Arrays multidimensionales (Matrices)

Un array multidimensional , como su nombre indica , es un array de dos o más dimensiones. La declaración de un array de varias dimensiones se realiza de la siguiente forma:

```
tipo nombre [dim1][dim2]……..;
tipo nombre [ ] [dim]……;
```

El número de elemento de un array multidimensional es el producto de las dimensiones indicadas por dim1 , dim2, …….

Ejemplo:

```
int v[2][3][4][5][3];
```

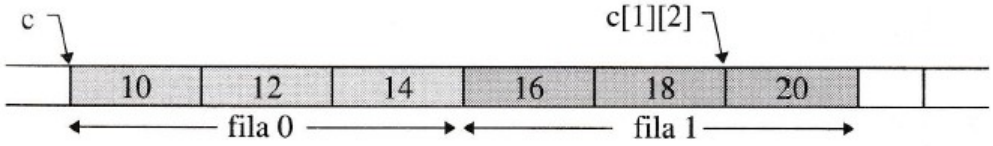
Este ejemplo define un array de 5 dimensiones con 2x3x4x5x3=360 elementos.

La primera dimensión se puede omitir cuando se inicializa el array, cuando se declara como un parámetro formal en una función o cuando se hace referencia a un array declarado en otra parte del programa.

Ejemplo:

```
int c [ ] [3] ={ 10,12,14,16,18,20}
```

Los valores en un anay se almacenan por filas. Según esto, el ejemplo anterior define un array "c" de 2 filas por 3 columnas; total 6 elementos. Su disposición en memoria se vería así:



Desde nuestro punto de vista, cuando se trate de arrays de dos dimensiones, es más fácil pensar en ellos como si de una tabla de m fias por n columnas se tratara.
Por ejemplo,

c

	columna 0	columna 1	columna 2
fila 0	10	12	14
fila 1	16	18	20

Para acceder a los elementos del array c, puesto que se trata de un array de dos dimensiones, utilizaremos dos subíndices, el primero indicará la fila y el segundo la columna donde se localiza el elemento. Según esto, los elementos del array c son:

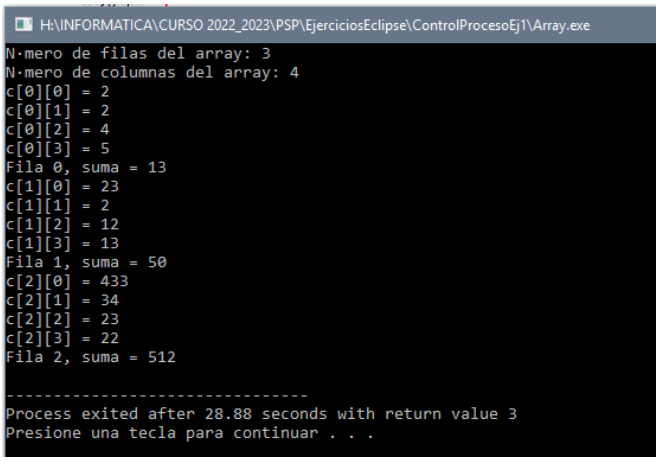
c	columna 0	columna 1	columna 2
fila 0	c[0][0]	c[0][1]	c[0][2]
fila 1	c[1][0]	c[1][1]	c[1][2]

Para acceder a un elemento en un array de dos dimensiones, se utiliza un subíndice que le indica al compilador cuántas filas hay que desplazarse, y otro que le indica cuántos elementos hay que avanzar en la fila actual, para situarse en dicho elemento. Así, para acceder al elemento c[1][2] hay que desplazarse a partir de c 1 fila, y avanzar 2 elementos sobre la fila actual. En definitiva, el cálculo que hace el compilador para saber cuántos elementos tiene que avanzar para acceder a un elemento cualquiera c[fila][columna] en un array de dos dimensiones es:

fila x elementos por fila + col

Ejemplo

Como ejemplo de aplicación de arrays multidimensionales, vamos a realizar un programa que asigne datos a un array c de dos dimensiones y a continuación escriba las sumas correspondientes a las filas del array.
La ejecución del programa presentará el aspecto siguiente:



Para ello, en primer lugar definimos como constante el número máximo filas y de columnas del array.

```
#define FILAS_MAX 10  /* número máximo de filas */
#define COLS_MAX  10  /* número máximo de columnas */
```

A continuación declaramos las variables

```
float c[FILAS_MAX][COLS_MAX];  /* array c de dos dimensiones */
float sumafila;        /* suma de los elementos de una fila */
int filas, cols;    /* filas y columnas del array de trabajo */
int fila, col;      /* fila y columna del elemento accedido */
```

