Illustrations List *(Main Page)*

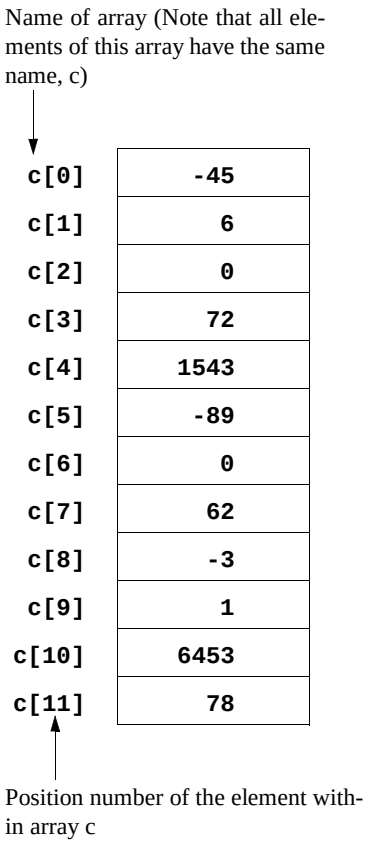

Fig. 4.1 A 12-element array.

Operators						Associativity	Type
`()`	`[]`					left to right	highest
`++`	`--`	`+`	`-`	`!`	`static_cast<`*type*`>( )`	right to left	unary
`*`	`/`	`%`				left to right	multiplicative
`+`	`-`					left to right	additive
`<<`	`>>`					left to right	insertion/extraction
`<`	`<=`	`>`	`>=`			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. 4.2 Operator precedence and associativity.

```
1  // Fig. 4.3: fig04_03.cpp
2  // initializing an array
3  #include <iostream.h>
4  #include <iomanip.h>
5
6  int main()
7  {
8     int i, n[ 10 ];
9
10    for ( i = 0; i < 10; i++ )       // initialize array
11       n[ i ] = 0;
12
13    cout << "Element" << setw( 13 ) << "Value" << endl;
14
15    for ( i = 0; i < 10; i++ )       // print array
16       cout << setw( 7 ) << i << setw( 13 ) << n[ i ] << endl;
17
18    return 0;
19 }
```

Fig. 4.3 Initializing the elements of an array to zeros (part 1 of 2).

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

Fig. 4.3 Initializing the elements of an array to zeros (part 2 of 2).

```
1  // Fig. 4.4: fig04_04.cpp
2  // Initializing an array with a declaration
3  #include <iostream.h>
4  #include <iomanip.h>
5
6  int main()
7  {
8     int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
9
10    cout << "Element" << setw( 13 ) << "Value" << endl;
11
12    for ( int i = 0; i < 10; i++ )
13       cout << setw( 7 ) << i << setw( 13 ) << n[ i ] << endl;
14
15    return 0;
16 }
```

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

Fig. 4.4 Initializing the elements of an array with a declaration.

```
1  // Fig. 4.5: fig04_05.cpp
2  // Initialize array s to the even integers from 2 to 20.
3  #include <iostream.h>
4  #include <iomanip.h>
5
6  int main()
7  {
8     const int arraySize = 10;
9     int j, s[ arraySize ];
10
11    for ( j = 0; j < arraySize; j++ )   // set the values
12       s[ j ] = 2 + 2 * j;
```

Fig. 4.5 Generating values to be placed into elements of an array (part 1 of 2).

```
13
14    cout << "Element" << setw( 13 ) << "Value" << endl;
15
16    for ( j = 0; j < arraySize; j++ )   // print the values
17       cout << setw( 7 ) << j << setw( 13 ) << s[ j ] << endl;
18
19    return 0;
20 }
```

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

Fig. 4.5 Generating values to be placed into elements of an array (part 2 of 2).

```
1  // Fig. 4.6: fig04_06.cpp
2  // Using a properly initialized constant variable
3  #include <iostream.h>
4
5  int main()
6  {
7     const int x = 7;  // initialized constant variable
8
9     cout << "The value of constant variable x is: "
10         << x << endl;
11
12    return 0;
13 }
```

Fig. 4.6 Correctly initializing and using a constant variable (part 1 of 2).

```
The value of constant variable x is: 7
```

