Tema 4. VECTORES

TEMA 4.
VECTORES (ARRAYS)

1. INTRODUCCIÓN.
2. ARRAYS UNIDIMENSIONALES.
3. ARRAYS MULTIDIMENSIONALES. MATRICES.
4. INICIALIZACIÓN DE ARRAYS.
5. OPERACIONES SOBRE ARRAYS.
6. PASO DE ARRAYS A FUNCIONES.
7. CADENAS DE CARACTERES (STRINGS).

1. Introducción

- ✓ **Los tipos de datos vistos hasta ahora se denominan *escalares* o *elementales***
  - × Las variables de estos tipos pueden tomar valores simples (un número entero, un número real, un carácter)
- ✓ **Un vector es un tipo de datos constituido por un número fijo de componentes del mismo tipo, llamado *tipo base***
  - × Ejemplo: las notas de un alumno durante un curso se pueden guardar en un vector de números reales
  - × Es una estructura de *acceso aleatorio*: cada componente puede ser accedida arbitrariamente
- ✓ **Los vectores pueden ser unidimensionales o multidimensionales**
- ✓ **En los lenguajes de programación los vectores se suelen llamar *arrays*.**

2. *Arrays* unidimensionales

- ✓ **Sintaxis de la declaración de un *array* unidimensional en C:**

```
tipoBase   nombreArray[expresionConstante];
```

- ✓ **Ejemplo: Array unidimensional de 5 enteros, llamado vector:**

```
int vector[5];
```

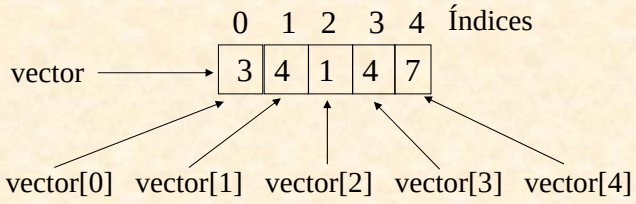
  - × Con la declaración del vector anterior, se dispone de espacio en memoria central para almacenar 5 valores enteros, referenciados con el nombre vector.

2. *Arrays* unidimensionales

- ✓ **Para acceder a cada componente del *array* se escribe el nombre seguido de la posición o *índice*, entre corchetes, de esa componente dentro del vector**
  - × Se debe tener en cuenta que los índices comienzan en 0
  - × Ejemplo: dada la declaración

```
int vector[5];
```

los índices del vector van de 0 a 4:



2. *Arrays* unidimensionales

EJEMPLO 1: Lectura y escritura de un array de enteros:

```
#include <stdio.h>

void main(void)
{
   int v[10]; // array de 10 enteros
   int i;

   for (i=0; i<10; i++)
   {
      printf("Valor %d: ", i);
      scanf("%d", &v[i]);
   }

   printf("Valores leidos:\n");

   for (i=0; i<10; i++)
      printf("Componente %d: %d\n", i, v[i]);
}
```


3. *Arrays* multidimensionales

- ✓ **Un *array* multidimensional se define de forma análoga a los unidimensionales, separando cada uno de los índices por un par de corchetes**
- ✓ **Sintaxis de la declaración:**

```
tipoBase nombreArray[constante1][constante2] ... [constanteN];
```

- ✓ **Ejemplo: declaración de un *array* bidimensional de 3x3 elementos (matriz) de números reales:**

```
float matriz[3][3];
```

  - El primer índice corresponde a las filas de la matriz y el segundo a las columnas
  - Para acceder a la componente (i,j):

```
matriz[i][j];
```

3. *Arrays* multidimensionales

EJEMPLO 2: Lectura y escritura de una matriz 3x3:

```
#include <stdio.h>

#define NUMFILAS 3
#define NUMCOLUMNAS 4

void main(void)
{
   float m[NUMFILAS][NUMCOLUMNAS];
   int i,j;

   // leemos la matriz por filas
   for (i=0; i<NUMFILAS; i++)
   {
      // leemos la fila i
      for (j=0; j<NUMCOLUMNAS; j++)
      {
          printf("Elemento (%d,%d): ",i,j);
          scanf("%f", &m[i][j]);
      }
   }

   // escribimos la matriz por filas

   for (i=0; i<NUMFILAS; i++)
   {
      // escribimos la fila i
      for (j=0; j<NUMCOLUMNAS; j++)
      {
        printf("%.3f", m[i][j]);
      }
      printf("\n");
   }
}
```


