```
 1 //
 2 // Shaun Chemplavil U08713628
 3 // shaun.chemplavil@gmail.com
 4 // C/C++ Programming III : Intermediate Programming with Objects
 5 // 151116  Raymond L. Mitchell III
 6 // Array.h
 7 // Win10
 8 // Visual C++ 19.0
 9 //
10 // File contains the class definition for Array template class
11 //
12
13 #ifndef SHAUNCHEMPLAVIL_ARRAY_H
14 #define SHAUNCHEMPLAVIL_ARRAY_H
15
16 #include<iostream>
17 #include <stdexcept>
18
19 using std::invalid_argument;
20 using std::cerr;
21
22 namespace ShaunChemplavil
23 {
24    template <typename ElemType = int, int SIZE = 3>
25    class Array
26    {
27    public:
28       // Default Constructor
29       Array() {}
30
31       // Copy Constructor
32       Array(const Array &source)
33       {
34          for (int idx = 0; idx < SIZE; idx++)
35             elements[idx] = source.elements[idx];
36       }
37
38       // Copy Assignment Operator
39       const Array &operator=(const Array &source)
40       {
41          // Prevent Self-Assignment
42          if (this == &source)
43             return *this;
44
45          for (int idx = 0; idx < SIZE; idx++)
46             elements[idx] = source.elements[idx];
47
48          //*this = source;
49
50          return *this;
51       }
52
```

```
53        bool operator==(const Array &other) const
54        {
55           for (int idx = 0; idx < SIZE; idx++)
56              if (elements[idx] != other.elements[idx])
57                 return false;
58
59           return true;
60        }
61
62        bool operator!=(const Array &other) const
63        {
64           return(!(*this == other));
65        }
66
67        // Subscript operator (L-value version)
68        ElemType& operator[](int index)
69        {
70           try
71           {
72              // check if valid index
73              if ((index < 0) || (index >= SIZE))
74                 throw invalid_argument{"Subscript out of range (L-Value)"};
75
76              return elements[index];
77           }
78           catch (invalid_argument &ex)
79           {
80              cerr << "Invalid Argument: " << ex.what() << "\n";
81              return elements[0];
82           }
83        }
84
85        // Subscript operator (R-value version)
86        ElemType operator[](int index) const
87        {
88           try
89           {
90              // check if valid index
91              if ((index < 0) || (index >= SIZE))
92                 throw invalid_argument{"Subscript out of range (R-Value)"};
93
94              return elements[index];
95           }
96           catch (invalid_argument &ex)
97           {
98              cerr << "Invalid Argument: " << ex.what() << "\n";
99              return elements[0];
100          }
101       }
102
103    private:
104       ElemType elements[SIZE];
```

```
105     };
106 }
107
108 #endif
109
```