Fig. 4.6 Correctly initializing and using a constant variable (part 2 of 2).

```
1  // Fig. 4.7: fig04_07.cpp
2  // A const object must be initialized
3
4  int main()
5  {
6     const int x;  // Error: x must be initialized
7
8     x = 7;        // Error: cannot modify a const variable
9
10    return 0;
11 }
```

```
Compiling FIG04_7.CPP:
Error FIG04_7.CPP 6: Constant variable 'x' must be
   initialized
Error FIG04_7.CPP 8: Cannot modify a const object
```

Fig. 4.7 A `const` object must be initialized.

```
1  // Fig. 4.8: fig04_08.cpp
2  // Compute the sum of the elements of the array
3  #include <iostream.h>
4
5  int main()
6  {
7     const int arraySize = 12;
8     int a[ arraySize ] = { 1, 3, 5, 4, 7, 2, 99,
9                            16, 45, 67, 89, 45 };
10    int total = 0;
11
12    for ( int i = 0; i < arraySize ; i++ )
13       total += a[ i ];
14
15    cout << "Total of array element values is " << total << endl;
16    return 0;
17 }
```

Fig. 4.8 Computing the sum of the elements of an array (part 1 of 2).

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

Fig. 4.8 Computing the sum of the elements of an array (part 2 of 2).

```
1  // Fig. 4.9: fig04_09.cpp
2  // Student poll program
3  #include <iostream.h>
4  #include <iomanip.h>
5
6  int main()
7  {
8     const int responseSize = 40, frequencySize = 11;
9     int responses[ responseSize ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8,
10        10, 1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7,
11        5, 6, 6, 5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
12    int frequency[ frequencySize ] = { 0 };
13
14    for ( int answer = 0; answer < responseSize; answer++ )
15       ++frequency[ responses[ answer ] ];
16
17    cout << "Rating" << setw( 17 ) << "Frequency" << endl;
18
19    for ( int rating = 1; rating < frequencySize; rating++ )
20       cout << setw( 6 ) << rating
21            << setw( 17 ) << frequency[ rating ] << endl;
22
23    return 0;
24 }
```

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

Fig. 4.9 A student poll analysis program.

```
1  // Fig. 4.10: fig04_10.cpp
2  // Histogram printing program
3  #include <iostream.h>
4  #include <iomanip.h>
5
6  int main()
7  {
8     const int arraySize = 10;
9     int n[ arraySize ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
10
11    cout << "Element" << setw( 13 ) << "Value"
12         << setw( 17 ) << "Histogram" << endl;
13
14    for ( int i = 0; i < arraySize ; i++ ) {
15       cout << setw( 7 ) << i << setw( 13 )
16            << n[ i ] << setw( 9 );
17
18       for ( int j = 0; j < n[ i ]; j++ )   // print one bar
19          cout << '*';
20
21       cout << endl;
22    }
23
24    return 0;
25 }
```

Fig. 4.10 A program that prints histograms (part 1 of 2).

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

Fig. 4.10 A program that prints histograms (part 2 of 2).

```
1  // Fig. 4.11: fig04_11.cpp
2  // Roll a six-sided die 6000 times
3  #include <iostream.h>
4  #include <iomanip.h>
5  #include <stdlib.h>
6  #include <time.h>
7
8  int main()
9  {
10    const int arraySize = 7;
11    int face, frequency[ arraySize ] = { 0 };
12
13    srand( time( 0 ) );
14
15    for ( int roll = 1; roll <= 6000; roll++ )
16       ++frequency[ 1 + rand() % 6 ]; // replaces 20-line switch
17                                      // of Fig. 3.8
18
19    cout << "Face" << setw( 13 ) << "Frequency" << endl;
20
```

Fig. 4.11 Dice-rolling program using arrays instead of `switch` (part 1 of 2).

```
21    // ignore element 0 in the frequency array
22    for ( face = 1; face < arraySize ; face++ )
23       cout << setw( 4 ) << face
24            << setw( 13 ) << frequency[ face ] << endl;
25
26    return 0;
27 }
```

```
Face    Frequency
   1         1037
   2          987
   3         1013
   4         1028
   5          952
   6          983
```

