48	49
‘ Q ’	‘ u ’	‘ e ’	‘ \r ’	...		
‘ f ’	‘ a ’	‘ c ’	‘ i ’	...		
‘ e ’	‘ s ’	‘ \r ’		...		

9. Cargue una matriz **A** y una matriz **B** con datos generados al azar. Realice las siguientes operaciones con matrices:

a. C = A + B.
b. C = A – B.
c. C = A x B.

10. Cargue una matriz de N x N con números correlativos y despliéguela en forma de caracol:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

El listado será: 1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10