Operator Overloading

Lab 7: Operator Overloading for Array Class

Rung-Bin Lin

International Bachelor Program in Informatics
Yuan Ze University

04/08/2021

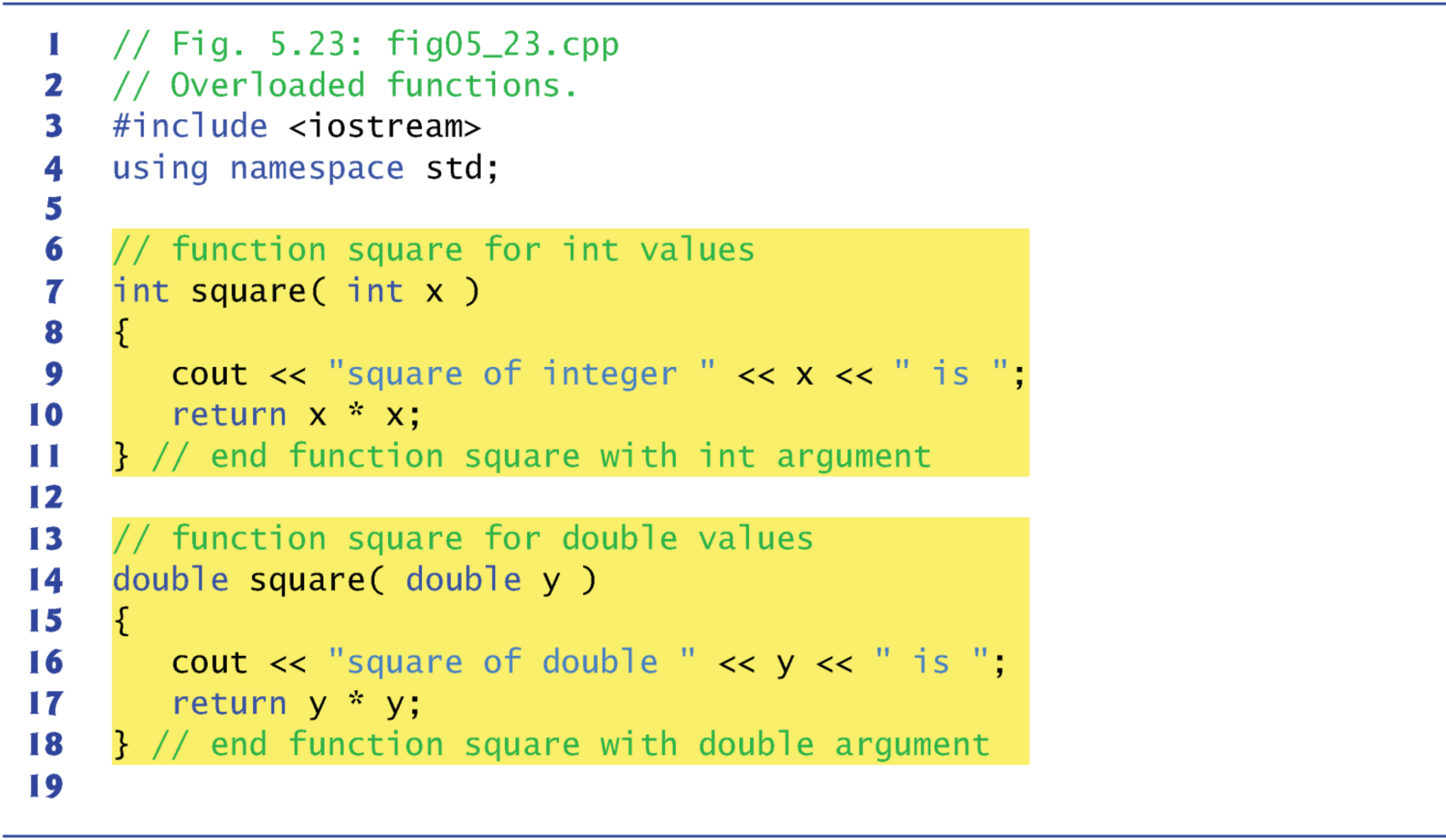
Operator Overloading

- Make the C++ built-in operators available for user-defined (class) objects so that the use of these operators is naturally extended to the objects of user-defined classes.

Operator Overloading VS. Function Overloading

- Operator overloading is a kind of function overloading.
- Function overloading
  - Several functions of the same name can be defined, as long as they have different signatures.
    - A signature is a combination of a function’s name and its parameter types (in order).
  - Overloaded functions can have different return types, but if they do, they must also have different parameter lists.
  - Overloaded functions are normally used to perform similar operations that involve different program logic on different data types.