Después, leemos el número de filas y de columnas que en realidad vamos a utilizar, comprobando que estos valores estén dentro del rango permitido; esto es:

```
do
{
 printf("Número de filas del array: ");
 scanf("%d", &filas);
}
while (filas < 1 || filas > FILAS_MAX);

do
{
 printf("Número de columnas del array: ");
 scanf("%d", &cols);
}
while (cols < 1 || cols > COLS_MAX);
```

El paso siguiente es asignar un valor desde el teclado a cada elemento del array

```
  for (col = 0; col < cols; col++)
  {
   printf("c[%d][%d] = ", fila, col);
   scanf("%f", &c[fila][col]);
  }
}
```

Una vez leído el array, calculamos y visualizamos la suma de cada fina

```
for (fila = 0; fila < filas; fila++)
{
 sumafila  = 0;
 for (col = 0; col < cols; col++)
   sumafila += c[fila][col];
 printf("Fila %d, suma = %g\n", fila, sumafila);
}
```

/******* Suma de las filas de un array bidimensional *********/**

```
#include <stdio.h>
#define FILAS_MAX 10  /* número máximo de filas */
#define COLS_MAX  10  /* número máximo de columnas */

void main()
{
  float c[FILAS_MAX][COLS_MAX];  /* array c de dos dimensiones */
  float sumafila;         /* suma de los elementos de una fila */
  int filas, cols;    /* filas y columnas del array de trabajo */
  int fila, col;       /* fila y columna del elemento accedido */

  do
  {
    printf("Número de filas del array: ");
    scanf("%d", &filas);
  }
  while (filas < 1 || filas > FILAS_MAX);

  do
  {
    printf("Número de columnas del array: ");
    scanf("%d", &cols);
  }
  while (cols < 1 || cols > COLS_MAX);

  /* Entrada de datos */
/*  for (fila = 0; fila < filas; fila++)
  {
    for (col = 0; col < cols; col++)
    {
      printf("c[%d][%d] = ", fila, col);
      scanf("%f", &c[fila][col]);
    }
  }
*/
  /* Escribir la suma de cada fila */
/*  printf("\n\n");
  for (fila = 0; fila < filas; fila++)
  {
    sumafila  = 0;
    for (col = 0; col < cols; col++)
      sumafila += c[fila][col];
    printf("Fila %d, suma = %g\n", fila, sumafila);
  }*/
for (fila = 0; fila < filas; fila++)
{
  sumafila  = 0;
```

```
  for (col = 0; col < cols; col++)
  {
   printf("c[%d][%d] = ", fila, col);
   scanf("%f", &c[fila][col]);
   sumafila += c[fila][col];
  }
  printf("Fila %d, suma = %g\n", fila, sumafila);
}
}
```

Arrays asociativos

Cuando el índice de un array es a su vez un dato, se dice que el array es asociativo. En estos casos, la solución del problema se hace más fácil si utilizamos esta coincidencia.
Por ejemplo, vamos a realizar un programa que cuente el número veces que aparece cada una de las letras de un texto introducido por el teclado continuación imprima el resultado. Para hacer el ejemplo sencillo, vamos a suponer que el texto sólo contiene letras minúsculas del alfabeto inglés (no hay ni letras acentuadas, ni la ll, ni la ñ). La solución podría ser de la forma siguiente:

Introducir texto. Para finalizar introducir la marca '*'

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
-------------------------------------------------------
5 1 0 3 4 0 0 0 8 0 0 6 3 6 7 0 0 2 11 1 2 0 0 0 2 1
```

Antes de empezar el problema vamos a ver algunos de los detalles que después utilizaremos en el programa. Por ejemplo:

```
#define N_ELEMENTOS 'z'-'a'+1 /*N_ELEMENTOS = 26 */
```

La directriz anterior define la constante N_ELEMENTOS con el valor 26. Recordemos, que cada carácter tiene asociado un valor entero (código ASCII) que es el que utlliza la máquina internamente para manipularlo. Así por ejemplo La 'z' tiene asociado el entero 122,la'a' el 97, etc. Según esto, la interpretación que ha hecho el preprocesador C de la directriz anterior es:

```
#define N_ELEMENTOS 122-97+1 /* N_ELEMENTOS = 26 */
```

Por la misma razón, si realizamos las declaraciones:

```
int c[256], car = 'a'; /* car tiene asignado el valor 97 */
```

La siguiente sentencia asigna a c[97]el valor diez,

```
c['a'] =10;
```

y esta otra sentencia que se muestra a continuación realiza la misma operación, porque car tiene asignado el carácter 'a'.

