

# Chapter 6 - Arreglos

## Esquema

### 6.1 Introducción

### 6.2 Matrices

### 6.3 Declarar las matrices

### 6.4 Ejemplos de utilización de matrices

### 6.5 Pasar de un conjunto a otro de funciones

### 6.6 Ordenación de los conjuntos

### 6.7 Estudio de caso: Cálculo de la media, la mediana y el modo utilizando matrices

### 6.8 Búsqueda de matrices

### 6.9 Matrices con suscripción múltiple



## 6.1 Introducción

- *Arreglos*
  - Estructuras de los elementos de datos relacionados
  - La entidad estática - el mismo tamaño en todo el programa
  - Las estructuras de datos dinámicos que se examinan posteriormente



## 6.2 Arreglos

- Arreglo
  - Grupo de posiciones de memoria consecutivas
  - Mismo nombre y tipo
- Para referirse a un elemento, especifique
  - Nombre del Arreglo
  - Número de posición
- Formato: *nombre-arreglo* [*posición*]
  - Primer elemento en posición 0
  - n-ésimo elemento del arreglollamado **c**: **c[0]**, **c[1]**...**c[n-1]**

Nombre del arreglo  
(Obsérvese que todos los  
elementos de este arreglo  
tienen el mismo nombre, c)

↓

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

↑

Número de posición del  
elemento dentro del  
arreglo c



## 6.2 Arreglos (II)

- Los elementos del arreglo son como variables normales

```
c[0] = 3;  
printf( "%d", c[0] );
```

- Se puede realizar operaciones en subíndice. If **x** = 3,  
**c[5-2] == c[3] == c[x]**



## 6.3 Declarando Arreglos

- Al declarar los arreglos, especifique

- Nombre
- Tipo de arreglo
- Numero de elementos

```
TipoArreglo NombreArreglo[ numeroDeElementos ];  
int c[ 10 ];  
float myArray[ 3284 ];
```

- Declarando múltiples arreglos del mismo tipo

- Formato similar al de las variables regulares

```
int b[ 100 ], x[ 27 ];
```



## 6.4 Examples Using Arrays

- Initializers

```
int n[5] = {1, 2, 3, 4, 5};
```

- If not enough initializers, rightmost elements become 0
- If too many, syntax error

```
int n[5] = {0}
```

- All elements 0
- C arrays have no bounds checking

- If size omitted, initializers determine it

```
int n[] = { 1, 2, 3, 4, 5 };
```

- 5 initializers, therefore 5 element array





```
1  /* Fig. 6.8: fig06_08.c
2      Programa que imprime Histogramas */
3  #include <stdio.h>
4  #define SIZE 10
5
6  int main()
7  {
8      int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
9      int i, j;
10
11     printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );
12
13     for ( i = 0; i <= SIZE - 1; i++ ) {
14         printf( "%7d%13d", i, n[ i ] ) ;
15
16         for ( j = 1; j <= n[ i ]; j++ )    /* print one bar */
17             printf( "%c", '*' );
18
19         printf( "\n" );
20     }
21
22     return 0;
23 }
```

1. Inicializa arreglo

2. Bucle

3. Impime



## Outline



## Salida del Programa

Element	Value	Histogram
0	19	*****
1	3	***
2	15	*****
3	7	*****
4	11	*****
5	9	*****
6	13	*****
7	5	*****
8	17	*****
9	1	*



## 6.4 Ejemplo Usando Arreglos (II)

- Arreglo de Caracteres

- Cadena "**hello**" es realmente un arreglo estático de caracteres
- Arreglos de Caracteres puede ser inicializado usando literales de cuerda

```
char cadena1[] = "first";
```

- carácter nulo '**\0**' finaliza una cadena
- **cadena1** tiene 6 elementos

```
char cadena1[] = { 'f', 'i', 'r', 's', 't', '\0' };
```



## 6.4 Ejemplo Usando Arreglos (III)

- Arreglo de Cadenas (continuado)
  - Acceder a los personajes individuales
    - `cadena1[ 3 ]` es el carácter 's'
  - Nombre de Arreglo es la dirección del arreglo, así & no es necesario para función **scanf**  
`scanf( "%s", string2 ) ;`
    - Lee los caracteres hasta encontrar un espacio en blanco
    - Puede escribir más allá del fin del arreglo, sea cuidadoso



```

1  /* Fig. 6.10: fig06_10.c
2      Tratando arreglos de caracteres como cadenas */
3  #include <stdio.h>
4
5  int main()
6  {
7      char string1[ 20 ], string2[] = "string literal";
8      int i;
9
10     printf(" Ingrese una cadena: ");
11     scanf( "%s", string1 );
12     printf( "string1 es: %s\nstring2: es %s\n"
13            "string1 con espacios entre los caracteres es:\n",
14            string1, string2 );
15
16     for ( i = 0; string1[ i ] != '\0'; i++ )
17         printf( "%c ", string1[ i ] );
18
19     printf( "\n" );
20     return 0;
21 }

