Template

Ren-Shiou Liu
Dept. of Industrial and Information Management
National Cheng Kung University

Lab

- Reimplement class Array from Chapter 11 as a class template
- Demonstrate the new Array class template in a program

```
1  // Fig. 11.6: Array.h
2  // Array class definition with overloaded operators.
3  #ifndef ARRAY_H
4  #define ARRAY_H
5
6  #include <iostream>
7  using namespace std;
8
9  class Array
10 {
11    friend ostream &operator<<( ostream &, const Array & );
12    friend istream &operator>>( istream &, Array & );
13 public:
14    Array( int = 10 ); // default constructor
15    Array( const Array & ); // copy constructor
16    ~Array(); // destructor
17    int getSize() const; // return size
18
19    const Array &operator=( const Array & ); // assignment operator
20    bool operator==( const Array & ) const; // equality operator
21
```

Fig. 11.6 | `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       int size; // pointer-based array size
35       int *ptr; // pointer to first element of pointer-based array
36    }; // end class Array
37
38    #endif
```

Fig. 11.6 | `Array` class definition with overloaded operators. (Part 2 of 2.)

```
1  // Fig 11.7: Array.cpp
2  // Array class member- and friend-function definitions.
3  #include <iostream>
4  #include <iomanip>
5  #include <cstdlib> // exit function prototype
6  #include "Array.h" // Array class definition
7  using namespace std;
8
9  // default constructor for class Array (default size 10)
10 Array::Array( int arraySize )
11 {
12    size = ( arraySize > 0 ? arraySize : 10 ); // validate arraySize
13    ptr = new int[ size ]; // create space for pointer-based array
14
15    for ( int i = 0; i < size; i++ )
16       ptr[ i ] = 0; // set pointer-based array element
17 } // end Array default constructor
18
```

Fig. 11.7 | `Array` class member- and `friend`-function definitions. (Part 1 of 7.)

```
19 // copy constructor for class Array;
20 // must receive a reference to prevent infinite recursion
21 Array::Array( const Array &arrayToCopy )
22    : size( arrayToCopy.size )
23 {
24    ptr = new int[ size ]; // create space for pointer-based array
25
26    for ( int i = 0; i < size; i++ )
27       ptr[ i ] = arrayToCopy.ptr[ i ]; // copy into object
28 } // end Array copy constructor
29
30 // destructor for class Array
31 Array::~Array()
32 {
33    delete [] ptr; // release pointer-based array space
34 } // end destructor
35
36 // return number of elements of Array
37 int Array::getSize() const
38 {
39    return size; // number of elements in Array
40 } // end function getSize
41
```

Fig. 11.7 | `Array` class member- and `friend`-function definitions. (Part 2 of 7.)

```
42  // overloaded assignment operator;
43  // const return avoids: ( a1 = a2 ) = a3
44  const Array &Array::operator=( const Array &right )
45  {
46     if ( &right != this ) // avoid self-assignment
47     {
48        // for Arrays of different sizes, deallocate original
49        // left-side array, then allocate new left-side array
50        if ( size != right.size )
51        {
52           delete [] ptr; // release space
53           size = right.size; // resize this object
54           ptr = new int[ size ]; // create space for array copy
55        } // end inner if
56
57        for ( int i = 0; i < size; i++ )
58           ptr[ i ] = right.ptr[ i ]; // copy array into object
59     } // end outer if
60
61     return *this; // enables x = y = z, for example
62  } // end function operator=
63
```

Fig. 11.7 | `Array` class member- and `friend`-function definitions. (Part 3 of 7.)

```
64  // determine if two Arrays are equal and
65  // return true, otherwise return false
66  bool Array::operator==( const Array &right ) const
67  {
68     if ( size != right.size )
69        return false; // arrays of different number of elements
70
71     for ( int i = 0; i < size; i++ )
72        if ( ptr[ i ] != right.ptr[ i ] )
73           return false; // Array contents are not equal
74
75     return true; // Arrays are equal
76  } // end function operator==
77
```

Fig. 11.7 | `Array` class member- and `friend`-function definitions. (Part 4 of 7.)

