STAT40780 Data Programming with C (online)

Lab Sheet 6 (Solutions)

Dr Marie Galligan

Summer 2015

This week's lab requires you to do some reading, and to modify a C++ class definition.

1 Read up on C++ container classes

The following webpage discusses container classes in C++ and defines a simple array container class. Examine and become familar with the source code.

`http://www.learncpp.com/cpp-tutorial/104-container-classes/`

I will also include a pdf of the tutorial on Blackboard.

2 Modify the container class

The second task for this week is to modify the array container class defined in the tutorial above. The modified class definition should include a function **minArray** that finds the minimum value in the array, and prints the minimum value as "The minimum value of the array is ...".

You will also need to modify the `main()` function so that it calls the `minArray` function. Use **endl** to move the next output to a new line.

A possible solution

Modification to the class definition (saved in header file classArray.h)

```
1
2  //
3  //  array_class.h
4  //  from website http://www.learncpp.com/cpp-tutorial/104-container-classes/
5  //
6  //  Created by Marie Galligan on 04/06/2015.
7  //
8  //
9
10 #ifndef ARRAYCLASS_H
11 #define ARRAYCLASS_H
12
13 #include <assert.h> // for assert()
14 #include <iostream>
15
16
17 using namespace std;
18
19 class IntArray
20 {
21 private:
22     int m_nLength;
23     int *m_pnData;
24
25 public:
26     IntArray()
27     {
28         m_nLength = 0;
29         m_pnData = 0;
30     }
31
32     IntArray(int nLength)
33     {
34         m_pnData = new int[nLength];
35         m_nLength = nLength;
36     }
37
38     ~IntArray()
39     {
40         delete[] m_pnData;
41     }
42
43     void Erase()
44     {
45         delete[] m_pnData;
46         // We need to make sure we set m_pnData to 0 here, otherwise it will
47         // be left pointing at deallocated memory!
48         m_pnData = 0;
49         m_nLength = 0;
50     }
51
52     int& operator[](int nIndex)
53     {
54         assert(nIndex >= 0 && nIndex < m_nLength);
55         return m_pnData[nIndex];
56     }
57
58     // Reallocate resizes the array.  Any existing elements will be destroyed.
59     // This function operates quickly.
60     void Reallocate(int nNewLength)
61     {
62         // First we delete any existing elements
```

```
63          Erase();
64
65          // If our array is going to be empty now, return here
66          if (nNewLength<= 0)
67              return;
68
69          // Then we have to allocate new elements
70          m_pnData = new int[nNewLength];
71          m_nLength = nNewLength;
72      }
73
74      // Resize resizes the array.  Any existing elements will be kept.
75      // This function operates slowly.
76      void Resize(int nNewLength)
77      {
78          // If we are resizing to an empty array, do that and return
79          if (nNewLength <= 0)
80          {
81              Erase();
82              return;
83          }
84
85          // Now we can assume nNewLength is at least 1 element.
86          // This algorithmworks as follows:
87          //  First we are going to allocate a new array.  Then we
88          // are going to copy elements from the existing array
89          // to the new array. Once that is done,
90          //  we can destroy the old array, and make m_pnData
91          // point to the new array.
92
93          // First we have to allocate a new array
94          int *pnData = new int[nNewLength];
95
96          // Then we have to figure out how many elements to
97          // copy from the existing array to the new array.
98          //  We want to copy as many elements as there are
99          // in the smaller of the two arrays.
100         if (m_nLength > 0)
101         {
102
103             //the ? : operator works like an ifelse statement in R
104             int nElementsToCopy =
105                 (nNewLength > m_nLength) ? m_nLength : nNewLength;
106
107             // Now copy the elements one by one
108             for (int nIndex=0; nIndex < nElementsToCopy; nIndex++)
109                 pnData[nIndex] = m_pnData[nIndex];
110         }
111
112         // Now we can delete the old array because we don't need it any more
113         delete[] m_pnData;
114
115         // And use the new array instead!  Note that this simply
116         // makes m_pnData point to the same address as the
117         //  new array we dynamically allocated.  Because
118         // pnData was dynamically allocated, it won't be destroyed
119         //when it goes out of scope.
120         m_pnData = pnData;
121         m_nLength = nNewLength;
122     }
123
124
125     void InsertBefore(int nValue, int nIndex)
126     {
127         // Sanity check our nIndex value
128         assert(nIndex >= 0 && nIndex <= m_nLength);
129
130         // First create a new array one element larger than the old array
```

```
131         int *pnData = new int[m_nLength+1];
132
133         // Copy all of the elements up to the index
134         for (int nBefore=0; nBefore < nIndex; nBefore++)
135             pnData[nBefore] = m_pnData[nBefore];
136
137         // insert our new element into the new array
138         pnData[nIndex] = nValue;
139
140         // Copy all of the values after the inserted element
141         for (int nAfter=nIndex; nAfter < m_nLength; nAfter++)
142             pnData[nAfter+1] = m_pnData[nAfter];
143
144         // Finally, delete the old array, and use the new array instead
145         delete[] m_pnData;
146         m_pnData = pnData;
147         m_nLength += 1;
148     }
149
150     void Remove(int nIndex)
151     {
152         // Sanity check our nIndex value
153         assert(nIndex >= 0 && nIndex < m_nLength);
154
155         // First create a new array one element smaller than the old array
156         int *pnData = new int[m_nLength-1];
157
158         // Copy all of the elements up to the index
159         for (int nBefore=0; nBefore < nIndex; nBefore++)
160             pnData[nBefore] = m_pnData[nBefore];
161
162         // Copy all of the values after the inserted element
163         for (int nAfter=nIndex+1; nAfter < m_nLength; nAfter++)
164             pnData[nAfter-1] = m_pnData[nAfter];
165
166         // Finally, delete the old array, and use the new array instead
167         delete[] m_pnData;
168         m_pnData = pnData;
169         m_nLength -= 1;
170     }
171
172     // A couple of additional functions just for convenience
173     void InsertAtBeginning(int nValue) { InsertBefore(nValue, 0); }
174     void InsertAtEnd(int nValue) { InsertBefore(nValue, m_nLength); }
175
176     int GetLength() { return m_nLength; }
177
178     //function to find the minimum value of the array
179     void minArray()
180     {
181         int minVal = m_pnData[ 0 ];
182
183         for( int j = 0; j < m_nLength; j++ ){
184
185             if( m_pnData[ j ] < minVal )
186             {
187                 minVal = m_pnData[ j ];
188             }
189
190         }
191         cout << "Minimum value of the array is " << minVal << endl ;
192     } //end of minArray()
193
194
195 };  //end of class definition
196
197 #endif
198 //end of header file
```