```



## Outline

1. Inicializa cadenas

2. Imprime Cadenas

2.1 Define Bucle

2.2 Imprime caracteres individualmente

```

Ingrese una cadena: Hello there
string1 es: Hello
string2 es: string literal
string1 con espacios entre los caracteres es:
H e l l o

```

Salida de Programa

## 6.5 Pasando los arreglos a las funciones

- Pasando arreglos

- Especificar el nombre del arreglo sin corchetes

```
int myArray[ 24 ] ;
```

```
myFunction( myArray, 24 ) ;
```

- El tamaño del arreglo suele pasar a la función
- Arreglos pasado con *llamada-por-referencia*
- Nombre del arreglo es la dirección del primer elemento
- Función sabe dónde está almacenada el arreglo
  - Modifica las ubicaciones de la memoria original

- Pasando elemento de arreglo

- Pasado por *llamada-por-valor*
- Pase el nombre suscrito (esto es, **myArray[3]**) a función



## 6.5 Pasando los arreglos a las funciones (II)

- Prototipo de Función

```
void modifyArray( int b[], int arraySize );
```

- El nombre de parámetros es opcional en los prototipos

- `int b[]` podría simplemente ser `int []`
- `int arraySize` podría simplemente ser `int`



```

1  /* Fig. 6.13: fig06 13.c
2  Pasar las arreglos y los elementos del arreglo a las funciones */
3  #include <stdio.h>
4  #define SIZE 5
5
6  void modifyArray( int [], int ); /* parece extraño */
7  void modifyElement( int );
8
9  int main()
10 {
11     int a[ SIZE ] = { 0, 1, 2, 3, 4 }, i;
12
13     printf( "Efecto de pasar un arreglo completo con llamada"
14            "por referencia:\n\nLos valores del "
15            "arreglo original son:\n" );
16
17     for ( i = 0; i <= SIZE - 1; i++ )
18         printf( "%3d", a[ i ] );
19
20     printf( "\n" );
21     modifyArray( a, SIZE ); /* passed call by reference */
22     printf( "Los valores de los arreglos modificados son:\n" );
23
24     for ( i = 0; i <= SIZE - 1; i++ )
25         printf( "%3d", a[ i ] );
26
27     printf( "\n\nEfecto de pasar un arreglo con llamada"
28            "por valor:\n\nEl valor de a[3] es %d\n", a[ 3 ] );
29     modifyElement( a[ 3 ] );
30     printf( "El valor de a[ 3 ] es %d\n", a[ 3 ] );
31     return 0;
32 }

```



## Outline



1. Definiciones de la función

2. Pasar arreglo para una función

2.1 Pasar el elemento de arreglo a una función

3. Print

```

1  /* Fig. 6.13: fig06 13.c
2  Pasar los arreglos y los elementos del arreglo a las funciones
3  #include <stdio.h>
4  #define SIZE 5
5
6  void modifyArray( int [], int ); /* parece extraño */
7  void modifyElement( int );
8
9  int main()
10 {
11     int a[ SIZE ] = { 0, 1, 2, 3, 4 }, i;
12
13     printf( "Efecto de pasar un arreglo completo con llamada"
14            "por referencia:\n\nLos valores del "
15            "arreglo original son:\n" );
16
17     for ( i = 0; i <= SIZE - 1; i++ )
18         printf( "%3d", a[ i ] );
19
20     printf( "\n" );
21     modifyArray( a, SIZE ); /* passed call by reference */
22     printf( "Los valores de los arreglos modificados son:\n"
23
24     for ( i = 0; i <= SIZE - 1; i++ )
25         printf( "%3d", a[ i ] );
26
27     printf( "\n\nEfecto de pasar un arreglo con llamada"
28            "por valor:\n\nEl valor de a[3] es %d\n", a[ 3 ]
29     modifyElement( a[ 3 ] );
30     printf( "El valor de a[ 3 ] es %d\n", a[ 3 ] );
31     return 0;
32 }