Fig. 4.11 Dice-rolling program using arrays instead of `switch` (part 2 of 2)

```
1  // Fig. 4_12: fig04_12.cpp
2  // Treating character arrays as strings
3  #include <iostream.h>
4
5  int main()
6  {
7     char string1[ 20 ], string2[] = "string literal";
8
9     cout << "Enter a string: ";
10    cin >> string1;
11    cout << "string1 is: " << string1
12         << "\nstring2 is: " << string2
13         << "string1 with spaces between characters is:\n";
14
15    for ( int i = 0; string1[ i ] != '\0'; i++ )
16       cout << string1[ i ] << ' ';
17
18    cin >> string1;  // reads "there"
19    cout << "\nstring1 is: " << string1 << endl;
20
21    cout << endl;
22    return 0;
23 }
```

Fig. 4.12 Treating character arrays as strings (part 1 of 2).

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

Fig. 4.12 Treating character arrays as strings (part 2 of 2).

```
1  // Fig. 4.13: fig04_13.cpp
2  // Static arrays are initialized to zero
3  #include <iostream.h>
4
5  void staticArrayInit( void );
6  void automaticArrayInit( void );
```

Fig. 4.13 Comparing **`static`** array initialization and automatic array initialization (part 1 of 3).

```
7
8  int main()
9  {
10    cout << "First call to each function:\n";
11    staticArrayInit();
12    automaticArrayInit();
13
14    cout << "\n\nSecond call to each function:\n";
15    staticArrayInit();
16    automaticArrayInit();
17    cout << endl;
18
19    return 0;
20 }
21
22 // function to demonstrate a static local array
23 void staticArrayInit( void )
24 {
25    static int array1[ 3 ];
26    int i;
27
28    cout << "\nValues on entering staticArrayInit:\n";
29
30    for ( i = 0; i < 3; i++ )
31       cout << "array1[" << i << "] = " << array1[ i ] << "  ";
32
33    cout << "\nValues on exiting staticArrayInit:\n";
34
35    for ( i = 0; i < 3; i++ )
36       cout << "array1[" << i << "] = "
37            << ( array1[ i ] += 5 ) << "  ";
38 }
39
40 // function to demonstrate an automatic local array
41 void automaticArrayInit( void )
42 {
43    int i, array2[ 3 ] = { 1, 2, 3 };
44
45    cout << "\n\nValues on entering automaticArrayInit:\n";
46
47    for ( i = 0; i < 3; i++ )
48       cout << "array2[" << i << "] = " << array2[ i ] << "  ";
49
50    cout << "\nValues on exiting automaticArrayInit:\n";
51
52    for ( i = 0; i < 3; i++ )
53       cout << "array2[" << i << "] = "
54            << ( array2[ i ] += 5 ) << "  ";
55 }
```

Fig. 4.13 Comparing **`static`** array initialization and automatic array initialization (part 2 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
```

Fig. 4.13 Comparing `static` array initialization and automatic array initialization (part 3 of 3).

```
1  // Fig. 4.14: fig04_14.cpp
2  // Passing arrays and individual array elements to functions
3  #include <iostream.h>
4  #include <iomanip.h>
5
6  void modifyArray( int [], int );  // appears strange
7  void modifyElement( int );
8
9  int main()
10 {
11    const int arraySize = 5;
12    int i, a[ arraySize ] = { 0, 1, 2, 3, 4 };
13
14    cout << "Effects of passing entire array call-by-reference:"
15         << "\n\nThe values of the original array are:\n";
16
17    for ( i = 0; i < arraySize; i++ )
18       cout << setw( 3 ) << a[ i ];
19
20    cout << endl;
21
22    // array a passed call-by-reference
23    modifyArray( a, arraySize );
24
25    cout << "The values of the modified array are:\n";
26
27    for ( i = 0; i < arraySize; i++ )
28       cout << setw( 3 ) << a[ i ];
29
30    cout << "\n\n\n"
31         << "Effects of passing array element call-by-value:"
```

```
32          << "\n\nThe value of a[3] is " << a[ 3 ] << '\n';
```

Fig. 4.14 Passing arrays and individual array elements to functions (part 1 of 2).

