Capitolo 6 - Array

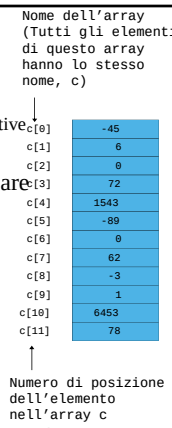
Outline

Array
Definizione degli Array
Esempi con Array
Array come parametri a funzioni
Sorting Arrays
Esercizi: Calcolo della Media, Mediana e Moda usando Array
Array Multidimensionali

Array

- Array
  - Gruppo di locazioni di memoria consecutive
  - Stesso nome e tipo
- Per riferirsi a un elemento, specificare
  - Nome dell'array
  - Posizione
- Formato:

  arrayname`[` *position number* `]`
  - Primo elemento a posizione `0`
  - array di `n` elementi di nome `c`:
    - `c[ 0 ], c[ 1 ]...c[ n – 1 ]`



Array

- Gli elementi di un array sono gestiti come normali variabili

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

- Operazioni eseguite con un indice.
  - Es. Se x è 3 allora la seguente è corretta

```
c[ 5 - 2 ] == c[ 3 ] == c[ x ]
```


Array

Operatori						Associatività	Tipo
`[]`	`()`					left to right	highest
`++`	`--`	`!`	*(type)*			right to left	unary
`*`	`/`	`%`				left to right	multiplicative
`+`	`-`					left to right	additive
`<`	`<=`	`>`	`>=`			left to right	relational
`==`	`!=`					left to right	equality
`&&`						left to right	logical and
`||`						left to right	logical or
`?:`						right to left	conditional
`=`	`+=`	`-=`	`*=`	`/=`	`%=`	right to left	assignment
`,`						left to right	comma

Fig. 6.2 Operator precedence.

Definizione degli Array

- Nella definizione degli array, specificare
  - Tipo dell'array
  - Nome
  - Numero di elementi
    ```
    arrayType arrayName[ numberOfElements ];
    ```
  - Esempi:
    ```
    int c[ 10 ];
    float myArray[ 3284 ];
    ```
- Definizione di array multipli dello stesso tipo
  - Formato simile alle variabili regolari
  - Esempio:
    ```
    int b[ 100 ], x[ 27 ];
    ```


Esempi con Array

- Inizializzazione
  ```
  int n[ 5 ] = { 1, 2, 3, 4, 5 };
  ```
  - Se non ci sono sufficienti valori di inizializzazione, gli elementi più a destra diventano `0`
    ```
    int n[ 5 ] = { 1 }
    ```
    - Tutti gli elementi hanno il valore 0, ad eccezione del primo che vale 1
  - Se ci sono troppo valori di inizializzazione, viene prodotto un errore sintattico
  - Gli array in C non hanno il controllo dei limiti
- Se la dimensione è omessa, gli inizializzatori la determinano
  ```
  int n[ ] = { 1, 2, 3, 4, 5 };
  ```
  5 inizializzatori, perciò array di 5 elementi

Outline

fig06_03.c

```
1  /* Fig. 6.3: fig06_03.c
2     Initializing an array */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int n[ 10 ]; /* n is an array of 10 integers */
9     int i;       /* counter */
10
11    /* initialize elements of array n to 0 */
12    for ( i = 0; i < 10; i++ ) {
13       n[ i ] = 0; /* set element at location i to 0 */
14    } /* end for */
15
16    printf( "%s%13s\n", "Element", "Value" );
17
18    /* output contents of array n in tabular format */
19    for ( i = 0; i < 10; i++ ) {
20       printf( "%7d%13d\n", i, n[ i ] );
21    } /* end for */
22
23    return 0; /* indicates successful termination */
24
25 } /* end main */
```


Outline

Program Output

```
Element        Value
      0            0
      1            0
      2            0
      3            0
      4            0
      5            0
      6            0
      7            0
      8            0
      9            0
```

Esempi con Array

- Array di caratteri
  - La stringa `"first"` è un array statico di caratteri
  - Gli arrays di caratteri possono essere inizializzati usando stringhe di letterali

```
char string1[] = "first";
```

