Many programs present data to users in a graphical manner.

One simple way to display numeric data graphically is with a bar chart that shows each numeric value as a bar of asterisks (*).

Our next program (Fig. 7.9) stores grade distribution data in an array of 11 elements, each corresponding to a category of grades, and displays a bar for each element.
7.4.5 Using Bar Charts to Display Array
Data Graphically

```cpp
// Fig. 7.9: fig07_09.cpp
// Bar chart printing program.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    const int arraySize = 11;
    int n[ arraySize ] = { 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 };

    cout << "Grade distribution:" << endl;

    // for each element of array n, output a bar of the chart
    for ( int i = 0; i < arraySize; i++ )
    {
        // output bar labels ("0-9:", ..., "90-99:", "100:" )
        if ( i == 0 )
            cout << " 0-9: ";
        else if ( i == 10 )
            cout << " 100: ";
        else
            cout << i * 10 << "-" << ( i * 10 ) + 9 << ": ";
    }
```

**Fig. 7.9** Bar chart printing program. (Part 1 of 2.)
7.4.5 Using Bar Charts to Display Array Data Graphically

```cpp
24     // print bar of asterisks
25     for ( int stars = 0; stars < n[ i ]; stars++ )
26         cout << '*';
27
28         cout << endl; // start a new line of output
29     } // end outer for
30 } // end main
```

Grade distribution:
- 0-9: 
- 10-19: 
- 20-29: 
- 30-39: 
- 40-49: 
- 50-59: 
- 60-69: * 
- 70-79: ** 
- 80-89: **** 
- 90-99: ** 
- 100: *

**Fig. 7.9** Bar chart printing program. (Part 2 of 2.)
Common Programming Error 7.6

Although it’s possible to use the same control variable in a for statement and in a second for statement nested inside, this is confusing and can lead to logic errors.
Sometimes, programs use counter variables to summarize data, such as the results of a survey.

In Fig. 6.9, we used separate counters in our die-rolling program to track the number of occurrences of each side of a die as the program rolled the die 6,000,000 times.

An array version of this program is shown in Fig. 7.10.

The single statement in line 18 of this program replaces the switch statement in lines 25–47 of Fig. 6.9.
7.4.6 Using the Elements of an Array as Counters

```cpp
// Fig. 7.10: fig07_10.cpp
// Roll a six-sided die 6,000,000 times.
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
    const int arraySize = 7; // ignore element zero
    int frequency[ arraySize ] = {}; // initialize elements to 0

    srand( time( 0 ) ); // seed random number generator

    // roll die 6,000,000 times; use die value as frequency index
    for ( int roll = 1; roll <= 6000000; roll++ )
        frequency[ 1 + rand() % 6 ]++;

    cout << "Face" << setw( 13 ) << "Frequency" << endl;
}
```

**Fig. 7.10** Die-rolling program using an array instead of switch. (Part 1 of 2.)
7.4.6 Using the Elements of an Array as Counters

```cpp
// output each array element's value
for (int face = 1; face < arraySize; face++)
    cout << setw(4) << face << setw(13) << frequency[face] << endl;
}
```

<table>
<thead>
<tr>
<th>Face</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000167</td>
</tr>
<tr>
<td>2</td>
<td>1000149</td>
</tr>
<tr>
<td>3</td>
<td>1000152</td>
</tr>
<tr>
<td>4</td>
<td>998748</td>
</tr>
<tr>
<td>5</td>
<td>999626</td>
</tr>
<tr>
<td>6</td>
<td>1001158</td>
</tr>
</tbody>
</table>

**Fig. 7.10** Die-rolling program using an array instead of `switch`. (Part 2 of 2.)
Our next example (Fig. 7.11) uses arrays to summarize the results of data collected in a survey.

Consider the following problem statement:

- Forty students were asked to rate the quality of the food in the student cafeteria on a scale of 1 to 10 (1 meaning awful and 10 meaning excellent). Place the 40 responses in an integer array and summarize the results of the poll.

C++ has no array bounds checking to prevent the computer from referring to an element that does not exist.

Thus, an executing program can “walk off” either end of an array without warning.

