Class Templates `array` and `vector`; Catching Exceptions

7

Objectives

In this chapter you'll:

- Use C++ Standard Library class template `array`—a fixed-size collection of related data items.
- Declare `array`s, initialize `array`s and refer to the elements of `array`s.
- Use `array`s to store, sort and search lists and tables of values.
- Use the range-based `for` statement.
- Pass `array`s to functions.
- Use C++ Standard Library function `sort` to arrange `array` elements in ascending order.
- Use C++ Standard Library function `binary_search` to locate an element in a sorted `array`.
- Declare and manipulate multidimensional `array`s.
- Use one- and two-dimensional `array`s to build a real-world `GradeBook` class.
- Use C++ Standard Library class template `vector`—a variable-size collection of related data items.

Self-Review Exercises

7.1 *(Fill in the Blanks)* Answer each of the following:

a) Lists and tables of values can be stored in _____ or _____.

ANS: arrays, vectors.

b) An array's elements are related by the fact that they have the same _____ and _____.

ANS: array name, type.

c) The number used to refer to a particular element of an array is called its _____.

ANS: subscript or index.

d) A(n) _____ should be used to declare the size of an array, because it eliminates magic numbers.

ANS: constant variable.

e) The process of placing the elements of an array in order is called _____ the array.

ANS: sorting.

f) The process of determining if an array contains a particular key value is called _____ the array.

ANS: searching.

g) An array that uses two subscripts is referred to as a(n) _____ array.

ANS: two-dimensional.

7.2 *(True or False)* State whether the following are *true* or *false*. If the answer is *false*, explain why.

a) A given array can store many different types of values.

ANS: False. An array can store only values of the same type.

b) An array subscript should normally be of data type float.

ANS: False. An array subscript should be an integer or an integer expression.

c) If there are fewer initializers in an initializer list than the number of elements in the array, the remaining elements are initialized to the last value in the initializer list.

ANS: False. The remaining elements are initialized to zero.

d) It's an error if an initializer list has more initializers than there are elements in the array.

ANS: True.

7.3 *(Write C++ Statements)* Write one or more statements that perform the following tasks for an array called fractions:

a) Define a constant variable arraySize to represent the size of an array and initialize it to 10.

ANS:
```
const size_t arraySize{10};
```

b) Declare an array with arraySize elements of type double, and initialize the elements to 0.

ANS:
```
array<double, arraySize> fractions{0.0};
```

c) Name the fourth element of the array.

ANS:
```
fractions[3]
```

d) Refer to array element 4.

ANS:
```
fractions[4]
```

e) Assign the value 1.667 to array element 9.

ANS:
```
fractions[9] = 1.667;
```

f) Assign the value 3.333 to the seventh element of the array.

ANS:
```
fractions[6] = 3.333;
```

g) Display array elements 6 and 9 with two digits of precision to the right of the decimal point, and show the output that's actually displayed on the screen.

ANS:
```
cout << fixed << setprecision(2);
cout << fractions[6] << ' ' << fractions[9] << endl;
```
Output: `3.33 1.67`

h) Display all the array elements using a counter-controlled for statement. Define the integer variable i as a control variable for the loop. Show the output.

ANS:
```
for (size_t i{0}; i < fractions.size(); ++i) {
   cout << "fractions[" << i << "] = " << fractions[i] << endl;
}
```
Output:
```
fractions[0] = 0.0
fractions[1] = 0.0
fractions[2] = 0.0
fractions[3] = 0.0
fractions[4] = 0.0
fractions[5] = 0.0
fractions[6] = 3.333
fractions[7] = 0.0
fractions[8] = 0.0
fractions[9] = 1.667
```

i) Display all the array elements separated by spaces using a range-based for statement.

ANS:
```
for (double element : fractions)
   cout << element << ' ';
```

7.4 *(Two-Dimensional array Questions)* Answer the following questions regarding a two-dimensional array called table:

a) Declare the array to store int values and to have 3 rows and 3 columns. Assume that the constant variable arraySize has been defined to be 3.