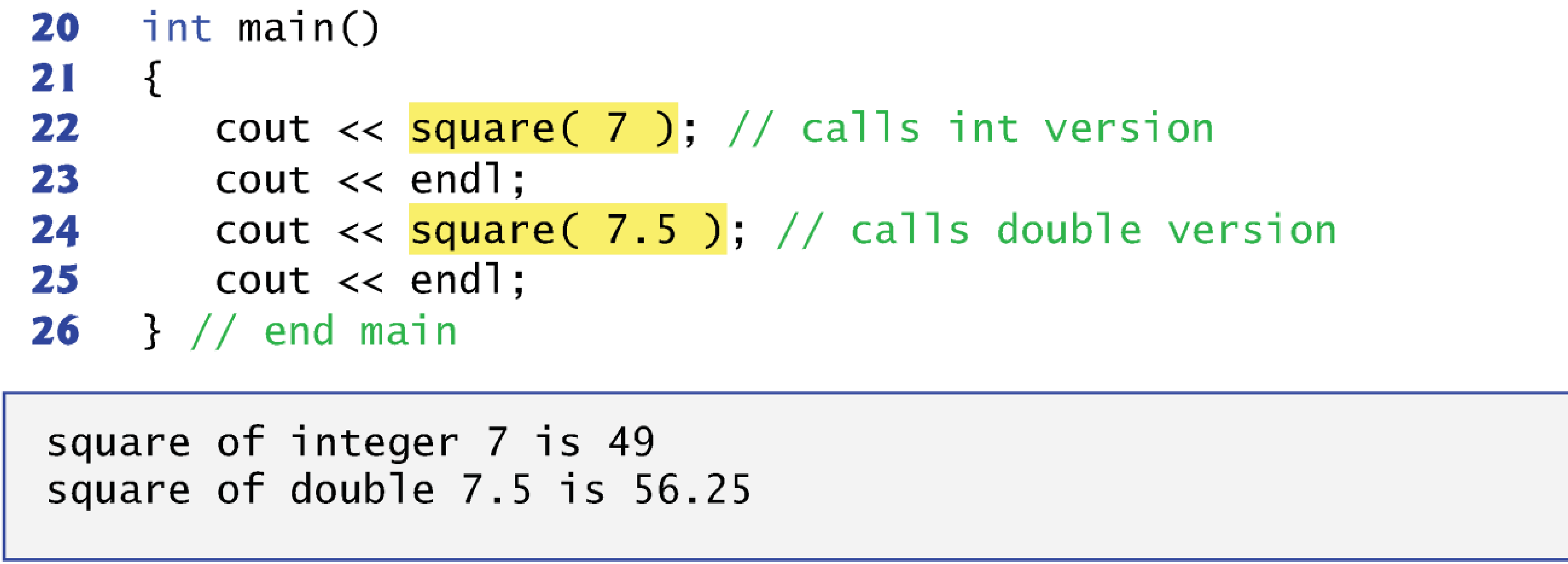
Function Overloading



```
// Fig. 5.23: fig05_23.cpp
// Overloaded functions.
#include <iostream>
using namespace std;

// function square for int values
int square( int x )
{
   cout << "square of integer " << x << " is ";
   return x * x;
} // end function square with int argument

// function square for double values
double square( double y )
{
   cout << "square of double " << y << " is ";
   return y * y;
} // end function square with double argument

```

Fig. 5.23 | Overloaded `square` functions. (Part 1 of 2.)

Function Overloading cont.



```
int main()
{
   cout << square( 7 ); // calls int version
   cout << endl;
   cout << square( 7.5 ); // calls double version
   cout << endl;
} // end main
```

```
square of integer 7 is 49
square of double 7.5 is 56.25
```

Fig. 5.23 | Overloaded `square` functions. (Part 2 of 2.)

Restrictions on Operator Overloading

- Operators that are overloaded as non-static member functions
  - The **leftmost operand** must be an object of the operator’s class.
  - Like addition operator X + Y if X is an object of the operator’s class, but Y may be an object of the operator’s class.
- Operators that are overloaded as global functions
  - The leftmost operand may be an object of a different type or a fundamental type. Like <<, >>, …operators.
  - Usually make friend to the class whose objects will use the operator.
- Operator precedence can not be changed by overloading
- No new operators can be created.

