CASE STUDY

9 Creating an Abstract Array Data Type—Part 2

In Chapter 8 we demonstrated the `IntList` class, which behaves like 20-element integer array, with the added ability of performing bounds checking. In this section we continue our development of the `IntList` class by adding the following member functions:

`linearSearch`	A function that performs a linear search on the array for a specified value. If the value is found in the array, its subscript is returned. If the value is not found in the array, –1 is returned.
`binarySearch`	A function that performs a binary search on the array for a specified value. If the value is found in the array, its subscript is returned. If the value is not found in the array, –1 is returned.
`bubbleSort`	A function that uses the bubble sort algorithm to sort the array in ascending order.
`selectionSort`	A function that uses the selection sort algorithm to sort the array in ascending order.

The member functions are implemented as simple modifications of the algorithms presented in Chapter 9. The complete code for the class appears here:

Contents of `IntList.h`

```
 1 // Class Declaration for the enhanced 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    int  linearSearch(int);
18    int  binarySearch(int);
19    void bubbleSort();
20    void selectionSort();
21 };
22
23 #endif
```

Contents of IntList.cpp

```
 1 // Member function definitions for the enhanced IntList class
 2 #include <iostream>
 3 #include "IntList.h"
 4 using namespace std;
 5
 6 /*********************************************************
 7  *                          IntList                      *
 8  * This is the default constructor.                      *
 9  * It initializes each element in the list to zero.      *
10  *********************************************************/
11 IntList::IntList()
12 {
13    for (int index = 0; index < MAX_SIZE; index++)
14       list[index] = 0;
15 }
16
17 /*********************************************************
18  *                        isValid                        *
19  * This private member function returns true if the argument  *
20  * is a valid subscript into the list. Otherwise, it displays *
21  * an error message and returns false.                   *
22  *********************************************************/
23 bool IntList::isValid(int element)
24 {
25    if (element < 0 || element > MAX_SIZE - 1)
26    {
27       cout << "ERROR: " << element;
28       cout << " is an invalid subscript.\n";
29       return false;
30    }
31    else
32       return true;
33 }
34
```

```
35 /***************************************************************
36  *                             set                             *
37  * This public member function is passed an element number and  *
38  * a value. If the element number is a valid array subscript,   *
39  * the value is stored in the array at that location and the    *
40  * function returns true. Otherwise,the function returns false. *
41  ***************************************************************/
42 bool IntList::set(int element, int value)
43 {
44    if (isValid(element))
45    {
46       list[element] = value;
47       return true;
48    }
49    else
50       return false;
51 }
52
53 /* ***************************************************************
54  *                             get                               *
55  * This public member function is passed an element number. If   *
56  * it is a valid array subscript, the value stored in the array  *
57  * at that location is retrieved and is made available to the    *
58  * calling function by placing it in a reference parameter.      *
59  * The function then returns true. If the element number passed  *
60  * in is not a valid subscript, the function returns false.      *
61  ***************************************************************/
62 bool IntList::get(int element, int &value)
63 {
64    if (isValid(element))
65    {
66       value = list[element];
67       return true;
68    }
69    else
70       return false;
71 }
72
73 /* ***************************************************************
74  *                       linearSearch                            *
75  * This public member function performs a linear search on the   *
76  * list, looking for value. If it is found, its array subscript *
77  * is returned. Otherwise, -1 is returned, indicating the value *
78  * is not in the array.                                          *
79  ***************************************************************/
```

```
 80 int IntList::linearSearch(int value)
 81 {
 82    int index = 0;                // Used as a subscript to search array
 83    int position = -1;            // Used to record position of search value
 84    bool found = false;           // Flag to indicate if the value was found
 85
 86    while (index < MAX_SIZE && !found)
 87    {
 88       if (list[index] == value)  // If the value is found
 89       {
 90          found = true;           // Set the flag
 91          position = index;       // Record the value's subscript
 92       }
 93       index++;                   // Go to the next element
 94    }
 95    return position;              // Return the position, or -1
 96 }                                // -1 indicates the value was not found
 97
 98 /* ************************************************************
 99  *                       binarySearch                       *
100  * This public member function performs a binary search on the  *
101  * list, looking for value. If it is found, its array subscript *
102  * is returned. Otherwise, -1 is returned, indicating the value *
103  * is not in the array.                                       *
104  ************************************************************/
105 int IntList::binarySearch(int value)
106 {
107    int  first = 0,
108         last = MAX_SIZE - 1,
109         middle,
110         position = -1;
111    bool found = false;
112
113    // First, sort the list.
114    selectionSort();
115
116    while (!found && first <= last)
117    {
118       middle = (first + last) / 2;
119       if (list[middle] == value)    // If value was found
120       {  found = true;
121          position = middle;         // Set position to the location
122       }                             // where it was found
123       else if (list[middle] > value)
124          last = middle - 1;
125       else
126          first = middle + 1;
127    }
128    return position;                 // If position is still -1 it
129 }                                   // indicates value was not found
130
```

```
131 /* ************************************************************
132  *                        bubbleSort                          *
133  * This public member function performs an ascending-order    *
134  * bubble sort on the list.                                   *
135  ************************************************************/
136 void IntList::bubbleSort()
137 {
138    int temp;
139    bool swap;
140
141    do
142    {
143       swap = false;
144       for (int count = 0; count < MAX_SIZE - 1; count++)
145       {
146          if (list[count] > list[count + 1])
147          {
148             temp = list[count];
149             list[count] = list[count + 1];
150             list[count + 1] = temp;
151             swap = true;
152          }
153       }
154    } while (swap);
155 }
156
157 /************************************************************
158  *                      selectionSort                         *
159  * This public member function performs an ascending-order    *
160  * selection sort on the list.                                *
161  ************************************************************/
162 void IntList::selectionSort()
163 {
164    int startScan, minIndex, minValue;
165
166    for (startScan = 0; startScan < MAX_SIZE - 1; startScan++)
167    {
168       minIndex = startScan;
169       minValue = list[startScan];
170       for(int index = startScan + 1; index < MAX_SIZE; index++)
171       {
172          if (list[index] < minValue)
173          {
174             minValue = list[index];
175             minIndex = index;
176          }
177       }
178       list[minIndex] = list[startScan];
179       list[startScan] = minValue;
180    }
181 }
```