    - Il carattere Null `'\0'` termina le stringhe
    - `string1` ha 6 elementi
      - E' equivalente a

```
char string1[] = { 'f', 'i', 'r', 's', 't', '\0' };
```

  - E' possibile accedere ai caratteri individuali
    `string1[ 3 ]` è il carattere 's'
  - Il nome dell'array è l'indirizzo dell'array, dunque & non è richiesto nella scanf

```
scanf( "%s", string2 );
```

    - Legge i caratteri fino a quando non si incontra uno spazio bianco
    - Attenzione: si può scrivere anche oltre i limiti di un array

Outline

fig06_04.c

```
1  /* Fig. 6.4: fig06_04.c
2     Initializing an array with an initializer list */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     /* use initializer list to initialize array n */
9     int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
10    int i; /* counter */
11
12    printf( "%s%13s\n", "Element", "Value" );
13
14    /* output contents of array in tabular format */
15    for ( i = 0; i < 10; i++ ) {
16       printf( "%7d%13d\n", i, n[ i ] );
17    } /* end for */
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```


Outline

Program Output

```
Element         Value
      0            32
      1            27
      2            64
      3            18
      4            95
      5            14
      6            90
      7            70
      8            60
      9            37
```


Outline

fig06_05.c

```
1  /* Fig. 6.5: fig06_05.c
2     Initialize the elements of array s to the even integers from 2 to 20 */
3  #include <stdio.h>
4  #define SIZE 10
5
6  /* function main begins program execution */
7  int main()
8  {
9     /* symbolic constant SIZE can be used to specify array size */
10    int s[ SIZE ]; /* array s has 10 elements */
11    int j;         /* counter */
12
13    for ( j = 0; j < SIZE; j++ ) { /* set the values */
14       s[ j ] = 2 + 2 * j;
15    } /* end for */
16
17    printf( "%s%13s\n", "Element", "Value" );
18
19    /* output contents of array s in tabular format */
20    for ( j = 0; j < SIZE; j++ ) {
21       printf( "%7d%13d\n", j, s[ j ] );
22    } /* end for */
23
24    return 0; /* indicates successful termination */
25
26 } /* end main */
```

```
Element         Value
      0             2
      1             4
      2             6
      3             8
      4            10
      5            12
      6            14
      7            16
      8            18
      9            20
```


Outline

Program Output


```c
/* Fig. 6.6: fig06_06.c
   Compute the sum of the elements of the array */
#include <stdio.h>
#define SIZE 12

/* function main begins program execution */
int main()
{
   /* use initializer list to initialize array */
   int a[ SIZE ] = { 1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45 };
   int i;         /* counter */
   int total = 0; /* sum of array */

   /* sum contents of array a */
   for ( i = 0; i < SIZE; i++ ) {
      total += a[ i ];
   } /* end for */

   printf( "Total of array element values is %d\n", total );

   return 0; /* indicates successful termination */

} /* end main */
```

```
Total of array element values is 383
```


Outline

fig06_06.c

Program Output


```c
/* Fig. 6.7: fig06_07.c
   Student poll program */
#include <stdio.h>
#define RESPONSE_SIZE 40 /* define array sizes */
#define FREQUENCY_SIZE 11

/* function main begins program execution */
int main()
{
   int answer; /* counter */
   int rating; /* counter */

   /* initialize frequency counters to 0 */
   int frequency[ FREQUENCY_SIZE ] = { 0 };

   /* place survey responses in array responses */
   int responses[ RESPONSE_SIZE ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
        1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
        5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
```


Outline

fig06_07.c (Part 1 of 2)


```c
   /* for each answer, select value of an element of array responses
      and use that value as subscript in array frequency to
      determine element to increment */
   for ( answer = 0; answer < RESPONSE_SIZE; answer++ ) {
      ++frequency[ responses [ answer ] ];
   } /* end for */

   /* display results */
   printf( "%s%17s\n", "Rating", "Frequency" );