4. Inicialización de *arrays*

- ✓ **Se puede asignar valores iniciales a las componentes de un *array* en la declaración**
  - × En la inicialización se puede omitir el tamaño del array
- ✓ **Ejemplo: inicialización de un *array* unidimensional**

```
int v[5] = {7, -5, 0, 4, 1 };
int v[] = {7, -5, 0, 4, 1 };        // Equivalente a la anterior
```

- ✓ **Ejemplo: inicialización de un *array* bidimensional**

```
int m[2][3] = {9, 0, -1, 3, 4, 7 };
int m[][3] = {9, 0, -1, 3, 4, 7 };
```

$$m = \begin{pmatrix} 9 & 0 & -1 \\ 3 & 4 & 7 \end{pmatrix}$$

5. Operaciones sobre *arrays*

- ✓ **En C no existen funciones estándar para el manejo de *arrays* generales: por ejemplo no existen funciones para leer, sumar, multiplicar o escribir dos vectores**
- ✓ **Las distintas operaciones sobre *arrays* se han de efectuar elemento a elemento usando sentencias de iteración**
- ✓ **La programación de estas operaciones se suele hacer con funciones**
- ✓ **Las operaciones de manipulación de *arrays* más frecuentes son las siguientes:**
  - × Lectura, escritura, asignación
  - × Comparación
  - × Búsqueda de un elemento dentro de un vector
  - × Ordenación

6. Paso de *arrays* a funciones

- ✓ **Para declarar que una función recibe un *array* como parámetro, se debe declarar el parámetro como un *array* del tipo base adecuado**
  - × Si el *array* es unidimensional, no es necesario especificar el tamaño del mismo (corchetes vacíos).
  - × Si el *array* es multidimensional, se debe especificar el tamaño de todas las dimensiones, excepto el de la primera, que se puede dejar vacío.
- ✓ **En la llamada a la función que espera un *array* como parámetro, el nombre del mismo debe aparecer solo, sin corchetes ni índices**
  - × De esa forma, se permite que el *array* completo sea pasado como parámetro a la función.
- ✓ **Los *arrays* se pasan por *dirección*: al pasar un *array* como parámetro no se hace una copia para la función de los valores de los elementos del *array***
  - × El nombre de un *array* representa la dirección de memoria donde se encuentra la primera componente del mismo, y a partir de esta dirección se accede a todas las componentes .

6. Paso de *arrays* a funciones

EJEMPLO 3: Uso de funciones con *arrays* unidimensionales:

```
#include <stdio.h>

#define TAM 5

void LeerVector(int v[])
{
   int i;

   for (i=0; i<TAM; i++)
     scanf("%d", &v[i]);
}

void EscribirVector(int v[])
{
   int i;

   for (i=0; i<TAM; i++)
     printf("%d ", v[i]);
}
```

```
void AsignarVector(int destino[], int origen[])
{
   int i;

   for (i=0; i<TAM; i++)
       destino[i] = origen[i];
}

void main(void)
{
   int v1[TAM], v2[TAM];

   printf("Introducir el vector 1: ");
   LeerVector(v1);
   AsignarVector(v2,v1);
   printf("Vector 2: ");
   EscribirVector(v2);
}
```