You should ensure that all array references remain within the bounds of the array.
7.4.7 Using Arrays to Summarize Survey Results

```cpp
// Fig. 7.11: fig07_11.cpp
// Poll analysis program.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    // define array sizes
    const int responseSize = 40; // size of array responses
    const int frequencySize = 11; // size of array frequency

    // place survey responses in array responses
    const int responses[ responseSize ] = { 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 };

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

    // for each answer, select responses element and use that value
    // as frequency subscript to determine element to increment
    for ( int answer = 0; answer < responseSize; answer++ )
        frequency[ responses[ answer ] ]++;
```

**Fig. 7.11** Poll analysis program. (Part 1 of 2.)
cout << "Rating" << setw( 17 ) << "Frequency" << endl;

// output each array element's value
for ( int rating = 1; rating < frequencySize; rating++ )
    cout << setw( 6 ) << rating << setw( 17 ) << frequency[ rating ]
    << endl;

<table>
<thead>
<tr>
<th>Rating</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

Fig. 7.11  | Poll analysis program. (Part 2 of 2.)
Common Programming Error 7.7

Referring to an element outside the array bounds is an execution-time logic error. It isn’t a syntax error.
Error-Prevention Tip 7.1
When looping through an array, the index should never go below 0 and should always be less than the total number of array elements (one less than the size of the array). Make sure that the loop-termination condition prevents accessing elements outside this range.
Portability Tip 7.1

The (normally serious) effects of referencing elements outside the array bounds are system dependent. Often this results in changes to the value of an unrelated variable or a fatal error that terminates program execution.
7.4.8 Static Local Arrays and Automatic Local Arrays

- A program initializes static local arrays when their declarations are first encountered.
- If a static array is not initialized explicitly by you, each element of that array is initialized to zero by the compiler when the array is created.
Performance Tip 7.1

We can apply static to a local array declaration so that it is not created and initialized each time the program calls the function and is not destroyed each time the function terminates. This can improve performance, especially when using large arrays.
// Fig. 7.12: fig07_12.cpp
// Static arrays are initialized to zero.
#include <iostream>
using namespace std;

void staticArrayInit( void ); // function prototype
void automaticArrayInit( void ); // function prototype
const int arraySize = 3;

int main()
{
    cout << "First call to each function:\n";
    staticArrayInit();
    automaticArrayInit();

    cout << "\n\nSecond call to each function:\n";
    staticArrayInit();
    automaticArrayInit();
    cout << endl;
} // end main

Fig. 7.12  static array initialization and automatic array initialization. (Part 1 of 4.)
// function to demonstrate a static local array
void staticArrayInit( void )
{
    // initializes elements to 0 first time function is called
    static int array1[ arraySize ]; // static local array

    cout << "\nValues on entering staticArrayInit:\n";

    // output contents of array1
    for ( int i = 0; i < arraySize; i++ )
    {
        cout << "array1[" << i << "] = " << array1[ i ] << " \n";
    }

    cout << "\nValues on exiting staticArrayInit:\n";

    // modify and output contents of array1
    for ( int j = 0; j < arraySize; j++ )
    {
        cout << "array1[" << j << "] = " << ( array1[ j ] += 5 ) << " \n";
    }
} // end function staticArrayInit

**Fig. 7.12**  |  static array initialization and automatic array initialization. (Part 2 of 4.)
7.4.8 Static Local Arrays and Automatic Local Arrays

```c++
41  // function to demonstrate an automatic local array
42  void automaticArrayInit( void )
43  {
44  // initializes elements each time function is called
45  int array2[ arraySize ] = { 1, 2, 3 }; // automatic local array
46
47  cout << "\n\nValues on entering automaticArrayInit:\n";
48
49  // output contents of array2
50  for ( int i = 0; i < arraySize; i++ )
51    cout << "array2[" << i << "] = " << array2[ i ] << " \n";
52
53  cout << "\nValues on exiting automaticArrayInit:\n";
54
55  // modify and output contents of array2
56  for ( int j = 0; j < arraySize; j++ )
57    cout << "array2[" << j << "] = " << ( array2[ j ] += 5 ) << " \n";
58  } // end function automaticArrayInit
```

**Fig. 7.12** | static array initialization and automatic array initialization. (Part 3 of 4.)
7.4.8 Static Local Arrays and Automatic Local Arrays

First call to each function:

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

Values on entering automaticArrayInit:
Values on exiting automaticArrayInit:

Second call to each function:

Values on entering staticArrayInit:
Values on exiting staticArrayInit:

Values on entering automaticArrayInit:
Values on exiting automaticArrayInit:

Fig. 7.12  |  static array initialization and automatic array initialization. (Part 4 of 4.)
To pass an array argument to a function, specify the name of the array without any brackets.

When passing an array to a function, the array size is normally passed as well, so the function can process the specific number of elements in the array.
- Otherwise, we would need to build this knowledge into the called function itself or, worse yet, place the array size in a global variable.

C++ passes arrays to functions by reference—the called functions can modify the element values in the callers’ original arrays.