   /* output frequencies in tabular format */
   for ( rating = 1; rating < FREQUENCY_SIZE; rating++ ) {
      printf( "%6d%17d\n", rating, frequency[ rating ] );
   } /* end for */

   return 0; /* indicates successful termination */

} /* end main */
```

```
Rating        Frequency
     1                2
     2                2
     3                2
     4                2
     5                5
     6               11
     7                5
     8                7
     9                1
    10                3
```


Outline

fig06_07.c (Part 2 of 2)

Program Output

```
1  /* Fig. 6.8: fig06_08.c
2     Histogram printing program */
3  #include <stdio.h>
4  #define SIZE 10
5
6  /* function main begins program execution */
7  int main()
8  {
9     /* use initializer list to initialize array n */
10    int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
11    int i; /* outer counter */
12    int j; /* inner counter */
13
14    printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );
15
16    /* for each element of array n, output a bar in histogram */
17    for ( i = 0; i < SIZE; i++ ) {
18       printf( "%7d%13d          ", i, n[ i ] );
19
20       for ( j = 1; j <= n[ i ]; j++ ) { /* print one bar */
21          printf( "%c", '*' );
22       } /* end inner for */
23
```


fig06_08.c (Part 1 of 2)


```
24       printf( "\n" ); /* start next line of output */
25    } /* end outer for */
26
27    return 0; /* indicates successful termination */
28
29 } /* end main */
```

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


fig06_08.c (Part 2 of 2)

Program Output


```
1  /* Fig. 6.9: fig06_09.c
2     Roll a six-sided die 6000 times */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6  #define SIZE 7
7
8  /* function main begins program execution */
9  int main()
10 {
11    int face;                    /* random number with value 1 - 6 */
12    int roll;                    /* roll counter */
13    int frequency[ SIZE ] = { 0 }; /* initialize array to 0 */
14
15    srand( time( NULL ) ); /* seed random-number generator */
16
17    /* roll die 6000 times */
18    for ( roll = 1; roll <= 6000; roll++ ) {
19       face = rand() % 6 + 1;
20       ++frequency[ face ]; /* replaces 26-line switch of Fig. 5.8 */
21    } /* end for */
22
23    printf( "%s%17s\n", "Face", "Frequency" );
24
```


fig06_09.c (Part 1 of 2)


```
25    /* output frequency elements 1-6 in tabular format */
26    for ( face = 1; face < SIZE; face++ ) {
27       printf( "%4d%17d\n", face, frequency[ face ] );
28    } /* end for */
29
30    return 0; /* indicates successful termination */
31
32 } /* end main */
```

```
Face        Frequency
   1             1029
   2              951
   3              987
   4             1033
   5             1010
   6              990
```


fig06_09.c (Part 2 of 2)

Program Output

```c
/* Fig. 6.10: fig06_10.c
   Treating character arrays as strings */
#include <stdio.h>

/* function main begins program execution */
int main()
{
   char string1[ 20 ];               /* reserves 20 characters */
   char string2[] = "string literal"; /* reserves 15 characters */
   int i;                            /* counter */

   /* read string from user into array string2 */
   printf("Enter a string: ");
   scanf( "%s", string1 );

   /* output strings */
   printf( "string1 is: %s\nstring2 is: %s\n"
           "string1 with spaces between characters is:\n",
           string1, string2 );

   /* output characters until null character is reached */
   for ( i = 0; string1[ i ] != '\0'; i++ ) {
      printf( "%c ", string1[ i ] );
   } /* end for */
```


Outline

fig06_10.c (Part 1 of 2)

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

   return 0; /* indicates successful termination */

} /* end main */
```

```
Enter a string: Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
```


Outline

fig06_10.c (Part 2 of 2)

```c
/* Fig. 6.11: fig06_11.c
   Static arrays are initialized to zero */
#include <stdio.h>

void staticArrayInit( void );    /* function prototype */
void automaticArrayInit( void ); /* function prototype */

/* function main begins program execution */
int main()
{
   printf( "First call to each function:\n" );
   staticArrayInit();
   automaticArrayInit();

   printf( "\n\nSecond call to each function:\n" );
   staticArrayInit();
   automaticArrayInit();

   return 0; /* indicates successful termination */

} /* end main */
```


