Lecture 14:
Arrays

Ioan Raicu
Department of Electrical Engineering & Computer Science
Northwestern University

EECS 211
Fundamentals of Computer Programming II
April 20th, 2010

7.4.5 Using Bar Charts to Display Array Data Graphically

- 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

```
1  // Fig. 7.9: fig07_09.cpp
2  // Bar chart printing program.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9     const int arraySize = 11;
10    int n[ arraySize ] = { 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 };
11
12    cout << "Grade distribution:" << endl;
13
14    // for each element of array n, output a bar of the chart
15    for ( int i = 0; i < arraySize; i++ )
16    {
17       // output bar labels ("0-9:", ..., "90-99:", "100:" )
18       if ( i == 0 )
19          cout << "  0-9: ";
20       else if ( i == 10 )
21          cout << "  100: ";
22       else
23          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

```
24
25          // print bar of asterisks
26          for ( int stars = 0; stars < n[ i ]; stars++ )
27             cout << '*';
28
29          cout << endl; // start a new line of output
30       } // end outer for
31    } // 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.)

7.4.5 Using Bar Charts to Display Array Data Graphically

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.*

7.4.6 Using the Elements of an Array as Counters

- 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

```
1  // Fig. 7.10: fig07_10.cpp
2  // Roll a six-sided die 6,000,000 times.
3  #include <iostream>
4  #include <iomanip>
5  #include <cstdlib>
6  #include <ctime>
7  using namespace std;
8
9  int main()
10 {
11    const int arraySize = 7; // ignore element zero
12    int frequency[ arraySize ] = {}; // initialize elements to 0
13
14    srand( time( 0 ) ); // seed random number generator
15
16    // roll die 6,000,000 times; use die value as frequency index
17    for ( int roll = 1; roll <= 6000000; roll++ )
18       frequency[ 1 + rand() % 6 ]++;
19
20    cout << "Face" << setw( 13 ) << "Frequency" << endl;
21
```

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

```
22      // output each array element's value
23      for ( int face = 1; face < arraySize; face++ )
24         cout << setw( 4 ) << face << setw( 13 ) << frequency[ face ]
25            << endl;
26   } // end main
```

```
Face    Frequency
   1      1000167
   2      1000149
   3      1000152
   4       998748
   5       999626
   6      1001158
```

Fig. 7.10 | Die-rolling program using an array instead of `switch`. (Part 2 of 2.)

7.4.7 Using Arrays to Summarize Survey Results

- 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

```
1  // Fig. 7.11: fig07_11.cpp
2  // Poll analysis program.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9     // define array sizes
10    const int responseSize = 40; // size of array responses
11    const int frequencySize = 11; // size of array frequency
12
13    // place survey responses in array responses
14    const int responses[ responseSize ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8,
15       10, 1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7,
16       5, 6, 6, 5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
17
18    // initialize frequency counters to 0
19    int frequency[ frequencySize ] = {};
20
21    // for each answer, select responses element and use that value
22    // as frequency subscript to determine element to increment
23    for ( int answer = 0; answer < responseSize; answer++ )
24       frequency[ responses[ answer ] ]++;
```

Fig. 7.11 | Poll analysis program. (Part 1 of 2.)

7.4.7 Using Arrays to Summarize Survey Results

```
25
26      cout << "Rating" << setw( 17 ) << "Frequency" << endl;
27
28      // output each array element's value
29      for ( int rating = 1; rating < frequencySize; rating++ )
30         cout << setw( 6 ) << rating << setw( 17 ) << frequency[ rating ]
31            << endl;
32   } // end main
```

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

Fig. 7.11 | Poll analysis program. (Part 2 of 2.)

7.4.7 Using Arrays to Summarize Survey Results

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.

7.4.7 Using Arrays to Summarize Survey Results

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.

7.4.7 Using Arrays to Summarize Survey Results

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.

7.4.8 Static Local Arrays and Automatic Local Arrays

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.*

7.4.8 Static Local Arrays and Automatic Local Arrays

```
1  // Fig. 7.12: fig07_12.cpp
2  // Static arrays are initialized to zero.
3  #include <iostream>
4  using namespace std;
5
6  void staticArrayInit( void ); // function prototype
7  void automaticArrayInit( void ); // function prototype
8  const int arraySize = 3;
9
10 int main()
11 {
12    cout << "First call to each function:\n";
13    staticArrayInit();
14    automaticArrayInit();
15
16    cout << "\n\nSecond call to each function:\n";
17    staticArrayInit();
18    automaticArrayInit();
19    cout << endl;
20 } // end main
21
```

Fig. 7.12 | `static` array initialization and automatic array initialization. (Part 1 of 4.)