```
33
34     modifyElement( a[ 3 ] );
35
36     cout << "The value of a[3] is " << a[ 3 ] << endl;
37
38     return 0;
39  }
40
41  void modifyArray( int b[], int sizeOfArray )
42  {
43     for ( int j = 0; j < sizeOfArray; j++ )
44        b[ j ] *= 2;
45  }
46
47  void modifyElement( int e )
48  {
49     cout << "Value in modifyElement is "
50          << ( e *= 2 ) << endl;
51  }
```

```
Effects of passing entire array call-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 call-by-value:

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

Fig. 4.14 Passing arrays and individual array elements to functions (part 2 of 2).

```
1  // Fig. 4.15: fig04_15.cpp
2  // Demonstrating the const type qualifier
3  #include <iostream.h>
4
5  void tryToModifyArray( const int [] );
6
7  int main()
8  {
9     int a[] = { 10, 20, 30 };
10
11    tryToModifyArray( a );
12    cout << a[ 0 ] << ' ' << a[ 1 ] << ' ' << a[ 2 ] << '\n';
13    return 0;
14 }
15
16 void tryToModifyArray( const int b[] )
17 {
18    b[ 0 ] /= 2;    // error
19    b[ 1 ] /= 2;    // error
20    b[ 2 ] /= 2;    // error
21 }
```

```
Compiling FIG04_15.CPP:
Error FIG04_15.CPP 18: Cannot modify a const object
Error FIG04_15.CPP 19: Cannot modify a const object
Error FIG04_15.CPP 20: Cannot modify a const object
Warning FIG04_15.CPP 21: Parameter 'b' is never used
```

Fig. 4.15 Demonstrating the `const` type qualifier.

```
1  // Fig. 4.16: fig04_16.cpp
2  // This program sorts an array's values into
3  // ascending order
4  #include <iostream.h>
5  #include <iomanip.h>
6
7  int main()
8  {
9     const int arraySize = 10;
10    int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
11    int i, hold;
12
13    cout << "Data items in original order\n";
14
15    for ( i = 0; i < arraySize; i++ )
16       cout << setw( 4 ) << a[ i ];
17
18    for ( int pass = 0; pass < arraySize - 1; pass++ ) // passes
19
20       for ( i = 0; i < arraySize - 1; i++ )      // one pass
21
22          if ( a[ i ] > a[ i + 1 ] ) {      // one comparison
23             hold = a[ i ];                 // one swap
24             a[ i ] = a[ i + 1 ];
25             a[ i + 1 ] = hold;
26          }
27
28    cout << "\nData items in ascending order\n";
29
```

Fig. 4.16 Sorting an array with bubble sort (part 1 of 2).

```
30    for ( i = 0; i < arraySize; i++ )
31       cout << setw( 4 ) << a[ i ];
32
33    cout << endl;
34    return 0;
35 }
```

```
Data items in original order
   2   6   4   8  10  12  89  68  45  37
Data items in ascending order
   2   4   6   8  10  12  37  45  68  89
```

Fig. 4.16 Sorting an array with bubble sort (part 2 of 2).

```
1  // Fig. 4.17: fig04_17.cpp
2  // This program introduces the topic of survey data analysis.
3  // It computes the mean, median, and mode of the data.
4  #include <iostream.h>
5  #include <iomanip.h>
6
7  void mean( const int [], int );
8  void median( int [], int );
9  void mode( int [], int [], int );
10 void bubbleSort( int[], int );
11 void printArray( const int[], int );
12
13 int main()
14 {
15    const int responseSize = 99;
16    int frequency[ 10 ] = { 0 },
17        response[ responseSize ] =
18           { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
19             7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
20             6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
21             7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
22             6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
23             7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
24             5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
25             7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
26             7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
27             4, 5, 6, 1, 6, 5, 7, 8, 7 };
28
29    mean( response, responseSize );
30    median( response, responseSize );
31    mode( frequency, response, responseSize );
32
33    return 0;
34 }
35
36 void mean( const int answer[], int arraySize )
37 {
38    int total = 0;
39
40    cout << "********\n  Mean\n********\n";
41
42    for ( int j = 0; j < arraySize; j++ )
43       total += answer[ j ];
44
```