```



## Outline



1. Definiciones de la función

2. Pasar arreglo para una función

2.1 Pasar el elemento de arreglo a una función

Arreglo enteros pasadas con llamada por referencia, y pueden ser modificados

Los elementos del arreglo pasado con llamada por valor, y no pueden ser modificados

33

```
34 void modifyArray( int b[], int size )
```

```
35 {
```

```
36     int j;
```

```
37
```

```
38     for ( j = 0; j <= size - 1; j++ )
```

```
39         b[ j ] *= 2;
```

```
40 }
```

```
41
```

```
42 void modifyElement( int e )
```

```
43 {
```

```
44     printf( "Value in modifyElement is %d\n", e *= 2 );
```

```
45 }
```



## Outline



### 3.1 Definiciones de la función

Efecto de pasar arreglo entero con llamada por referencia:

Los valores del arreglo original son:

0 1 2 3 4

Los valores del arreglo modificado son:

0 2 4 6 8

Efecto de pasar elemento de arreglo con llamada por valor:

El valor de a[3] es 6

Valor en modifyElement es 12

El valor de a[3] es 6

Salid de Programa



## 6.6 Ordenar Arreglos

- Ordenamiento de datos
  - Importante aplicación informática
  - Prácticamente todas las organizaciones deben clasificar algunos datos
    - Las cantidades masivas deben ser ordenadas
- Bubble sort (sinking sort)
  - Varios pasos a través del arreglo
  - Sucesivamente pares de elementos son comparados
    - Si el orden crece (o identico), no hay cambios
    - Si el orden decrece, los elementos son intercambiados
  - Repetir
- Ejemplo:  
original: 3 4 2 6 7  
paso 1: 3 2 4 6 7  
paso 2: 2 3 4 6 7
  - Elements mas pequeños "burbujean" hacia arriba



## 6.7 Caso de Estudio: Calcular Media, Mediana, y Moda usando arreglos

- Media - promedio
- Mediana - número en el medio de una lista ordenada
  - 1, 2, 3, 4, 5  
3 es la mediana
- Moda - numero que ocurre más veces
  - 1, 1, 1, 2, 3, 3, 4, 5  
1 es la moda



```

1  /* Fig. 6.16: fig06 16.c
2      Este programa introduce el tema del análisis de datos de
3      Calcula la media, la mediana y la moda de los datos */
4  #include <stdio.h>
5  #define SIZE 99
6
7  void mean( const int [] );
8  void median( int [] );
9  void mode( int [], const int [] ) ;
10 void bubbleSort( int [] );
11 void printArray( const int [] );
12
13 int main()
14 {
15     int frequency[ 10 ] = { 0 };
16     int response[ SIZE ] =
17         { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
18           7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
19           6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
20           7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
21           6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
22           7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
23           5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
24           7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
25           7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
26           4, 5, 6, 1, 6, 5, 7, 8, 7 };
27
28     mean( response );
29     median( response );
30     mode( frequency, response );
31     return 0;
32 }

```



## Outline



### 1. Prototipos

#### 1.1 Inicializa arreglos

### 2. Llama functioness mean, median, and mode

```

33
34 void mean( const int answer[] )
35 {
36     int j, total = 0;
37
38     printf( "%s\n%s\n%s\n", "*****", "   Mean", "*****"
39
40     for ( j = 0; j <= SIZE - 1; j++ )
41         total += answer[ j ];
42
43     printf( "La media es el valor promedio de los datos\n"
44            "La media es igual al total de\n"
45            "todos los datos divididos por el número\n"
46            "de los elementos de datos( %d ). El valor medio
47            "est corrida es: %d / %d = %.4f\n\n",
48            SIZE, total, SIZE, ( double ) total / SIZE );
49 }
50
51 void median( int answer[] )
52 {
53     printf( "\n%s\n%s\n%s\n%s",
54            "*****", " Mediana", "*****",
55            "El arreglo desordenado de respuestas es" );
56
57     printArray( answer );
58     bubbleSort( answer );
59     printf( "\n\nLa arreglo ordenada es" );
60     printArray( answer );
61     printf( "\n\nLa mediana es el elemento %d of\n"
62            "el arreglo de %d elementos ordenado.\n"
63            "Para esta corrida la media es %d\n\n",
64            SIZE / 2, SIZE, answer[ SIZE / 2 ] );