```
78  // overloaded subscript operator for non-const Arrays;
79  // reference return creates a modifiable lvalue
80  int &Array::operator[]( int subscript )
81  {
82     // check for subscript out-of-range error
83     if ( subscript < 0 || subscript >= size )
84     {
85        cerr << "\nError: Subscript " << subscript
86           << " out of range" << endl;
87        exit( 1 ); // terminate program; subscript out of range
88     } // end if
89
90     return ptr[ subscript ]; // reference return
91  } // end function operator[]
92
93  // overloaded subscript operator for const Arrays
94  // const reference return creates an rvalue
95  int Array::operator[]( int subscript ) const
96  {
97     // check for subscript out-of-range error
98     if ( subscript < 0 || subscript >= size )
99     {
100       cerr << "\nError: Subscript " << subscript
101          << " out of range" << endl;
```

Fig. 11.7 | `Array` class member- and `friend`-function definitions. (Part 5 of 7.)

```
102         exit( 1 ); // terminate program; subscript out of range
103      } // end if
104
105      return ptr[ subscript ]; // returns copy of this element
106   } // end function operator[]
107
108   // overloaded input operator for class Array;
109   // inputs values for entire Array
110   istream &operator>>( istream &input, Array &a )
111   {
112      for ( int i = 0; i < a.size; i++ )
113         input >> a.ptr[ i ];
114
115      return input; // enables cin >> x >> y;
116   } // end function
117
```

Fig. 11.7 | `Array` class member- and `friend`-function definitions. (Part 6 of 7.)

```
118 // overloaded output operator for class Array
119 ostream &operator<<( ostream &output, const Array &a )
120 {
121    int i;
122
123    // output private ptr-based array
124    for ( i = 0; i < a.size; i++ )
125    {
126       output << setw( 12 ) << a.ptr[ i ];
127
128       if ( ( i + 1 ) % 4 == 0 ) // 4 numbers per row of output
129          output << endl;
130    } // end for
131
132    if ( i % 4 != 0 ) // end last line of output
133       output << endl;
134
135    return output; // enables cout << x << y;
136 } // end function operator<<
```

Fig. 11.7 | `Array` class member- and `friend`-function definitions. (Part 7 of 7.)

```
1  // Fig. 11.8: fig11_08.cpp
2  // Array class test program.
3  #include <iostream>
4  #include "Array.h"
5  using namespace std;
6
7  int main()
8  {
9     Array integers1( 7 ); // seven-element Array
10    Array integers2; // 10-element Array by default
11
12    // print integers1 size and contents
13    cout << "Size of Array integers1 is "
14       << integers1.getSize()
15       << "\nArray after initialization:\n" << integers1;
16
17    // print integers2 size and contents
18    cout << "\nSize of Array integers2 is "
19       << integers2.getSize()
20       << "\nArray after initialization:\n" << integers2;
21
22    // input and print integers1 and integers2
23    cout << "\nEnter 17 integers:" << endl;
24    cin >> integers1 >> integers2;
```

Fig. 11.8 | Array class test program. (Part 1 of 6.)

```
25
26      cout << "\nAfter input, the Arrays contain:\n"
27         << "integers1:\n" << integers1
28         << "integers2:\n" << integers2;
29
30      // use overloaded inequality (!=) operator
31      cout << "\nEvaluating: integers1 != integers2" << endl;
32
33      if ( integers1 != integers2 )
34         cout << "integers1 and integers2 are not equal" << endl;
35
36      // create Array integers3 using integers1 as an
37      // initializer; print size and contents
38      Array integers3( integers1 ); // invokes copy constructor
39
40      cout << "\nSize of Array integers3 is "
41         << integers3.getSize()
42         << "\nArray after initialization:\n" << integers3;
43
44      // use overloaded assignment (=) operator
45      cout << "\nAssigning integers2 to integers1:" << endl;
46      integers1 = integers2; // note target Array is smaller
47
```

Fig. 11.8 | `Array` class test program. (Part 2 of 6.)

```
48       cout << "integers1:\n" << integers1
49          << "integers2:\n" << integers2;
50
51       // use overloaded equality (==) operator
52       cout << "\nEvaluating: integers1 == integers2" << endl;
53
54       if ( integers1 == integers2 )
55          cout << "integers1 and integers2 are equal" << endl;
56
57       // use overloaded subscript operator to create rvalue
58       cout << "\nintegers1[5] is " << integers1[ 5 ];
59
60       // use overloaded subscript operator to create lvalue
61       cout << "\n\nAssigning 1000 to integers1[5]" << endl;
62       integers1[ 5 ] = 1000;
63       cout << "integers1:\n" << integers1;
64
65       // attempt to use out-of-range subscript
66       cout << "\nAttempt to assign 1000 to integers1[15]" << endl;
67       integers1[ 15 ] = 1000; // ERROR: out of range
68    } // end main
```

Fig. 11.8 | `Array` class test program. (Part 3 of 6.)