Outline

fig06_11.c (Part 1 of 3)

```c
/* function to demonstrate a static local array */
void staticArrayInit( void )
{
   /* initializes elements to 0 first time function is called */
   static int array1[ 3 ];
   int i; /* counter */

   printf( "\nValues on entering staticArrayInit:\n" );

   /* output contents of array1 */
   for ( i = 0; i <= 2; i++ ) {
      printf( "array1[ %d ] = %d  ", i, array1[ i ] );
   } /* end for */

   printf( "\nValues on exiting staticArrayInit:\n" );

   /* modify and output contents of array1 */
   for ( i = 0; i <= 2; i++ ) {
      printf( "array1[ %d ] = %d  ", i, array1[ i ] += 5 );
   } /* end for */

} /* end function staticArrayInit */
```


Outline

fig06_11.c (Part 2 of 3)

```
/* function to demonstrate an automatic local array */
void automaticArrayInit( void )
{
   /* initializes elements each time function is called */
   int array2[ 3 ] = { 1, 2, 3 };
   int i; /* counter */

   printf( "\n\nValues on entering automaticArrayInit:\n" );

   /* output contents of array2 */
   for ( i = 0; i <= 2; i++ ) {
      printf("array2[ %d ] = %d  ", i, array2[ i ] );
   } /* end for */

   printf( "\nValues on exiting automaticArrayInit:\n" );

   /* modify and output contents of array2 */
   for ( i = 0; i <= 2; i++ ) {
      printf( "array2[ %d ] = %d  ", i, array2[ i ] += 5 );
   } /* end for */

} /* end function automaticArrayInit */
```


fig06_11.c (Part 3 of 3)


```
First call to each function:

Values on entering staticArrayInit:
array1[ 0 ] = 0  array1[ 1 ] = 0  array1[ 2 ] = 0
Values on exiting staticArrayInit:
array1[ 0 ] = 5  array1[ 1 ] = 5  array1[ 2 ] = 5

Values on entering automaticArrayInit:
array2[ 0 ] = 1  array2[ 1 ] = 2  array2[ 2 ] = 3
Values on exiting automaticArrayInit:
array2[ 0 ] = 6  array2[ 1 ] = 7  array2[ 2 ] = 8

Second call to each function:

Values on entering staticArrayInit:
array1[ 0 ] = 5  array1[ 1 ] = 5  array1[ 2 ] = 5
Values on exiting staticArrayInit:
array1[ 0 ] = 10  array1[ 1 ] = 10  array1[ 2 ] = 10

Values on entering automaticArrayInit:
array2[ 0 ] = 1  array2[ 1 ] = 2  array2[ 2 ] = 3
Values on exiting automaticArrayInit:
array2[ 0 ] = 6  array2[ 1 ] = 7  array2[ 2 ] = 8
```


Program Output

Array come parametri a funzioni

- Passaggio di array
  - Per passare un array come argomento a una funzione, specificare il nome dell'array senza parentesi quadre
    ```
    int myArray[ 24 ];
    myFunction( myArray, 24 );
    ```
    - La dimensione dell'array è in genere passata come ulteriore parametro
  - Gli array sono passati per riferimento
  - Il nome dell'array è l'indirizzo del suo primo elemento
  - La funzione conosce dove l'array è memorizzato
    - Vengono modificate le posizioni originali in memoria
- Passaggio di elementi singoli dell'array
  - Passaggio per valore
  - Passare il nome con l'indice (i.e., `myArray[ 3 ]`) alla funzione

Array come parametri a funzioni

- Prototipo di funzione
  ```
  void modifyArray( int b[], int arraySize );
  ```
  - I nomi dei parametri sono opzionali nel prototipo
    - `int b[]` potrebbe essere scritto `int []`
    - `int arraySize` potrebbe essere semplicemente `int`

```
1  /* Fig. 6.12: fig06_12.c
2     The name of an array is the same as &array[ 0 ] */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     char array[ 5 ]; /* define an array of size 5 */
9
10    printf( "    array = %p\n&array[0] = %p\n"
11       "  &array = %p\n",
12       array, &array[ 0 ], &array );
13
14    return 0; /* indicates successful termination */
15
16 } /* end main */
```

```
    array = 0012FF78
&array[0] = 0012FF78
  &array = 0012FF78
```


