```
/*
 * Array.h
 *
 *  Created on: Nov 12, 2013
 *      Author: Nathaniel Gallinger
 */

#ifndef ARRAY_H_
#define ARRAY_H_

#include <stdexcept>
using std::invalid_argument;

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

    // Copy Constructor
    Array(ElemType source[SIZE])
    {
      this->elements = source;
    }

    // Overloaded = operator
    Array<ElemType, SIZE> operator=(ElemType source[SIZE])
    {
      ElemType retval[SIZE];

      for (int idx = 0; idx < SIZE; idx++) {
        retval[idx] = source[idx];
      }

      return retval;
    }

    // Overloaded == operator
    bool operator==(const Array<ElemType, SIZE> other)
    {
      bool retval = true;

      for (int idx = 0; idx < SIZE; idx++) {
        if(!(this->elements[idx] == other[idx]))
          retval = false;
      }

      return retval;
    }

    // Overloaded != operator
    bool operator!=(const Array<ElemType, SIZE> other)
    {
      bool retval = true;

      for (int idx = 0; idx < SIZE; idx++) {
        if(!(this->elements[idx] == other[idx]))
          retval = false;
      }

      return !retval;
    }

    // Overloaded [] operator, L-value
    ElemType &operator[](int index)
    {
      if((index < 0) || (index >= SIZE))
        throw invalid_argument("subscript index is out of range");

      return this->elements[index];
    }

    // Overloaded [] operator, L-value
    ElemType operator[](int index) const
    {
      if((index < 0) || (index >= SIZE))
        throw invalid_argument("subscript index is out of range");

      return this->elements[index];
    }

  private:
    ElemType elements[SIZE];
  };
}


#endif /* ARRAY_H_ */


/*
 * hw6.c
 *
 *  Created on: Nov 12, 2013
 *      Author: Nathaniel Gallinger
 */
#include "Array.h"
#include <iostream>
#include <stdexcept>

using std::cerr;
using std::cout;
using std::invalid_argument;
using namespace NathanielGallinger;

int main()
{
  const char ARRAY_LENGTH = 5;

  // Default constructor
  Array<int, ARRAY_LENGTH> arrayOfFiveInts;

  // Modify with L-value subscript operator
  for (int idx = 0; idx < ARRAY_LENGTH; idx++) {
    arrayOfFiveInts[idx] = idx;
  }

  // Output with R-value subscript operator
  for (int idx = 0; idx < ARRAY_LENGTH; idx++) {
    cout << arrayOfFiveInts[idx] << "\n";
  }

  // Const array using copy constructor
  const Array<int, ARRAY_LENGTH> constArray(arrayOfFiveInts);

  // Test copy assignment operator
  Array<int, ARRAY_LENGTH> arrayOfFiveInts2;
  arrayOfFiveInts2 = arrayOfFiveInts;

  // Compare arrays using == and !=
  cout << "Are these == ? : " << (arrayOfFiveInts2 == arrayOfFiveInts) << "\n";
  cout << "Are these != ? : " << (arrayOfFiveInts2 != arrayOfFiveInts) << "\n";

  // L-value access < 0
  try
  {
    arrayOfFiveInts[-1] = 10;
  }
  catch (invalid_argument &ex)
  {
    cerr << ex.what() << "\n";
  }

  // L-value access >= SIZE< 0
  try
  {
    arrayOfFiveInts[ARRAY_LENGTH] = 10;
  }
  catch (invalid_argument &ex)
  {
    cerr << ex.what() << "\n";
  }

  // R-value access < 0
  try
  {
    cout << arrayOfFiveInts[-1] << "\n";
  }
  catch (invalid_argument &ex)
  {
    cerr << ex.what() << "\n";
  }

  // R-value access >= SIZE
  try
  {
    cout << arrayOfFiveInts[ARRAY_LENGTH] << "\n";
  }
  catch (invalid_argument &ex)
  {
    cerr << ex.what() << "\n";
  }
}
```

```
Output:
0
1
2
3
4
Are these == ? : 1
Are these != ? : 0
subscript index is out of range
subscript index is out of range
subscript index is out of range
subscript index is out of range
```