CASE STUDY

8 Creating an Abstract Array Data Type—Part 1

One of the benefits of object-oriented programming is the ability to create abstract data types that are improvements on built-in data types. As you have seen in Chapter 8, arrays provide no bounds checking in C++. You can, however, create a class that has array-like characteristics and performs bounds checking. For example, look at this `IntList` class.

Contents of `IntList.h`

```
 1 // Class declaration for the IntList class
 2 #ifndef INTLIST_H
 3 #define INTLIST_H
 4
 5 const int MAX_SIZE = 20;
 6
 7 class IntList
 8 {
 9 private:
10    int  list[MAX_SIZE];
11    bool isValid(int);
12 public:
13    // Constructor
14    IntList();
15    bool set(int, int);
16    bool get(int, int&);
17 };
18
19 #endif
```

Contents of `IntList.cpp`

```
 1 // Member function definitions for the IntList class
 2 #include <iostream>
 3 #include "IntList.h"
 4 using namespace std;
 5
 6 /*************************************************************
 7  *                      Constructor                          *
 8  * Initializes each element in the list to zero.             *
 9  *************************************************************/
10 IntList::IntList()
11 {
12    for (int index = 0; index < MAX_SIZE; index++)
13       list[index] = 0;
14 }
15
```

```
16 /***********************************************************
17  *                        isValid                          *
18  * This private member function returns true if the argument  *
19  * is a valid subscript into the list. Otherwise, it displays *
20  * an error message and returns false.                      *
21  ***********************************************************/
22 bool IntList::isValid(int element)
23 {
24    if (element < 0 || element > MAX_SIZE - 1)
25    {
26       cout << "ERROR: " << element;
27       cout << " is an invalid subscript.\n";
28       return false;
29    }
30    else
31       return true;
32 }
33
34 /***********************************************************
35  *                          set                            *
36  * This public member function is passed an element number and *
37  * a value. If the element number is a valid array subscript,  *
38  * the value is stored in the array at that location and the   *
39  * function returns true. Otherwise,the function returns false.*
40  ***********************************************************/
41 bool IntList::set(int element, int value)
42 {
43    if (isValid(element))
44    {
45       list[element] = value;
46       return true;
47    }
48    else
49       return false;
50 }
51
52 /* ***********************************************************
53  *                            get                            *
54  * This public member function is passed an element number. If   *
55  * it is a valid array subscript, the value stored in the array  *
56  * at that location is retrieved and is made available to the    *
57  * calling function by placing it in a reference parameter.      *
58  * The function then returns true. If the element number passed  *
59  * in is not a valid subscript, the function returns false.      *
60  ***********************************************************/
61 bool IntList::get(int element, int &value)
62 {
63    if (isValid(element))
64    {
65       value = list[element];
66       return true;
67    }
68    else
69       return false;
70 }
```

The `IntList` class allows you to store and retrieve numbers in a 20-element array of integers. Here is a synopsis of the members.

`list`	A 20-element array of integers used to hold the list.
`isValid`	This function validates a subscript into the array. It accepts a subscript value as an argument and returns Boolean `true` if the subscript is in the range 0 to 19. If the value is outside that range, an error message is displayed and Boolean `false` is returned.
`Constructor`	The class constructor initializes each element of the `list` array to zero.
`set`	The `set` member function sets a specific element of the `list` array to a value. The first argument is the element subscript and the second argument is the value to be stored in that element. The function uses `isValid` to validate the subscript. If an invalid subscript is passed to the function, no value is stored in the array and Boolean `false` is returned. If the subscript is valid, the function stores the value in the array and returns Boolean `true`.
`get`	The `get` member function retrieves a value from a specific element in the `list` array. The first argument is the subscript of the element whose value is to be retrieved. The function uses `isValid` to validate the subscript. If the subscript is valid, the value is copied into the second argument (which is passed to a reference variable), and Boolean `true` is returned. If the subscript is invalid, no value is retrieved from the array and Boolean `false` is returned.

The following program demonstrates the class. A loop uses the `set` member to fill the array with 9s and prints an asterisk on the screen each time a 9 is successfully stored. Then another loop uses the `get` member to retrieve the values from the array, and prints them on the screen. Finally, a statement uses the `set` member to demonstrate the subscript validation by attempting to store a value in element 50.

IntListTest.cpp

```
// This program tests the IntList class.
#include <iostream>
#include "IntList.h"
using namespace std;
//Remember to add IntList.cpp to the project containing this file.

int main()
{   const int SIZE = 20;
   int x;
   IntList numbers;      // Create an IntList object,
                         // which is an array of ints
   int val;

   // Store 9s in the list and display an asterisk
   // each time a 9 is successfully stored.
   for (x = 0; x < SIZE; x++)
   {
      if (numbers.set(x, 9))
         cout << "* ";
   }
   cout << endl;

   // Display the 9s
   for (x = 0; x < SIZE; x++)
   {
      if (numbers.get(x, val))
         cout << val << " ";
   }
   cout << endl;

   // Attempt to store a value outside the list's bounds.
   if (numbers.set(50, 9))
      cout << "Element 50 successfully set.\n";
   return 0;
}
```

Program Output

```
* * * * * * * * * * * * * * * * * * * *
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
ERROR: 50 is an invalid subscript.
```