Outline

fig06_12.c

Program Output


```
1  /* Fig. 6.13: fig06_13.c
2     Passing arrays and individual array elements to functions */
3  #include <stdio.h>
4  #define SIZE 5
5
6  /* function prototypes */
7  void modifyArray( int b[], int size );
8  void modifyElement( int e );
9
10 /* function main begins program execution */
11 int main()
12 {
13    int a[ SIZE ] = { 0, 1, 2, 3, 4 }; /* initialize a */
14    int i; /* counter */
15
16    printf( "Effects of passing entire array by reference:\n\nThe "
17          "values of the original array are:\n" );
18
19    /* output original array */
20    for ( i = 0; i < SIZE; i++ ) {
21       printf( "%3d", a[ i ] );
22    } /* end for */
23
24    printf( "\n" );
25
```


Outline

fig06_13.c (Part 1 of 3)


```
26    /* pass array a to modifyArray by reference */
27    modifyArray( a, SIZE );
28
29    printf( "The values of the modified array are:\n" );
30
31    /* output modified array */
32    for ( i = 0; i < SIZE; i++ ) {
33       printf( "%3d", a[ i ] );
34    } /* end for */
35
36    /* output value of a[ 3 ] */
37    printf( "\n\n\nEffects of passing array element "
38          "by value:\n\nThe value of a[3] is %d\n", a[ 3 ] );
39
40    modifyElement( a[ 3 ] ); /* pass array element a[ 3 ] by value */
41
42    /* output value of a[ 3 ] */
43    printf( "The value of a[ 3 ] is %d\n", a[ 3 ] );
44
45    return 0; /* indicates successful termination */
46
47 } /* end main */
48
```


Outline

fig06_13.c (Part 2 of 3)


```
49 /* in function modifyArray, "b" points to the original array "a"
50    in memory */
51 void modifyArray( int b[], int size )
52 {
53    int j; /* counter */
54
55    /* multiply each array element by 2 */
56    for ( j = 0; j < size; j++ ) {
57       b[ j ] *= 2;
58    } /* end for */
59
60 } /* end function modifyArray */
61
62 /* in function modifyElement, "e" is a local copy of array element
63    a[ 3 ] passed from main */
64 void modifyElement( int e )
65 {
66    /* multiply parameter by 2 */
67    printf( "Value in modifyElement is %d\n", e *= 2 );
68 } /* end function modifyElement */
```


Outline

fig06_13.c (Part 3 of 3)

```
Effects of passing entire array by reference:

The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8


Effects of passing array element by value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[ 3 ] is 6
```


Outline

Program Output


```
1  /* Fig. 6.14: fig06_14.c
2     Demonstrating the const type qualifier with arrays */
3  #include <stdio.h>
4
5  void tryToModifyArray( const int b[] ); /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10    int a[] = { 10, 20, 30 }; /* initialize a */
11
12    tryToModifyArray( a );
13
14    printf("%d %d %d\n", a[ 0 ], a[ 1 ], a[ 2 ] );
15
16    return 0; /* indicates successful termination */
17
18 } /* end main */
19
```


Outline

fig06_14.c (Part 1 of 2)


```
20 /* in function tryToModifyArray, array b is const, so it cannot be
21    used to modify the original array a in main. */
22 void tryToModifyArray( const int b[] )
23 {
24    b[ 0 ] /= 2;    /* error */
25    b[ 1 ] /= 2;    /* error */
26    b[ 2 ] /= 2;    /* error */
27 } /* end function tryToModifyArray */
```

```
Compiling...
FIG06_14.C
fig06_14.c(24) : error C2166: l-value specifies const object
fig06_14.c(25) : error C2166: l-value specifies const object
fig06_14.c(26) : error C2166: l-value specifies const object
```


Outline

fig06_14.c (Part 2 of 2)