ANS:
```
array<array<int, arraySize>, arraySize> table;
```

b) How many elements does the array contain?

ANS: Nine.

c) Use a counter-controlled for statement to initialize each element of the array to the sum of its subscripts.

ANS:
```
for (size_t row{0}; row < table.size(); ++row) {
   for (size_t column{0}; column < table[row].size(); ++column) {
      table[row][column] = row + column;
   }
}
```

d) Write a nested for statement that displays the values of each element of array table in tabular format with 3 rows and 3 columns. Each row and column should be labeled with the row or column number. Assume that the array was initialized with an initializer list containing the values from 1 through 9 in order. Show the output.

ANS:
```
cout << "     [0]  [1]  [2]" << endl;
for (size_t i{0}; i < arraySize; ++i) {
   cout << '[' << i << "] ";
   for (size_t j{0}; j < arraySize; ++j) {
      cout << setw(3) << table[i][j] << "  ";
   }
   cout << endl;
}
```

Output:

```
     [0]  [1]  [2]
[0]   1    2    3
[1]   4    5    6
[2]   7    8    9
```

7.5 ***(Find the Error)*** Find and correct the error in each of the following program segments:

a) `#include <iostream>;`

ANS: *Error:* Semicolon at end of `#include` preprocessing directive.
Correction: Eliminate semicolon.

b) `arraySize = 10; // arraySize was declared const`

ANS: *Error:* Assigning a value to a constant variable using an assignment statement.
Correction: Initialize the constant variable in a `const size_t arraySize` declaration.

c) Assume that `array<int, 10> b{};`

```
for (size_t i{0}; i <= b.size(); ++i) {
   b[i] = 1;
}
```

ANS: *Error:* Referencing an `array` element outside the bounds of the `array` (`b[10]`).
Correction: Change the loop-continuation condition to use < rather than <=.

d) Assume that a is a two-dimensional `array` of `int` values with two rows and two columns:

`a[1, 1] = 5;`

ANS: *Error:* `array` subscripting done incorrectly.
Correction: Change the statement to `a[1][1] = 5;`

Exercises

7.6 ***(Fill in the Blanks)*** Fill in the blanks in each of the following:

a) The names of the four elements of `array p` are _____, _____, _____ and _____.

ANS: `p[0]`, `p[1]`, `p[2]`, `p[3]`

b) Naming an `array`, stating its type and specifying the number of elements in the `array` is called _____ the `array`.

ANS: declaring.

c) When accessing an `array` element, by convention, the first subscript in a two-dimensional `array` identifies an element's _____ and the second subscript identifies an element's _____.

ANS: row, column.

d) An *m*-by-*n* `array` contains _____ rows, _____ columns and _____ elements.

ANS: *m*, *n*, $m \times n$.

e) The name of the element in row 3 and column 5 of `array d` is _____.

ANS: `d[3][5]`.

7.7 ***(True or False)*** Determine whether each of the following is *true* or *false*. If *false*, explain why.

a) To refer to a particular location or element within an `array`, we specify the name of the `array` and the value of the particular element.

ANS: False. The name of the array and the subscript of the array are specified.

b) An `array` definition reserves space for an `array`.

ANS: True.

c) To reserve 100 locations for integer `array` `p`, you write

```
p[100];
```

ANS: False. A data type must be specified. An example of a correct definition would be:

```
array<int, 100> p;
```

d) A `for` statement must be used to initialize the elements of a 15-element `array` to zero.

ANS: False. The array can be initialized in a declaration with a member initializer list.

e) Nested `for` statements must be used to total the elements of a two-dimensional `array`.

ANS: False. The sum of the elements can be obtained without `for` statements, with one `for` statement, three `for` statements, etc.

7.8 ***(Write C++ Statements)*** Write C++ statements to accomplish each of the following:

a) Display the value of element 6 of character `array` `alphabet`.

ANS:
```
cout << alphabet[6] << '\n';
```

b) Input a value into element 4 of one-dimensional floating-point `array` `grades`.

