10.10 Case Study: `Array` Class (cont.)

- With C++, you can implement more robust array capabilities via classes and operator overloading as has been done with class templates `array` and `vector` in the C++ Standard Library.
- In this section, we'll develop our own custom array class that's preferable to built-in arrays.
- In this example, we create a powerful `Array` class:
  - Performs range checking.
  - Allows one `Array` object to be assigned to another with the assignment operator.
  - Objects know their own size.
  - Input or output entire arrays with the stream extraction and stream insertion operators, respectively.
  - Can compare `Array`s with the equality operators == and `!=`.
- C++ Standard Library class template `vector` provides many of these capabilities as well.

```
1  // Fig. 10.9: fig10_09.cpp
2  // Array class test program.
3  #include <iostream>
4  #include <stdexcept>
5  #include "Array.h"
6  using namespace std;
7
8  int main()
9  {
10    Array integers1( 7 ); // seven-element Array
11    Array integers2; // 10-element Array by default
12
13    // print integers1 size and contents
14    cout << "Size of Array integers1 is "
15       << integers1.getSize()
16       << "\nArray after initialization:\n" << integers1;
17
18    // print integers2 size and contents
19    cout << "\nSize of Array integers2 is "
20       << integers2.getSize()
21       << "\nArray after initialization:\n" << integers2;
22
23    // input and print integers1 and integers2
24    cout << "\nEnter 17 integers:" << endl;
25    cin >> integers1 >> integers2;
```

Fig. 10.9 | Array class test program. (Part 1 of 7.)

```
26
27      cout << "\nAfter input, the Arrays contain:\n"
28         << "integers1:\n" << integers1
29         << "integers2:\n" << integers2;
30
31      // use overloaded inequality (!=) operator
32      cout << "\nEvaluating: integers1 != integers2" << endl;
33
34      if ( integers1 != integers2 )
35         cout << "integers1 and integers2 are not equal" << endl;
36
37      // create Array integers3 using integers1 as an
38      // initializer; print size and contents
39      Array integers3( integers1 ); // invokes copy constructor
40
41      cout << "\nSize of Array integers3 is "
42         << integers3.getSize()
43         << "\nArray after initialization:\n" << integers3;
44
45      // use overloaded assignment (=) operator
46      cout << "\nAssigning integers2 to integers1:" << endl;
47      integers1 = integers2; // note target Array is smaller
48
```

Fig. 10.9 | Array class test program. (Part 2 of 7.)

```
49      cout << "integers1:\n" << integers1
50         << "integers2:\n" << integers2;
51
52      // use overloaded equality (==) operator
53      cout << "\nEvaluating: integers1 == integers2" << endl;
54
55      if ( integers1 == integers2 )
56         cout << "integers1 and integers2 are equal" << endl;
57
58      // use overloaded subscript operator to create rvalue
59      cout << "\nintegers1[5] is " << integers1[ 5 ];
60
61      // use overloaded subscript operator to create lvalue
62      cout << "\n\nAssigning 1000 to integers1[5]" << endl;
63      integers1[ 5 ] = 1000;
64      cout << "integers1:\n" << integers1;
65
```

Fig. 10.9 | Array class test program. (Part 3 of 7.)

```
66      // attempt to use out-of-range subscript
67      try
68      {
69         cout << "\nAttempt to assign 1000 to integers1[15]" << endl;
70         integers1[ 15 ] = 1000; // ERROR: subscript out of range
71      } // end try
72      catch ( out_of_range &ex )
73      {
74         cout << "An exception occurred: " << ex.what() << endl;
75      } // end catch
76   } // end main
```

Fig. 10.9 | `Array` class test program. (Part 4 of 7.)

```
Size of Array integers1 is 7
Array after initialization:
          0          0          0          0
          0          0          0

Size of Array integers2 is 10
Array after initialization:
          0          0          0          0
          0          0          0          0
          0          0

Enter 17 integers:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

After input, the Arrays contain:
integers1:
          1          2          3          4
          5          6          7
integers2:
          8          9         10         11
         12         13         14         15
         16         17
```

Fig. 10.9 | `Array` class test program. (Part 5 of 7.)

```
Evaluating: integers1 != integers2
integers1 and integers2 are not equal

Size of Array integers3 is 7
Array after initialization:
           1           2           3           4
           5           6           7

Assigning integers2 to integers1:
integers1:
           8           9          10          11
          12          13          14          15
          16          17

integers2:
           8           9          10          11
          12          13          14          15
          16          17
```

Fig. 10.9 | `Array` class test program. (Part 6 of 7.)

```
Evaluating: integers1 == integers2
integers1 and integers2 are equal

integers1[5] is 13

Assigning 1000 to integers1[5]
integers1:
           8           9          10          11
          12        1000          14          15
          16          17

Attempt to assign 1000 to integers1[15]
An exception occurred: Subscript out of range
```

Fig. 10.9 | `Array` class test program. (Part 7 of 7.)