Program Output

Esercizi: Calcolo della Media, Mediana e Moda usando array

- Media
- Mediana – punto medio tra il max e il min di un insieme di valori
  - 1, 2, 3, 4, 5
  - 3 è la mediana
- Moda – numero che occore più spesso
  - 1, 1, 1, 2, 3, 3, 4, 5
  - 1 è la moda

```c
/* Fig. 6.16: fig06_16.c
   This program introduces the topic of survey data analysis.
   It computes the mean, median, and  mode of the data */
#include <stdio.h>
#define SIZE 99

/* function prototypes */
void mean( const int answer[] );
void median( int answer[] );
void mode( int freq[], const int answer[] ) ;
void bubbleSort( int a[] );
void printArray( const int a[] );

/* function main begins program execution */
int main()
{
   int frequency[ 10 ] = { 0 }; /* initialize array frequency */

```


Outline

fig06_16.c (Part 1 of 8)


```c
   /* initialize array response */
   int response[ SIZE ] =
      { 6, 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 };

   /* process responses */
   mean( response );
   median( response );
   mode( frequency, response );

   return 0; /* indicates successful termination */

} /* end main */

```


Outline

fig06_16.c (Part 2 of 8)


```c
/* calculate average of all response values */
void mean( const int answer[] )
{
   int j;          /* counter */
   int total = 0; /* variable to hold sum of array elements */

   printf( "%s\n%s\n%s\n", "**********", "  Mean", "********** );

   /* total response values */
   for ( j = 0; j < SIZE; j++ ) {
      total += answer[  j  ];
   } /* end for */

   printf( "The mean is the average value of the data\n"
           "items. The mean is equal to the total of\n"
           "all the data items divided by the number\n"
           "of data items ( %d ). The mean value for\n"
           "this run is: %d / %d = %.4f\n\n",
           SIZE, total, SIZE, (  double  ) total / SIZE );
} /* end function mean */

```


Outline

fig06_16.c (Part 3 of 8)


```c
/* sort array and determine median element's value */
void median( int answer[] )
{
   printf( "\n%s\n%s\n%s\n%s",
           "**********", " Median", "**********",
           "The unsorted array of responses is" );

   printArray( answer ); /* output unsorted array */

   bubbleSort( answer ); /* sort array */

   printf( "\n\nThe sorted array is" );
   printArray( answer ); /* output sorted array */

   /* display median element */
   printf( "\n\nThe median is element %d of\n"
           "the sorted %d element array.\n"
           "For this run the median is %d\n\n",
           SIZE / 2, SIZE, answer[ SIZE / 2 ] );
} /* end function median */

```


Outline

fig06_16.c (Part 4 of 8)

```
83 /* determine most frequent response */
84 void mode( int freq[], const int answer[] )
85 {
86    int rating;        /* counter */
87    int j;             /* counter */
88    int h;             /* counter */
89    int largest = 0;   /* represents largest frequency */
90    int modeValue = 0; /* respesents most frequent response */
91
92    printf( "\n%s\n%s\n%s\n",
93            "**********", "  Mode", "********** );
94
95    /* initialize frequencies to 0 */
96    for ( rating = 1; rating <= 9; rating++ ) {
97       freq[ rating ] = 0;
98    } /* end for */
99
100    /* summarize frequencies */
101   for ( j = 0; j < SIZE; j++ ) {
102      ++freq[ answer[ j ] ];
103   } /* end for */
104
```


Outline

fig06_16.c (Part 5 of 8)


```
105   /* output headers for result columns */
106   printf( "%s%11s%19s\n\n%5s\n%54s\n\n",
107           "Response", "Frequency", "Histogram",
108           "1    1    2    2", "5    0    5    0    5" );
109
110   /* output results */
111   for ( rating = 1; rating <= 9; rating++ ) {
112      printf( "%8d%11d          ", rating, freq[ rating ] );
113
114      /* keep track of mode value and largest frequency value */
115      if ( freq[ rating ] > largest ) {
116         largest = freq[ rating ];
117         modeValue = rating;
118      } /* end if */
119
120      /* output histogram bar representing frequency value */
121      for ( h = 1; h <= freq[ rating ]; h++ ) {
122         printf( "*" );
123      } /* end inner for */
124
125      printf( "\n" ); /* being new line of output */
126   } /* end outer for */
127
```