7.4.8 Static Local Arrays and Automatic Local Arrays

```
22  // function to demonstrate a static local array
23  void staticArrayInit( void )
24  {
25     // initializes elements to 0 first time function is called
26     static int array1[ arraySize ]; // static local array
27
28     cout << "\nValues on entering staticArrayInit:\n";
29
30     // output contents of array1
31     for ( int i = 0; i < arraySize; i++ )
32        cout << "array1[" << i << "] = " << array1[ i ] << "   ";
33
34     cout << "\nValues on exiting staticArrayInit:\n";
35
36     // modify and output contents of array1
37     for ( int j = 0; j < arraySize; j++ )
38        cout << "array1[" << j << "] = " << ( array1[ j ] += 5 ) << "   ";
39  } // end function staticArrayInit
40
```

Fig. 7.12 | `static` array initialization and automatic array initialization. (Part 2 of 4.)

7.4.8 Static Local Arrays and Automatic Local Arrays

```
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 ] << "   ";
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 ) << "   ";
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:
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. 7.12 | `static` array initialization and automatic array initialization. (Part 4 of 4.)

7.5 Passing Arrays to Functions

- To pass an array argument to a function, specify the name of the array without any brack-ets.
- 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 start-ing address of the array is passed, the called function knows precisely where the array is stored in memory.

7.5 Passing Arrays to Functions

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.

7.5 Passing Arrays to Functions (cont.)

- 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 func-tion, use the subscripted name of the array element as an argument in the function call.

7.5 Passing Arrays to Functions (cont.)

```
1  // Fig. 7.13: fig07_13.cpp
2  // Passing arrays and individual array elements to functions.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  void modifyArray( int [], int ); // appears strange; array and size
8  void modifyElement( int ); // receive array element value
9
10 int main()
11 {
12    const int arraySize = 5; // size of array a
13    int a[ arraySize ] = { 0, 1, 2, 3, 4 }; // initialize array a
14
15    cout << "Effects of passing entire array by reference:"
16       << "\n\nThe values of the original array are:\n";
17
18    // output original array elements
19    for ( int i = 0; i < arraySize; i++ )
20       cout << setw( 3 ) << a[ i ];
21
22    cout << endl;
23
```

Fig. 7.13 | Passing arrays and individual array elements to functions. (Part 1 of 3.)

7.5 Passing Arrays to Functions (cont.)

```
24       // pass array a to modifyArray by reference
25       modifyArray( a, arraySize );
26       cout << "The values of the modified array are:\n";
27
28       // output modified array elements
29       for ( int j = 0; j < arraySize; j++ )
30          cout << setw( 3 ) << a[ j ];
31
32       cout << "\n\n\nEffects of passing array element by value:"
33          << "\n\na[3] before modifyElement: " << a[ 3 ] << endl;
34
35       modifyElement( a[ 3 ] ); // pass array element a[ 3 ] by value
36       cout << "a[3] after modifyElement: " << a[ 3 ] << endl;
37    } // end main
38
39    // in function modifyArray, "b" points to the original array "a" in memory
40    void modifyArray( int b[], int sizeOfArray )
41    {
42       // multiply each array element by 2
43       for ( int k = 0; k < sizeOfArray; k++ )
44          b[ k ] *= 2;
45    } // end function modifyArray
46
```

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

7.5 Passing Arrays to Functions (cont.)

```
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:

a[3] before modifyElement: 6
Value of element in modifyElement: 12
a[3] after modifyElement: 6
```

Fig. 7.13 | Passing arrays and individual array elements to functions. (Part 3 of 3.)

7.5 Passing Arrays to Functions (cont.)

```
1  // Fig. 7.14: fig07_14.cpp
2  // Demonstrating the const type qualifier.
3  #include <iostream>
4  using namespace std;
5
6  void tryToModifyArray( const int [] ); // function prototype
7
8  int main()
9  {
10    int a[] = { 10, 20, 30 };
11
12    tryToModifyArray( a );
13    cout << a[ 0 ] << ' ' << a[ 1 ] << ' ' << a[ 2 ] << '\n';
14 } // end main
15
16 // In function tryToModifyArray, "b" cannot be used
17 // to modify the original array "a" in main.
18 void tryToModifyArray( const int b[] )
19 {
20    b[ 0 ] /= 2; // compilation error
21    b[ 1 ] /= 2; // compilation error
22    b[ 2 ] /= 2; // compilation error
23 } // 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.)

Questions