Fig. 4.17 Survey data analysis program (part 1 of 3).

```
45    cout << "The mean is the average value of the data\n"
46         << "items. The mean is equal to the total of\n"
47         << "all the data items divided by the number\n"
48         << "of data items (" << arraySize
49         << "). The mean value for\nthis run is: "
50         << total << " / " << arraySize << " = "
51         << setiosflags( ios::fixed | ios::showpoint )
52         << setprecision( 4 ) << ( float ) total / arraySize
53         << "\n\n";
54 }
55
56 void median( int answer[], int size )
57 {
58    cout << "\n********\n Median\n********\n"
```

```
59          << "The unsorted array of responses is";
60
61       printArray( answer, size );
62       bubbleSort( answer, size );
63       cout << "\n\nThe sorted array is";
64       printArray( answer, size );
65       cout << "\n\nThe median is element " << size / 2
66            << " of\nthe sorted " << size
67            << " element array.\nFor this run the median is "
68            << answer[ size / 2 ] << "\n\n";
69    }
70
71    void mode( int freq[], int answer[], int size )
72    {
73       int rating, largest = 0, modeValue = 0;
74
75       cout << "\n********\n  Mode\n********\n";
76
77       for ( rating = 1; rating <= 9; rating++ )
78          freq[ rating ] = 0;
79
80       for ( int j = 0; j < size; j++ )
81          ++freq[ answer[ j ] ];
82
83       cout << "Response"<< setw( 11 ) << "Frequency"
84            << setw( 19 ) << "Histogram\n\n" << setw( 55 )
85            << "1    1    2    2\n" << setw( 56 )
86            << "5    0    5    0    5\n\n";
87
88       for ( rating = 1; rating <= 9; rating++ ) {
89          cout << setw( 8 ) << rating << setw( 11 )
90               << freq[ rating ] << "          ";
91
92          if ( freq[ rating ] > largest ) {
93             largest = freq[ rating ];
94             modeValue = rating;
95          }
```

Fig. 4.17 Survey data analysis program (part 2 of 3).

```
96
97          for ( int h = 1; h <= freq[ rating ]; h++ )
98             cout << '*';
99
100         cout << '\n';
101      }
102
103      cout << "The mode is the most frequent value.\n"
104           << "For this run the mode is " << modeValue
105           << " which occurred " << largest << " times." << endl;
106   }
107
108   void bubbleSort( int a[], int size )
109   {
110      int hold;
111
112      for ( int pass = 1; pass < size; pass++ )
113
114         for ( int j = 0; j < size - 1; j++ )
115
116            if ( a[ j ] > a[ j + 1 ] ) {
117               hold = a[ j ];
118               a[ j ] = a[ j + 1 ];
```

```
119             a[ j + 1 ] = hold;
120          }
121    }
122
123    void printArray( const int a[], int size )
124    {
125       for ( int j = 0; j < size; j++ ) {
126
127          if ( j % 20 == 0 )
128             cout << endl;
129
130          cout << setw( 2 ) << a[ j ];
131       }
132    }
```

Fig. 4.17 Survey data analysis program (part 3 of 3).

```
********
  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 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

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.
```

Fig. 4.18 Sample run for the survey data analysis program.

```
1  // Fig. 4.19: fig04_19.cpp
2  // Linear search of an array
3  #include <iostream.h>
4
5  int linearSearch( const int [], int, int );
6
7  int main()
8  {
9     const int arraySize = 100;
10    int a[ arraySize ], searchKey, element;
11
12    for ( int x = 0; x < arraySize; x++ )  // create some data
13       a[ x ] = 2 * x;
14
15    cout << "Enter integer search key:" << endl;
16    cin >> searchKey;
17    element = linearSearch( a, searchKey, arraySize );
18
19    if ( element != -1 )
20       cout << "Found value in element " << element << endl;
21    else
22       cout << "Value not found" << endl;
23
24    return 0;
25 }
```