ANS:
```
cin >> grades[4];
```

c) Initialize each of the 5 elements of one-dimensional integer `array` `values` to `8`.

ANS:
```
array<int, 5> values{8, 8, 8, 8, 8};
```

d) Total and display the elements of floating-point `array` `temperatures` of 100 elements.

ANS:
```
double total{0.0};
for (int k{0}; k < 100; ++k) {
   total += temperatures[k];
   cout << temperatures[k] << '\n';
}
```

or

```
double total{0.0};
for (int temp : temperatures) {
   total += temp;
   cout << temp << '\n';
}
```

e) Copy `array` `a` into the first portion of `array` `b`. Assume that both `arrays` contain `doubles` and that `arrays` `a` and `b` have 11 and 34 elements, respectively.

ANS:
```
for (int i{0}; i < 11; ++i) {
   b[i] = a[i];
}
```

f) Determine and display the smallest and largest values contained in 99-element floating-point `array` `w`.

ANS:
```
double smallest{w[0]};
double largest{w[0]};

for (int j{1}; j < 99; ++j) {
   if (w[j] < smallest) {
      smallest = w[j];
   }
   else if (w[j] > largest) {
       largest = w[j];
   }
}

cout << smallest << ' ' << largest;
```

or you could write the loop as

```
for (double element : w) {
   if (element < smallest) {
      smallest = element;
   }
   else if (element > largest) {
      largest = element;
   }
} // end for
```

7.9 ***(Two-Dimensional array Questions)*** Consider a 2-by-3 integer `array` `t`.

a) Write a declaration for `t`.

ANS: `array<array<int, 3>, 2> t;`

b) How many rows does `t` have?

ANS: 2

c) How many columns does `t` have?

ANS: 3

d) How many elements does `t` have?

ANS: 6

e) Write the names of all the elements in row 1 of `t`.

ANS: `t[1][0], t[1][1], t[1][2]`

f) Write the names of all the elements in column 2 of `t`.

ANS: `t[0][2], t[1][2]`

g) Write a statement that sets the element of `t` in the first row and second column to zero.

ANS: `t[0][1] = 0;`

h) Write a series of statements that initialize each element of `t` to zero. Do not use a loop.

ANS:

```
t[0][0] = 0;
t[0][1] = 0;
t[0][2] = 0;
t[1][0] = 0;
t[1][1] = 0;
t[1][2] = 0;
```

i) Write a nested counter-controlled `for` statement that initializes each element of `t` to zero.

ANS:

```
for (int i{0}; i < 2; ++i) {
   for (int j{0}; j < 3; ++j) {
      t[i][j] = 0;
    }
}
```

j) Write a nested range-based `for` statement that initializes each element of `t` to zero.

ANS:

```
for (auto& row : t) {
   for (auto& element : row) {
      element = 0;
   }
}
```

k) Write a statement that inputs the values for the elements of `t` from the keyboard.

ANS:

```
for (int i{0}; i < 2; ++i) {
   for (int j{0}; j < 3; ++j) {
      cin >> t[i][j];
   }
}
```

l) Write a series of statements that determine and display the smallest value in `array t`.

ANS:

```
int smallest{t[0][0]};

for (int i{0}; i < 2; ++i) {
   for (int j{0}; j < 3; ++j) {
      if (t[i][j] < smallest) {
         smallest = t[i][j];
      }
   }
}

cout << smallest;
```

m) Write a statement that displays the elements in row 0 of `t`.

ANS: `cout << t[0][0] << ' ' << t[0][1] << ' ' << t[0][2] << '\n';`

n) Write a statement that totals the elements in column 2 of `t`.

ANS: `int total{t[0][2] + t[1][2]};`

o) Write a series of statements that prints the `array t` in neat, tabular format. List the column subscripts as headings across the top and list the row subscripts at the left of each row.

ANS:

```
cout << "  0     1     2\n";

for (int r{0}; r < 2; ++r) {
   cout << r << ' ';

   for (int c{0}; c < 3; ++c) {
      cout << t[r][c] << "    ";
   }

   cout << '\n';
}
```

7.11 ***(One-Dimensional array Questions)*** Write statements that perform the following one-dimensional `array` operations:

a) Initialize the 10 elements of integer `array counts` to zero.