10.10 Case Study: `Array` Class (cont.)

- The `Array` copy constructor copies the elements of one `Array` into another.
- The copy constructor can also be invoked by writing line 39 as follows:
  - `Array integers3 = integers1;`
- The equal sign in the preceding statement is *not* the assignment operator.
- When an equal sign appears in the declaration of an object, it invokes a constructor for that object.
- This form can be used to pass only a single

10.10 Case Study: `Array` Class (cont.)

- *The array subscript operator `[]` is not restricted for use only with arrays*; it also can be used, for example, to select elements from other kinds of *container classes*, such as strings and dictionaries.
- Also, when `operator[]` functions are defined, *subscripts no longer have to be integers*—characters, strings, floats or even objects of user-defined classes also could be used.

10.10 Case Study: `Array` Class (cont.)

- Each `Array` object consists of a `size` member indicating the number of elements in the `Array` and an `int` pointer—`ptr`—that points to the dynamically allocated pointer-based array of integers managed by the `Array` object.
- When the compiler sees an expression like `cout << arrayObject`, it invokes non-member function `operator<<` with the call
  - `operator<<( cout, arrayObject )`
- When the compiler sees an expression like `cin >> arrayObject`, it invokes non-member function `operator>>` with the call
  - `operator>>( cin, arrayObject )`
- These stream insertion and stream extraction operator functions cannot be members of class `Array`, because the `Array` object is always mentioned on the right side of the stream insertion operator and the stream extraction operator.

10.10 Case Study: `Array` Class (cont.)

- You might be tempted to replace the counter-controlled `for` statement in lines 104–105 and many of the other `for` statements in class `Array`'s implementation with the C++11 range-based for statement.
- Unfortunately, range-based `for` does *not* work with dynamically allocated built-in arrays.

```
1  // Fig. 10.10: Array.h
2  // Array class definition with overloaded operators.
3  #ifndef ARRAY_H
4  #define ARRAY_H
5
6  #include <iostream>
7
8  class Array
9  {
10    friend std::ostream &operator<<( std::ostream &, const Array & );
11    friend std::istream &operator>>( std::istream &, Array & );
12
13 public:
14    explicit Array( int = 10 ); // default constructor
15    Array( const Array & ); // copy constructor
16    ~Array(); // destructor
17    size_t getSize() const; // return size
18
19    const Array &operator=( const Array & ); // assignment operator
20    bool operator==( const Array & ) const; // equality operator
21
```

Fig. 10.10 | `Array` class definition with overloaded operators. (Part 1 of 2.)

```
22      // inequality operator; returns opposite of == operator
23      bool operator!=( const Array &right ) const
24      {
25         return ! ( *this == right ); // invokes Array::operator==
26      } // end function operator!=
27
28      // subscript operator for non-const objects returns modifiable lvalue
29      int &operator[]( int );
30
31      // subscript operator for const objects returns rvalue
32      int operator[]( int ) const;
33   private:
34      size_t size; // pointer-based array size
35      int *ptr; // pointer to first element of pointer-based array
36   }; // end class Array
37
38   #endif
```

Fig. 10.10 | `Array` class definition with overloaded operators. (Part 2 of 2.)

```
1  // Fig. 10.11: Array.cpp
2  // Array class member- and friend-function definitions.
3  #include <iostream>
4  #include <iomanip>
5  #include <stdexcept>
6
7  #include "Array.h" // Array class definition
8  using namespace std;
9
10 // default constructor for class Array (default size 10)
11 Array::Array( int arraySize )
12    : size( arraySize > 0 ? arraySize :
13          throw invalid_argument( "Array size must be greater than 0" ) ),
14      ptr( new int[ size ] )
15 {
16    for ( size_t i = 0; i < size; ++i )
17       ptr[ i ] = 0; // set pointer-based array element
18 } // end Array default constructor
19
20 // copy constructor for class Array;
21 // must receive a reference to an Array
22 Array::Array( const Array &arrayToCopy )
```

Fig. 10.11 | `Array` class member- and `friend`-function definitions. (Part 1 of 6.)