Fig. 4.19 Linear search of an array (part 1 of 2).

```
26
27 int linearSearch( const int array[], int key, int sizeOfArray )
28 {
29    for ( int n = 0; n < sizeOfArray; n++ )
30       if ( array[ n ] == key )
31          return n;
32
33    return -1;
34 }
```

```
Enter integer search key:
36
Found value in element 18
```

```
Enter integer search key:
37
Value not found
```

Fig. 4.19 Linear search of an array (part 2 of 2).

```
1  // Fig. 4.20: fig04_20.cpp
2  // Binary search of an array
3  #include <iostream.h>
4  #include <iomanip.h>
5
6  int binarySearch( int [], int, int, int, int );
7  void printHeader( int );
8  void printRow( int [], int, int, int, int );
9
10 int main()
11 {
12    const int arraySize = 15;
13    int a[ arraySize ], key, result;
14
15    for ( int i = 0; i < arraySize; i++ )
16       a[ i ] = 2 * i;   // place some data in array
17
18    cout << "Enter a number between 0 and 28: ";
19    cin >> key;
20
21    printHeader( arraySize );
22    result = binarySearch( a, key, 0, arraySize - 1, arraySize );
23
```

Fig. 4.20 Binary search of a sorted array (part 1 of 4).

```
24    if ( result != -1 )
25       cout << '\n' << key << " found in array element "
26            << result << endl;
27    else
28       cout << '\n' << key << " not found" << endl;
29
30    return 0;
31 }
32
33 // Binary search
34 int binarySearch( int b[], int searchKey, int low, int high,
35                   int size )
36 {
37    int middle;
38
39    while ( low <= high ) {
40       middle = ( low + high ) / 2;
41
42       printRow( b, low, middle, high, size );
43
44       if ( searchKey == b[ middle ] )  // match
45          return middle;
46       else if ( searchKey < b[ middle ] )
47          high = middle - 1;        // search low end of array
48       else
49          low = middle + 1;         // search high end of array
50    }
51
52    return -1;   // searchKey not found
53 }
54
55 // Print a header for the output
56 void printHeader( int size )
57 {
58    cout << "\nSubscripts:\n";
59    for ( int i = 0; i < size; i++ )
```

```
60          cout << setw( 3 ) << i << ' ';
61
62       cout << '\n';
63
64       for ( i = 1; i <= 4 * size; i++ )
65          cout << '-';
66
67       cout << endl;
68    }
69
```

Fig. 4.20 Binary search of a sorted array (part 2 of 4).

```
70    // Print one row of output showing the current
71    // part of the array being processed.
72    void printRow( int b[], int low, int mid, int high, int size )
73    {
74       for ( int i = 0; i < size; i++ )
75          if ( i < low || i > high )
76             cout << "    ";
77          else if ( i == mid )           // mark middle value
78             cout << setw( 3 ) << b[ i ] << '*';
79          else
80             cout << setw( 3 ) << b[ i ] << ' ';
81
82       cout << endl;
83    }
```

```
Enter a number between 0 and 28: 25

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
-------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
                                 16  18  20  22* 24  26  28
                                                 24  26* 28
                                                 24*

25 not found
```

```
Enter a number between 0 and 28: 8

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
-------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
  0   2   4   6*  8  10  12
                  8  10* 12
                  8*

8 found in array element 4
```

Fig. 4.20 Binary search of a sorted array (part 3 of 4).

```
Enter a number between 0 and 28: 6

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
---------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
  0   2   4   6*  8  10  12

6 found in array element 3
```

Fig. 4.20 Binary search of a sorted array (part 4 of 4).