ANS: `array<int, 10> t{};`

b) Add 1 to each of the 15 elements of integer `array bonus`.

ANS:

```
for (int i{0}; i < 15; ++i) {
   ++bonus[i];
}
```

or

```
for (int& element : bonus) {
   ++element;
}
```

c) Read 12 values for the array of doubles named monthlyTemperatures from the keyboard.

ANS:

```
for (int p{0}; p < 12; ++p) {
   cin >> monthlyTemperatures[p];
}
```

or

```
for (int& element : monthlyTemperatures) {
   cin >> element;
}
```

d) Print the 5 values of integer array bestScores in column format.

ANS:

```
for (int u{0}; u < 5; ++u) {
   cout << bestScores[u] << '\t';
}
```

or

```
for (int element : bestScores) {
   cout << element << '\t';
}
```

7.12 ***(Find the Errors)*** Find the error(s) in each of the following statements:

a) Assume that a is an array of three ints.

```
cout << a[1] << " " << a[2] << " " << a[3] << endl;
```

ANS: a[3] is not a valid location in the array. a[2] is the last valid location.

b) `array<double, 3> f{1.1, 10.01, 100.001, 1000.0001};`

ANS: Too many initializers in the initializer list. Only 1, 2, or 3 values may be provided in the initializer list.

c) Assume that d is an array of doubles with two rows and 10 columns.

```
d[1, 9] = 2.345;
```

ANS: Incorrect syntax array element access. d[1][9] is the correct syntax.

7.15 ***(Two-Dimensional array Initialization)*** Label the elements of a 3-by-5 two-dimensional array sales to indicate the order in which they're set to zero by the following program segment:

```
for (size_t row{0}; row < sales.size(); ++row) {
   for (size_t column{0}; column < sales[row].size(); ++column) {
      sales[row][column] = 0;
   }
}
```

ANS: sales[0][0], sales[0][1], sales[0][2], sales[0][3], sales[0][4], sales[1][0], sales[1][1], sales[1][2], sales[1][3], sales[1][4], sales[2][0], sales[2][1], sales[2][2], sales[2][3], sales[2][4]

7.17 *(What Does This Code Do?)* What does the following program do?

```
// Ex. 7.17: Ex07_17.cpp
// What does this program do?
#include <iostream>
#include <array>
using namespace std;

const size_t arraySize{10};
int whatIsThis(const array<int, arraySize>&, size_t); // prototype

int main() {
   array<int, arraySize> a{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

   int result{whatIsThis(a, arraySize)};

   cout << "Result is " << result << endl;
}

// What does this function do?
int whatIsThis(const array<int, arraySize>& b, size_t size) {
   if (size == 1) { // base case
      return b[0];
   }
   else { // recursive step
      return b[size - 1] + whatIsThis(b, size - 1);
   }
}
```

Fig. 7.23 | What does this program do?

ANS: This program sums array elements. The output would be

```
Result is 55
```

7.20 *(What Does This Code Do?)* What does the following program do?

```
1  // Ex. 7.20: Ex07_20.cpp
2  // What does this program do?
3  #include <iostream>
4  #include <array>
5  using namespace std;
6
7  const size_t arraySize{10};
8  void someFunction(const array<int, arraySize>&, size_t); // prototype
9
10 int main() {
11    array<int, arraySize> a{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
12
13    cout << "The values in the array are:" << endl;
14    someFunction(a, 0);
15    cout << endl;
16 }
17
18 // What does this function do?
19 void someFunction(const array<int, arraySize>& b, size_t current) {
20    if (current < b.size()) {
21       someFunction(b, current + 1);
22       cout << b[current] << "  ";
23    }
24 }
```

Fig. 7.24 | What does this program do?

ANS: This program prints array elements in reverse order. The output would be

```
The values in the array are:
10  9  8  7  6  5  4  3  2  1
```