```cpp
//
// Shaun Chemplavil U08713628
// shaun.chemplavil@gmail.com
// C/C++ Programming III : Intermediate Programming with Objects
// 151116  Raymond L. Mitchell III
// Array.h
// Win10
// Visual C++ 19.0
//
// File contains the class definition for Array template class
//

#ifndef SHAUNCHEMPLAVIL_ARRAY_H
#define SHAUNCHEMPLAVIL_ARRAY_H

#include<iostream>
#include <stdexcept>

using std::invalid_argument;
using std::cerr;

namespace ShaunChemplavil
{
   template <typename ElemType = int, int SIZE = 3>
   class Array
   {
   public:
      // Default Constructor
      Array() {}

      // Copy Constructor
      Array(const Array &source)
      {
         for (int idx = 0; idx < SIZE; idx++)
            elements[idx] = source.elements[idx];
      }

      // Copy Assignment Operator
      const Array &operator=(const Array &source)
      {
         // Prevent Self-Assignment
         if (this == &source)
            return *this;

         for (int idx = 0; idx < SIZE; idx++)
            elements[idx] = source.elements[idx];

         //*this = source;

         return *this;
      }

```

```
      bool operator==(const Array &other) const
      {
         for (int idx = 0; idx < SIZE; idx++)
            if (elements[idx] != other.elements[idx])
               return false;

         return true;
      }

      bool operator!=(const Array &other) const
      {
         return(!(*this == other));
      }

      // Subscript operator (L-value version)
      ElemType& operator[](int index)
      {
         try
         {
            // check if valid index
            if ((index < 0) || (index >= SIZE))
               throw invalid_argument{"Subscript out of range (L-Value)"};

            return elements[index];
         }
         catch (invalid_argument &ex)
         {
            cerr << "Invalid Argument: " << ex.what() << "\n";
            return elements[0];
         }
      }

      // Subscript operator (R-value version)
      ElemType operator[](int index) const
      {
         try
         {
            // check if valid index
            if ((index < 0) || (index >= SIZE))
               throw invalid_argument{"Subscript out of range (R-Value)"};

            return elements[index];
         }
         catch (invalid_argument &ex)
         {
            cerr << "Invalid Argument: " << ex.what() << "\n";
            return elements[0];
         }
      }

   private:
      ElemType elements[SIZE];
```

```
    };
}

#endif

```

```cpp
//
// Shaun Chemplavil U08713628
// shaun.chemplavil@gmail.com
// C/C++ Programming III : Intermediate Programming with Objects
// 151116  Raymond L. Mitchell III
// hw6.cpp
// Win10
// Visual C++ 19.0
//
// Test Program for the Array Template class
//

#include "Array.h"

#include <iostream>

using std::cout;
using ShaunChemplavil::Array;

const int ARRAY_SIZE = 5;

int main()
{
   // a ) Creating default array with " << ARRAY_SIZE << " ints
   Array<int, ARRAY_SIZE> arrayOfFiveInts;

   // b) Modifying array with arbitrary values
   for (int idx = 0; idx < ARRAY_SIZE; idx++)
      arrayOfFiveInts[idx] = 2 * idx + 1;

   cout << "c) Outputting array contents\n";
   for (int idx = 0; idx < ARRAY_SIZE; idx++)
      cout << arrayOfFiveInts[idx] << "\n";

   // d) creating a const Array using the copy constructor
   const Array<int, ARRAY_SIZE> constArrayOfFiveInts(arrayOfFiveInts);

   // e) Assigning Array to existing Array using copy assignment
   Array<int, ARRAY_SIZE> copyAssignArrayOfFiveInts;
   copyAssignArrayOfFiveInts = arrayOfFiveInts;

   cout << "f) Comparing two equal arrays using equality operator:\n"
      << "Result: " << (arrayOfFiveInts == constArrayOfFiveInts) << "\n";

   cout << "g) Comparing two equal arrays using inequality operator:\n"
      << "Result: " << (arrayOfFiveInts != copyAssignArrayOfFiveInts) << "\n";

   cout << "h) Show invalid_argument for L-Value with negative subscript:\n";
   copyAssignArrayOfFiveInts[-1] = 5;

   cout << "i) Show invalid_argument for L-Value with > SIZE subscript:\n";
   copyAssignArrayOfFiveInts[ARRAY_SIZE + 1] = 5;
```

```
    cout << "j) Show invalid_argument for R-Value with negative subscript:\n";
    int temp = constArrayOfFiveInts[-1];

    cout << "k) Show invalid_argument for R-Value with > SIZE subscript:\n";
    temp = constArrayOfFiveInts[ARRAY_SIZE + 1];
}

```