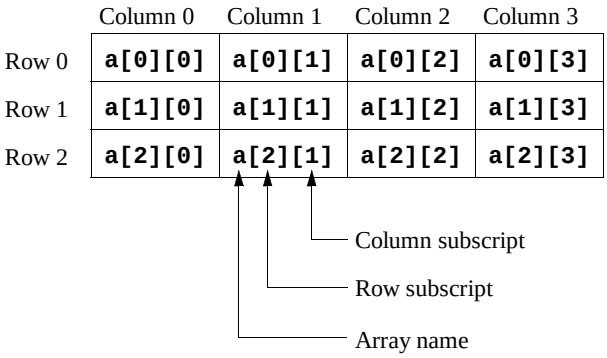

Fig. 4.21 A double-subscripted array with three rows and four columns.

```
1  // Fig. 4.22: fig04_22.cpp
2  // Initializing multidimensional arrays
3  #include <iostream.h>
4
5  void printArray( int [][ 3 ] );
6
7  int main()
8  {
9     int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } },
10        array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 },
11        array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };
12
13    cout << "Values in array1 by row are:" << endl;
14    printArray( array1 );
15
16    cout << "Values in array2 by row are:" << endl;
17    printArray( array2 );
18
19    cout << "Values in array3 by row are:" << endl;
20    printArray( array3 );
21
22    return 0;
23 }
24
```

Fig. 4.22 Initializing multidimensional arrays (part 1 of 2).

```
25 void printArray( int a[][ 3 ] )
26 {
27    for ( int i = 0; i < 2; i++ ) {
28
29       for ( int j = 0; j < 3; j++ )
30          cout << a[ i ][ j ] << ' ';
31
32       cout << endl;
33    }
34 }
```

```
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
```

Fig. 4.22 Initializing multidimensional arrays (part 2 of 2).

```
1  // Fig. 4.23: fig04_23.cpp
2  // Double-subscripted array example
3  #include <iostream.h>
4  #include <iomanip.h>
5
6  const int students = 3;   // number of students
7  const int exams = 4;      // number of exams
8
9  int minimum( int [][ exams ], int, int );
10 int maximum(int [][ exams ], int, int );
11 float average( int [], int );
12 void printArray( int [][ exams ], int, int );
13
```

Fig. 4.23 Example of using double-subscripted arrays (part 1 of 3).

```
14 int main()
15 {
16    int studentGrades[ students ][ exams ] =
17           { { 77, 68, 86, 73 },
18             { 96, 87, 89, 78 },
19             { 70, 90, 86, 81 } };
20
21    cout << "The array is:\n";
22    printArray( studentGrades, students, exams );
23    cout << "\n\nLowest grade: "
24         << minimum( studentGrades, students, exams )
25         << "\nHighest grade: "
26         << maximum( studentGrades, students, exams ) << '\n';
27
28    for ( int person = 0; person < students; person++ )
29       cout << "The average grade for student " << person << " is "
30            << setiosflags( ios::fixed | ios::showpoint )
31            << setprecision( 2 )
32            << average( studentGrades[ person ], exams ) << endl;
33
```

```
34       return 0;
35    }
36
37    // Find the minimum grade
38    int minimum( int grades[][ exams ], int pupils, int tests )
39    {
40       int lowGrade = 100;
41
42       for ( int i = 0; i < pupils; i++ )
43
44          for ( int j = 0; j < tests; j++ )
45
46             if ( grades[ i ][ j ] < lowGrade )
47                lowGrade = grades[ i ][ j ];
48
49       return lowGrade;
50    }
51
52    // Find the maximum grade
53    int maximum( int grades[][ exams ], int pupils, int tests )
54    {
55       int highGrade = 0;
56
57       for ( int i = 0; i < pupils; i++ )
58
59          for ( int j = 0; j < tests; j++ )
60
```

Fig. 4.23 Example of using double-subscripted arrays (part 2 of 3).

```
61             if ( grades[ i ][ j ] > highGrade )
62                highGrade = grades[ i ][ j ];
63
64       return highGrade;
65    }
66
67    // Determine the average grade for a particular student
68    float average( int setOfGrades[], int tests )
69    {
70       int total = 0;
71
72       for ( int i = 0; i < tests; i++ )
73          total += setOfGrades[ i ];
74
75       return ( float ) total / tests;
76    }
77
78    // Print the array
79    void printArray( int grades[][ exams ], int pupils, int tests )
80    {
81       cout << "                [0]  [1]  [2]  [3]";
82
83       for ( int i = 0; i < pupils; i++ ) {
84          cout << "\nstudentGrades[" << i << "] ";
85
86          for ( int j = 0; j < tests; j++ )
87             cout << setiosflags( ios::left ) << setw( 5 )
88                  << grades[ i ][ j ];
89       }
90    }
```

```
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
```

Fig. 4.23 Example of using double-subscripted arrays (part 3 of 3).