6. Paso de *arrays* a funciones

EJEMPLO 4: Uso de funciones con *arrays* bidimensionales:

```
#include <stdio.h>

#define NUMFILAS 3
#define NUMCOLUMNAS 4


void LeerMatriz(float m[][NUMCOLUMNAS])
{
   int i,j;

   for (i=0; i<NUMFILAS; i++)
   {
      // leemos la fila i
      for (j=0; j<NUMCOLUMNAS; j++)
      {
          printf("Elemento (%d,%d): ",i,j);
          scanf("%f", &m[i][j]);
      }
   }
}
```

```
void EscribirMatriz(float m[][NUMCOLUMNAS])
{

   for (i=0; i<NUMFILAS; i++)
   {
      // escribimos la fila i
      for (j=0; j<NUMCOLUMNAS; j++)
      {
         printf("%.3f", m[i][j]);
      }
      printf("\n");
   }
}

void main(void)
{
   float matriz[NUMFILAS][NUMCOLUMNAS];

   LeerMatriz(matriz);
   EscribirMatriz(matriz);
}
```

7. Cadenas de caracteres

- ✓ **Una cadena de caracteres es un *array* unidimensional de caracteres terminado con el carácter nulo ('\0').**
  - × Este carácter nulo se utiliza para indicar el fin de la cadena y no aparece cuando se muestra en pantalla.
- ✓ **Una constante de este tipo se expresa escribiendo la cadena de caracteres entre comillas dobles (" ").**
  - × Ejemplo: "hola"
  - × Las cadenas de caracteres se pueden inicializar igual que cualquier otro *array*.

```
char cad[5] = "hola";
char cad[5] = { 'h', 'o' , 'l', 'a', '\0' };
```

} DECLARACIONES EQUIVALENTES

  - × Nótese la diferencia entre las constantes de tipo carácter ('a') y las constantes de tipo cadena ("a"). La cadena "a" está formada por dos caracteres: 'a' y '\0'.

7. Cadenas de caracteres

- ✓ **Una cadena de *n* caracteres necesita un *array* de *n+1* elementos, puesto que hay que añadir el carácter '\0' que indica el fin de la cadena.**

"hola" → | h | o | l | a | \0 |

- ✓ **En la inicialización se puede omitir el tamaño del *array*:**

```
char cad[] = "hola";
```

- ✓ **Existen numerosas funciones de librería en C para el tratamiento de cadenas, cuyos prototipos se encuentran en <string.h>.**
- ✓ **Para la lectura y escritura de cadenas con printf y scanf, se utiliza el formato *%s*.**
- ✓ **Para leer y escribir cadenas, se pueden utilizar las funciones *gets* y *puts*, pasándoles como parámetro el nombre de la cadena.**

7. Cadenas de caracteres

EJEMPLO 5: Cálculo de la longitud de una cadena de caracteres:

```
#include <stdio.h>

int LongitudCad(char cad[]);

void main(void)
{
   char cad[100];
   int longitud;

   printf("Introduzca cadena: ");
   gets(cad);

   longitud = LongitudCad(cad);

   printf("La longitud es: %d\n", longitud);
   system("PAUSE");
}
```

```
int LongitudCad(char cad[])
{
   int i;

   i=0;
   while (cad[i] != '\0')
      i++;

   return(i);
}
```


7. Cadenas de caracteres

EJEMPLO 6: Copia de una cadena de caracteres en otra:

```
#include <stdio.h>

void CopiaCad(char a[], char b[]);

void main(void)
{
   char cad1[100], cad2[100];

   printf("Introduzca cadena: ");
   gets(cad1);

   CopiaCad(cad1, cad2);

   printf("La cadena copiada es:\n");
   puts(cad2);

   system("PAUSE");
}
```

```
void CopiaCad(char a[], char b[])
{
   int i, longa;

   for(i=0; a[i] != '\0'; i++)
      b[i] = a[i];
   b[i] = '\0';
}
```

7. Cadenas de caracteres

EJEMPLO 7: Funciona que comprueba si dos *strings* son iguales:

```
short CadenasIguales(char a[], char b[])
{
  short iguales;
  int longitud, i;

  iguales = 1;    // Verdadero
  longitud = LongitudCad(a);
  if (longitud != LongitudCad(b))
     iguales = 0;  // falso
  else
  {
     for (i=0; i<longitud; i++)
        if (a[i] != b[i])
        {
           iguales = 0;  // falso
           break;
        }
  }
  return(iguales);
}
```