Outline

fig06_16.c (Part 6 of 8)


```
128   /* display the mode value */
129   printf( "The mode is the most frequent value.\n"
130           "For this run the mode is %d which occurred"
131           " %d times.\n", modeValue, largest );
132} /* end function mode */
133
134/* function that sorts an array with bubble sort algorithm */
135void bubbleSort( int a[] )
136{
137   int pass; /* counter */
138   int j;    /* counter */
139   int hold; /* temporary location used to swap elements */
140
141   /* loop to control number of passes */
142   for ( pass = 1; pass < SIZE; pass++ ) {
143
144      /* loop to control number of comparisons per pass */
145      for ( j = 0; j < SIZE - 1; j++ ) {
146
147         /* swap elements if out of order */
148         if ( a[ j ] > a[ j + 1 ] ) {
149            hold = a[ j ];
150            a[ j ] = a[ j + 1 ];
151            a[ j + 1 ] = hold;
152         } /* end if */
153
```


Outline

fig06_16.c (Part 7 of 8)


```
154      } /* end inner for */
155
156   } /* end outer for */
157
158} /* end function bubbleSort */
159
160/* output array contents (20 values per row) */
161void printArray( const int a[] )
162{
163   int j; /* counter */
164
165   /* output array contents */
166   for ( j = 0; j < SIZE; j++ ) {
167
168      if ( j % 20 == 0 ) { /* begin new line every 20 values */
169         printf( "\n" );
170      } /* end if */
171
172      printf( "%2d", a[ j ] );
173   } /* end for */
174
175} /* end function printArray */
```


Outline

fig06_16.c (Part 8 of 8)

45

Outline

Program Output

```
********
  Mean
********
The mean is the average value of the data
items. The mean is equal to the total of
all the data items divided by the number
of data items ( 99 ). The mean value for
this run is: 681 / 99 = 6.8788


********
 Median
********
The unsorted array of responses is
 6 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

The sorted array is
 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 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 8
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
```


46

Outline

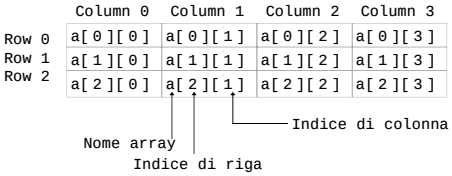
Program Output (continued)

```
The median is element 49 of
the sorted 99 element array.
For this run the median is 7


********
  Mode
********
Response  Frequency          Histogram

                                 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          *******************
The mode is the most frequent value.
For this run the mode is 8 which occurred 27 times.
```


47

Array Multidimensionali

- Array multidimensionali
  - Tabelle con righe e colonne (m x n array)
  - Come le matrici: specificare le righe, poi le colonne



48

Array Multidimensionali

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

1	2
3	4

  - Inizializzatori raggruppati per righe tra parentesi graffe
  - Se non sufficienti, gli elementi non specificati sono settati a zero

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

1	0
3	4

- Referenziazione degli elementi
  - Specificare la riga, poi la colonna

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

```
/* Fig. 6.21: fig06_21.c
   Initializing multidimensional arrays */
#include <stdio.h>

void printArray( const int a[][ 3 ] ); /* function prototype */

/* function main begins program execution */
int main()
{
   /* initialize array1, array2, array3 */
   int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
   int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
   int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };

   printf( "Values in array1 by row are:\n" );
   printArray( array1 );

   printf( "Values in array2 by row are:\n" );
   printArray( array2 );

   printf( "Values in array3 by row are:\n" );
   printArray( array3 );

   return 0; /* indicates successful termination */

} /* end main */
```


fig06_21.c (Part 1 of 2)


```
/* function to output array with two rows and three columns */
void printArray( const int a[][ 3 ] )
{
   int i; /* counter */
   int j; /* counter */

   /* loop through rows */
   for ( i = 0; i <= 1; i++ ) {

      /* output column values */
      for ( j = 0; j <= 2; j++ ) {
         printf( "%d ", a[ i ][ j ] );
      } /* end inner for */

      printf( "\n" ); /* start new line of output */
   } /* end outer for */

} /* end function printArray */
```

```
Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0
```


fig06_21.c (Part 2 of 2)