```



## Outline



### 3. Define function mean

#### 3.1 Define function median

##### 3.1.1 Sort Array

##### 3.1.2 Print middle element



## 3.2 Define function mode

### 3.2.1 Increase frequency[] depending on response[]

```
65 }
66
67 void mode( int freq[], const int answer[] )
68 {
69     int rating, j, h, largest = 0, modeValue = 0;
70
71     printf( "\n%s\n%s\n%s\n",
72             "*****", "  Moda", "*****" );
73
74     for ( rating = 1; rating <= 9; rating++ )
75         freq[ rating ] = 0;
76
77     for ( j = 0; j <= SIZE - 1; j++ )
78         ++freq[ answer[ j ] ];
79
80     printf( "%s%11s%19s\n\n%54s\n%54s\n\n",
81             "Respuesta", "Frecuencia", "Histograma",
82             "1    1    2    2", "5    0    5    0    5" );
83
84     for ( rating = 1; rating <= 9; rating++ ) {
85         printf( "%8d%11d", rating, freq[ rating ] );
86
87         if ( freq[ rating ] > largest ) {
88             largest = freq[ rating ];
89             modeValue = rating;
90         }
91
92         for ( h = 1; h <= freq[ rating ]; h++ )
93             printf( "*" );
94     }
```

Obsérvese cómo el subíndice en frequency[] es el valor de un elemento en answer[]

Imprime estrellas dependiendo del valor de **frequency[]**

```

95     printf( "\n" );
96 }
97
98     printf( "La moda es el valor más frecuente.\n"
99           "Para esta corrida la moda es %d el cual ocurrido"
100          " %d veces.\n", modeValue, largest );
101 }
102
103 void bubbleSort( int a[] )
104 {
105     int pass, j, hold;
106
107     for ( pass = 1; pass <= SIZE - 1; pass++ )
108
109         for ( j = 0; j <= SIZE - 2; j++ )
110
111             if ( a[ j ] > a[ j + 1 ] ) {
112                 hold = a[ j ];
113                 a[ j ] = a[ j + 1 ];
114                 a[ j + 1 ] = hold;
115             }
116 }
117
118 void printArray( const int a[] )
119 {
120     int j;
121
122     for ( j = 0; j <= SIZE - 1; j++ ) {
123
124         if ( j % 20 == 0 )
125             printf( "\n" );

```



Outline



3.3 Define bubbleSort

3.3 Define printArray

Bubble sort: Si el elemento está fuera de orden, swap entonces

126

127           printf( "%2d", a[ j ] );

128        }

129    }



Outline



## Program Output

\*\*\*\*\*

Media

\*\*\*\*\*

La media es el valor promedio de los datos.

La media es igual al total de todos los datos

dividido por el número de elementos de datos (99).

El valor medio para esta ejecución es:  $681 / 99 = 6.8788$

\*\*\*\*\*

Mediana

\*\*\*\*\*

El arreglo desordenado de respuestas es

7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 8

6 7 8 9 3 9 8 7 8 7 7 8 9 8 9 8 9 7 8 9

6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5 3

5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 8

7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7

El arreglo ordenado es

1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 5

5 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7

7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8

8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8

9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

La mediana es el elemento 49 de

los 99 elementos del arreglo ordenado.

Para esta corrida la mediana es 7



## Program Output

```
*****
Moda
*****
Respuesta  Frecuencia  Histograma

          1    1    2    2
          5    0    5    0    5

1          1          *
2          3          ***
3          4          ****
4          5          *****
5          8          ********
6          9          ********
7         23          *****************
8         27          *****************
9         19          *****************

La moda es el valor con mayor frecuencia.
Para esta corrida la moda es 8 el cual ocurri3 27 veces.
```



## 6.8 Buscando arreglos: Búsqueda Lineal y Búsqueda Binaria

- Buscar en arreglo por un *valor clave*
- Búsqueda lineal
  - Simple
  - Compara cada elemento del arreglo con el valor clave
  - Útil para arreglos pequeños y desordenados



## 6.8 Buscando arreglos: Búsqueda Lineal y Búsqueda Binaria (II)

- Búsqueda binaria
  - Para arreglos ordenados
  - Compara el elemento del medio con la clave
    - Si igual, coincidencia encontrada
    - Si clave < medio, mira en la primera mitad del arreglo
    - Si clave > medio, mira en la última mitad
    - Repite
  - Muy rápido; a lo más  $n$  pasos, cuando  $2^n > \text{número de elementos}$ 
    - 30 element array takes at most 5 steps

$$2^5 > 30$$



## 6.9 Arreglos de Multiple-Subindices

- Arreglos de Multiple subindices
  - Tablas con filas y columnas( $m \times n$ )
  - Como matrices: especificar la fila, luego la columna

	Column 0	Column 1	Column 2	Column 3
Row 0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
Row 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
Row 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>

Diagram illustrating the structure of a 2D array (matrix) with 3 rows and 4 columns. The array is named `a`. The row subscript is the first index, and the column subscript is the second index. The diagram shows the array name `a` and the row subscript `2` pointing to the third row, and the column subscript `1` pointing to the second column, resulting in the element `a[2][1]`.