The value of the name of the array is the address in the computer’s memory of the first element of the array.
- Because the starting address of the array is passed, the called function knows precisely where the array is stored in memory.
Performance Tip 7.2

Passing arrays by reference makes sense for performance reasons. Passing by value would require copying each element. For large, frequently passed arrays, this would be time consuming and would require considerable storage for the copies of the array elements.

Software Engineering Observation 7.3

It’s possible to pass an array by value (by using a simple trick we explain in Chapter 21)—however, this is rarely done.
• Although entire arrays are passed by reference, individual array elements are passed by value exactly as simple variables are.

• Such simple single pieces of data are called **scalars** or **scalar quantities**.

• To pass an element of an array to a function, use the subscripted name of the array element as an argument in the function call.
7.5 Passing Arrays to Functions (cont.)

```cpp
// Fig. 7.13: fig07_13.cpp
// Passing arrays and individual array elements to functions.
#include <iostream>
#include <iomanip>
using namespace std;

void modifyArray( int [], int ); // appears strange; array and size
void modifyElement( int ); // receive array element value

int main()
{
    const int arraySize = 5; // size of array a
    int a[ arraySize ] = { 0, 1, 2, 3, 4 }; // initialize array a

    cout << "Effects of passing entire array by reference:";
    cout << "\n\nThe values of the original array are:\n";

    // output original array elements
    for ( int i = 0; i < arraySize; i++ )
        cout << setw( 3 ) << a[ i ];

    cout << endl;
}
```

**Fig. 7.13** | Passing arrays and individual array elements to functions. (Part 1 of 3.)
7.5 Passing Arrays to Functions (cont.)

```cpp
// pass array a to modifyArray by reference
modifyArray( a, arraySize );
cout << "The values of the modified array are:\n";

// output modified array elements
for ( int j = 0; j < arraySize; j++ )
    cout << setw( 3 ) << a[ j ];

cout << "\n\n\nEffects of passing array element by value:

} // end main

// in function modifyArray, "b" points to the original array "a" in memory
void modifyArray( int b[], int sizeOfArray )
{
    // multiply each array element by 2
    for ( int k = 0; k < sizeOfArray; k++ )
        b[ k ] *= 2;
} // end function modifyArray
```

Fig. 7.13 | Passing arrays and individual array elements to functions. (Part 2 of 3.)
7.5 Passing Arrays to Functions (cont.)

```c++
47    // in function modifyElement, "e" is a local copy of
48    // array element a[ 3 ] passed from main
49    void modifyElement( int e )
50    {
51       // multiply parameter by 2
52       cout << "Value of element in modifyElement: " << ( e *= 2 ) << endl;
53    } // end function modifyElement
```

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:

Value of element in modifyElement: 12

**Fig. 7.13** Passing arrays and individual array elements to functions. (Part 3 of 3.)
7.5 Passing Arrays to Functions (cont.)

// Fig. 7.14: fig07_14.cpp
// Demonstrating the const type qualifier.
#include <iostream>
using namespace std;

void tryToModifyArray( const int [] ); // function prototype

int main()
{
    int a[] = { 10, 20, 30 };
    tryToModifyArray( a );
    cout << a[ 0 ] << ' ' << a[ 1 ] << ' ' << a[ 2 ] << '\n';
} // end main

// In function tryToModifyArray, "b" cannot be used
// to modify the original array "a" in main.
void tryToModifyArray( const int b[] )
{
    b[ 0 ] /= 2; // compilation error
    b[ 1 ] /= 2; // compilation error
    b[ 2 ] /= 2; // compilation error
} // end function tryToModifyArray

Fig. 7.14 | const type qualifier applied to an array parameter. (Part 1 of 2.)
7.5 Passing Arrays to Functions (cont.)

*Microsoft Visual C++ compiler error message:*

c:\cpphtp7_examples\ch07\fig07_14\fig07_14.cpp(20) : error C3892: 'b' : you cannot assign to a variable that is const
c:\cpphtp7_examples\ch07\fig07_14\fig07_14.cpp(21) : error C3892: 'b' : you cannot assign to a variable that is const
c:\cpphtp7_examples\ch07\fig07_14\fig07_14.cpp(22) : error C3892: 'b' : you cannot assign to a variable that is const

*GNU C++ compiler error message:*

fig07_14.cpp:20: error: assignment of read-only location
fig07_14.cpp:21: error: assignment of read-only location
fig07_14.cpp:22: error: assignment of read-only location

**Fig. 7.14** \ const\ type qualifier applied to an array parameter. (Part 2 of 2.)