Program Output


```
/* Fig. 6.22: fig06_22.c
   Double-subscripted array example */
#include <stdio.h>
#define STUDENTS 3
#define EXAMS 4

/* function prototypes */
int minimum( const int grades[][ EXAMS ], int pupils, int tests );
int maximum( const int grades[][ EXAMS ], int pupils, int tests );
double average( const int setOfGrades[], int tests );
void printArray( const int grades[][ EXAMS ], int pupils, int tests );

/* function main begins program execution */
int main()
{
   int student; /* counter */

   /* initialize student grades for three students (rows) */
   const int studentGrades[ STUDENTS ][ EXAMS ] =
      { { 77, 68, 86, 73 },
        { 96, 87, 89, 78 },
        { 70, 90, 86, 81 } };
```


fig06_22.c (Part 1 of 6)


```
   /* output array studentGrades */
   printf( "The array is:\n" );
   printArray( studentGrades, STUDENTS, EXAMS );

   /* determine smallest and largest grade values */
   printf( "\n\nLowest grade: %d\nHighest grade: %d\n",
      minimum( studentGrades, STUDENTS, EXAMS ),
      maximum( studentGrades, STUDENTS, EXAMS ) );

   /* calculate average grade for each student */
   for ( student = 0; student <= STUDENTS - 1; student++ ) {
      printf( "The average grade for student %d is %.2f\n",
         student, average( studentGrades[ student ], EXAMS ) );
   } /* end for */

   return 0; /* indicates successful termination */

} /* end main */

/* Find the minimum grade */
int minimum( const int grades[][ EXAMS ], int pupils, int tests )
{
   int i;              /* counter */
   int j;              /* counter */
   int lowGrade = 100; /* initialize to highest possible grade */
```


fig06_22.c (Part 2 of 6)

```
   /* loop through rows of grades */
   for ( i = 0; i < pupils; i++ ) {

      /* loop through columns of grades */
      for ( j = 0; j < tests; j++ ) {

         if ( grades[ i ][ j ] < lowGrade ) {
            lowGrade = grades[ i ][ j ];
         } /* end if */

      } /* end inner for */

   } /* end outer for */

   return lowGrade; /* return minimum grade */

} /* end function minimum */

/* Find the maximum grade */
int maximum( const int grades[][ EXAMS ], int pupils, int tests )
{
   int i;            /* counter */
   int j;            /* counter */
   int highGrade = 0; /* initialize to lowest possible grade */

```

Outline

53

fig06_22.c (Part 3 of 6)


```
   /* loop through rows of grades */
   for ( i = 0; i < pupils; i++ ) {

      /* loop through columns of grades */
      for ( j = 0; j < tests; j++ ) {

         if ( grades[ i ][ j ] > highGrade ) {
            highGrade = grades[ i ][ j ];
         } /* end if */

      } /* end inner for */

   } /* end outer for */

   return highGrade; /* return maximum grade */

} /* end function maximum */

/* Determine the average grade for a particular student */
double average( const int setOfGrades[], int tests )
{
   int i;          /* counter */
   int total = 0; /* sum of test grades */

```

Outline

54

fig06_22.c (Part 4 of 6)


```
   /* total all grades for one student */
   for ( i = 0; i < tests; i++ ) {
      total += setOfGrades[ i ];
   } /* end for */

   return ( double ) total / tests; /* average */

} /* end function average */

/* Print the array */
void printArray( const int grades[][ EXAMS ], int pupils, int tests )
{
   int i; /* counter */
   int j; /* counter */

   /* output column heads */
   printf( "                [0]  [1]  [2]  [3]" );

   /* output grades in tabular format */
   for ( i = 0; i < pupils; i++ ) {

      /* output label for row */
      printf( "\nstudentGrades[%d] ", i );

```

Outline

55

fig06_22.c (Part 5 of 6)


```
      /* output grades for one student */
      for ( j = 0; j < tests; j++ ) {
         printf( "%-5d", grades[ i ][ j ] );
      } /* end inner for */

   } /* end outer for */

} /* end function printArray */

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
```

Outline

56

fig06_22.c (Part 6 of 6)