## 6.9 Arreglos de Multiple-Subindices (II)

- Inicialización

```
int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };
```

1	2
3	4

- Inicializadores agrupados por filas en llaves
- Si no es suficiente, los elementos no especificados son puestos a cero

```
int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };
```

1	0
3	4

- Referenciando elementos

- Especifique la fila, luego la columna

```
printf( "%d", b[ 0 ][ 1 ] );
```



```

1  /* Fig. 6.22: fig06 22.c
2      Ejemplo de matriz de doble sub indice */
3  #include <stdio.h>
4  #define STUDENTS 3
5  #define EXAMS 4
6
7  int minimum( const int [][] EXAMS , int, int );
8  int maximum( const int [][] EXAMS , int, int );
9  double average( const int [], int );
10 void printArray( const int [][] EXAMS , int, int );
11
12 int main()
13 {
14     int student;
15     const int studentGrades[ STUDENTS ][ EXAMS ] =
16         { { 77, 68, 86, 73 },
17           { 96, 87, 89, 78 },
18           { 70, 90, 86, 81 } };
19
20     printf( "The array is:\n" );
21     printArray( studentGrades, STUDENTS, EXAMS );
22     printf( "\n\nLowest grade: %d\nHighest grade: %d\n",
23           minimum( studentGrades, STUDENTS, EXAMS ),
24           maximum( studentGrades, STUDENTS, EXAMS ) );
25
26     for ( student = 0; student <= STUDENTS - 1; student++ )
27         printf( "The average grade for student %d is %.2f\n",
28               student,
29               average( studentGrades[ student ], EXAMS ) );
30
31     return 0;
32 }

```



## Outline



### 1. Initialize variables

#### 1.1 Define functions to take double scripted arrays

#### 1.2 Initialize studentgrades [][]

### 2. Call functions minimum, maximum, and average

Cada fila es un estudiante particular, cada columna son las notas del examen.



## 3. Define functions

```
33
34  /* Find the minimum grade */
35  int minimum( const int grades[][ EXAMS ],
36              int pupils, int tests )
37  {
38      int i, j, lowGrade = 100;
39
40      for ( i = 0; i <= pupils - 1; i++ )
41          for ( j = 0; j <= tests - 1; j++ )
42              if ( grades[ i ][ j ] < lowGrade )
43                  lowGrade = grades[ i ][ j ];
44
45      return lowGrade;
46  }
47
48  /* Find the maximum grade */
49  int maximum( const int grades[][ EXAMS ],
50              int pupils, int tests )
51  {
52      int i, j, highGrade = 0;
53
54      for ( i = 0; i <= pupils - 1; i++ )
55          for ( j = 0; j <= tests - 1; j++ )
56              if ( grades[ i ][ j ] > highGrade )
57                  highGrade = grades[ i ][ j ];
58
59      return highGrade;
60  }
61
62  /* Determine the average grade for a particular exam */
63  double average( const int setOfGrades[], int tests )
64  {
```

```

65     int i, total = 0;
66
67     for ( i = 0; i <= tests - 1; i++ )
68         total += setOfGrades[ i ];
69
70     return ( double ) total / tests;
71 }
72
73 /* Print the array */
74 void printArray( const int grades[][ EXAMS ],
75                 int pupils, int tests )
76 {
77     int i, j;
78
79     printf( "          [0]  [1]  [2]  [3]" );
80
81     for ( i = 0; i <= pupils - 1; i++ ) {
82         printf( "\nstudentGrades[%d] ", i );
83
84         for ( j = 0; j <= tests - 1; j++ )
85             printf( "%-5d", grades[ i ][ j ] );
86     }
87 }

```



## 3. Define functions



## Program Output

The array is:

	[0]	[1]	[2]	[3]
studentGrades[0]	77	68	86	73
studentGrades[1]	96	87	89	78
studentGrades[2]	70	90	86	81

Lowest grade: 68

Highest grade: 96

The average grade for student 0 is 76.00

The average grade for student 1 is 87.50

The average grade for student 2 is 81.75