Modification to the main() function

```
1
2  //
3  //  arrayClass.cpp
4  //
5  //
6  //  Created by Marie Galligan on 04/06/2015.
7  //
8  //
9
10
11 #include <iostream>
12 #include "arrayClass.h"
13
14 using namespace std;
15
16 int main()
17 {
18     // Declare an array with 10 elements
19     //calls the constructor of the class
20     IntArray cArray(10); //Passes argument of 10 to the constructor
21
22     // Fill the array with numbers 1 through 10
23     for (int i=0; i<10; i++)
24         cArray[i] = i+1;     // the [] operator is overloaded
25                              //  to allow access to elements of the
26                              //array data member
27
28     // Resize the array to 8 elements
29     cArray.Resize(8);   //calling the Resize member function
30
31     // Insert the number 20 before the 5th element
32     cArray.InsertBefore(20, 5);
33
34     // Remove the 3rd element
35     cArray.Remove(3);
36
37     // Add 30 and 40 to the end and beginning
38     cArray.InsertAtEnd(30);
39     cArray.InsertAtBeginning(40);
40
41     //Find and print the minimum value of the array
42     cArray.minArray();
43
44     // Print out all the numbers
45     for (int j=0; j<cArray.GetLength(); j++)
46         cout << cArray[j] << endl ;
47
48     return 0;
49 }
```