The following program demonstrates the selection sort capability of the class by storing 20 random numbers in the array, displaying them, sorting them, and then displaying them again.

SelectSort.cpp

```
// This program uses the IntList class and demonstrates its
// selection sort capability.
#include <iostream>
#include <cstdlib>           // Needed to use the rand() function
#include "IntList.h"
using namespace std;

int main()
{
   const int SIZE = 20;
   IntList numbers;
   int val;

   // Store random numbers in the list
   for (int index = 0; index < SIZE; index++)
   {
      if (!numbers.set(index, rand()))
         cout << "Error storing a value.\n";
   }
   cout << endl;

   // Display the numbers
   for (int index = 0; index < SIZE; index++)
   {
      if (numbers.get(index, val))
         cout << val << endl;
   }
   cout << "Press ENTER to continue...\n";
   cin.get();

   // Sort the numbers using selectionSort
   numbers.selectionSort();

   // Display the numbers
   cout << "Here are the sorted values:\n";
   for (int index = 0; index < SIZE; index++)
   {
      if (numbers.get(index, val))
         cout << val << endl;
   }
   return 0;
}
```

Program Output

```
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
5705
28145
23281
16827
9961
491
2995
11942
4827
5436
Press ENTER to continue...

Here are the sorted values:
41
491
2995
4827
5436
5705
6334
9961
11478
11942
15724
16827
18467
19169
23281
24464
26500
26962
28145
29358
```

The following program demonstrates the class's binary search algorithm. As in the previous program, twenty random numbers are generated and stored in the array. The program displays a list of the numbers and asks the user to pick one. The `binarySearch` function is then used to find that number's subscript position in the sorted array.

BinarySearch.cpp

```
// This program uses the IntList class and demonstrates its
// binary search capability.
#include <iostream>
#include <cstdlib>      // Needed to use the rand() function
#include "IntList.h"
using namespace std;

int main()
{
   const int SIZE = 20;
   IntList numbers;
   int val,
       searchResult;

   // Store random numbers in the list.
   for (int index = 0; index < SIZE; index++)
   {
      if (!numbers.set(index, rand()))
         cout << "Error storing a value.\n";
   }
   cout << endl;

   // Display the numbers
   for (int index = 0; index < SIZE; index++)
   {
      if (numbers.get(index, val))
         cout << val << endl;
   }
   cout << "Enter one of the numbers shown above: ";
   cin  >> val;

   // Search the list for the entered value
   cout << "Searching...\n";
   searchResult = numbers.binarySearch(val);
   if (searchResult == -1)
      cout << "That value was not found in the array.\n";
   else
   {
      cout << "After the array was sorted, that value\n";
      cout << "is found at subscript " << searchResult << endl;
   }
   return 0;
}
```

Program Output

```
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
5705
28145
23281
16827
9961
491
2995
11942
4827
5436
Enter one of the numbers shown above: 5705
Searching...
After the array was sorted, that value
is found at subscript 5
```