```
[car] = 10;
```

Entonces, si leemos un carácter (de la 'a' a la 'z')

```
car = getchar();
```

y a continuación realizamos la operación

```
c[car] ++;
```

¿***qué elemento del array c se ha incrementado***? La respuesta es, el de índice igual al código correspondiente al carácter leído. Hemos hecho coincidir el carácter leído con el índice del array. Así cada vez que leamos una 'a' se incrementará el contador c[97] o lo que es lo mismo c['a']; tenemos un contador de 'a'. Análogamente diremos para el resto de los caracteres.

Pero ¿qué pasa con los elementos c[0] a c[96]? Según hemos planteado el problema inicial (con qué frecuencia aparecen los caracteres de la 'a' a la 'z') quedarían sin utilizar. Esto, aunque no presenta ningún problema, se puede evitar así:

```
c[car – ‘a’]++;
```

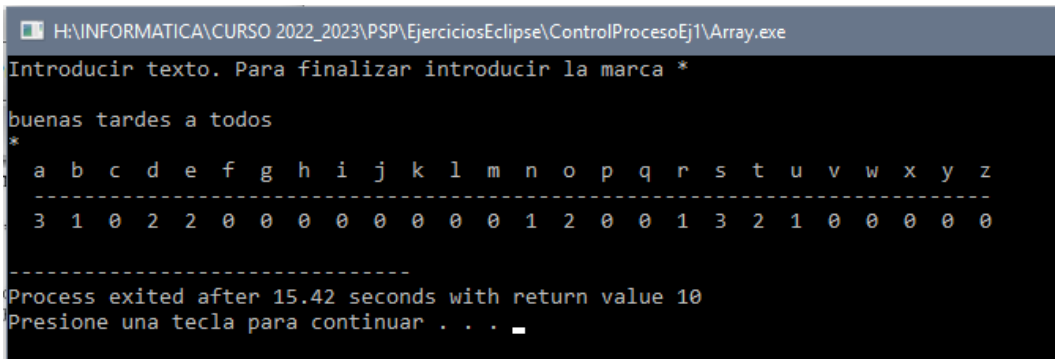
Pata car igual a 'a' se trataría del elemento c[0] y para car igual a 'z' se trataría del elemento c[26]. De esta forma podemos definir un array de enteros con un número de elementos igual al número de caracteres de la 'a' a la'z'. El primer elemento será el contador de 'a', el segundo el de 'b', y así sucesivamente.
Un contador es una variable que inicialmente vale cero (suponiendo que la cuenta empieza desde uno) y que después se incrementa en una unidad cada vez que ocurre el suceso que se desea contar.

El programa completo se muestra a continuación.

/******** Frecuencia de las letras en un texto **********/**

```
#include <stdio.h>
#define N_ELEMENTOS 'z'-'a'+1  /* número de elementos */

void main()
{
 int c[N_ELEMENTOS]; /* array c */
 char car; /* índice */

 /* Poner los elementos del array a cero */
 for (car = 'a'; car <= 'z'; car++)
  c[car - 'a'] = 0;

 /* Entrada de datos y cálculo de la tabla de frecuencias */
 printf("Introducir texto. Para finalizar introducir la marca *\n\n");
 while ((car = getchar()) != ‘*’)
 {
  /* Si el carácter leído está entre la 'a' y la 'z'
     incrementar el contador correspondiente */
  if (car >= 'a' && car <= 'z')
    c[car - 'a']++;
```

```
 }

 /* Escribir la tabla de frecuencias */
 for (car = 'a'; car <= 'z'; car++)
   printf("  %c", car);
 printf("\n  ----------------------------------------"
        "----------------------------------\n");

 for (car = 'a'; car <= 'z'; car++)
   printf("%3d", c[car - 'a']);
 putchar('\n');
}
```



Arrays internos static

Las variables static internas son locales a una función, pero a diferencia de variables automáticas (auto) su existencia es pernanente, en lugar de definirse al activarse la función y eliminarse al finalizar la misma. Esto es, las variables static internas proporcionan un medio de almacenamiento permanente y privado en una función y al igual que las variables globales son,inicializadas automáticamente a cero. Hay otra diferencia, las variables automáticas se definen en un área de memoria denominada pila (stack), mientras que las variables estáticas se definen en el área de memoria destinada a los datos globales.

Si en el programa definimos el array c estático, no es necesario poner sus elementos a cero. Por ejemplo,

```
void main()
{
static int c [N-ELEMENTOS]; /* array c */
char car; /* índice */

/* Entrada de datos y cá1culo de 1a tabla de frecuencias */
}
```