Operators Overloaded as Member Functions

```
class Array {
   friend ostream &operator<<( ostream &, const Array & );
   friend istream &operator>>( istream &, Array & );
public:
   Array( int = 10 ); // default constructor
   Array( const Array & ); // copy constructor
   ~Array(); // destructor
   int getSize() const; // return size
   const Array &operator=( const Array & ); // assignment operator
   bool operator==( const Array & ) const; // equality operator
   // inequality operator; returns opposite of == operator
   bool operator!=( const Array &right ) const  {
      return ! ( *this == right ); // invokes Array::operator==
   } // end function operator!=
   // subscript operator for non-const objects returns modifiable lvalue
   int &operator[]( int );
   // subscript operator for const objects returns rvalue
   int operator[]( int ) const;
private:
   int size; // pointer-based array size
   int *ptr; // pointer to first element of pointer-based array
}; // end class Array
```

Operators Overloaded as Global Functions

```
class Array {
   friend ostream &operator<<( ostream &, const Array & );
   friend istream &operator>>( istream &, Array & );
public:
   Array( int = 10 ); // default constructor
  …
  …
}
```

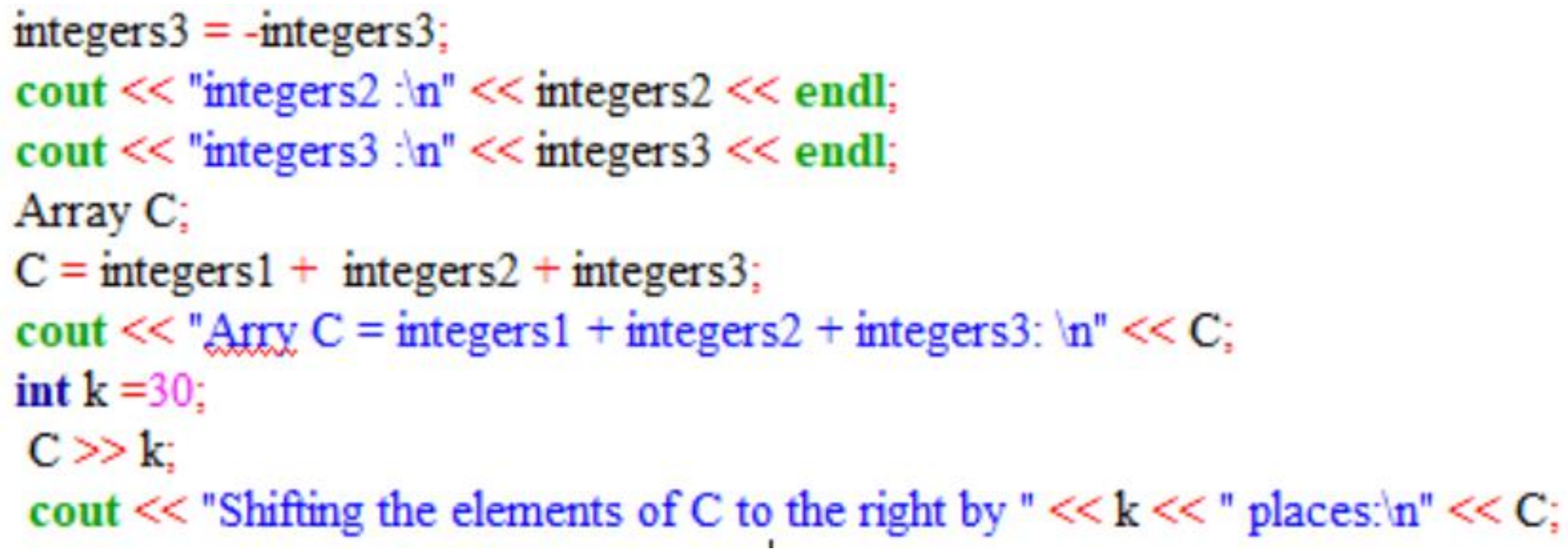
It is important to study the code in Fig. 11.6~Fig. 11.8, Fig. 11.9~Fig. 11.11 .

Lab 7: Class Array

- Add an operator + into the code in Fig. 11.6, 11.7, 11.8 to concatenate two arrays, said array A and array B, into an array, said C. Place the elements of second array after the elements of first array.
  - Example, A=(1,2,3) and B=(4,5,6,7), after executing A+B, C will be (1,2,3,4,5,6,7)
- Add an operator >> to shift the elements in an array of n elements to the right by k places. If k > arraySize, i.e., the size of the array, the elements are moved by (k mod arraySize) places. For example, A>> 1 means that an array element A[i] will be moved to (i+1)th place if i+1<arraySize. A[arraySize-1] will be placed at zeroth place.
  - Example, A=(1,2,3,4,5), after performing A>>7, C will become A=(4,5,1,2,3)
- Add an operator - to negate every element in an array. For example, if A=(1,2,3,4,5), then -A will be (-1, -2, -3, -4, -5).
- Both + and >> are binary operators whereas – is an unary operator.

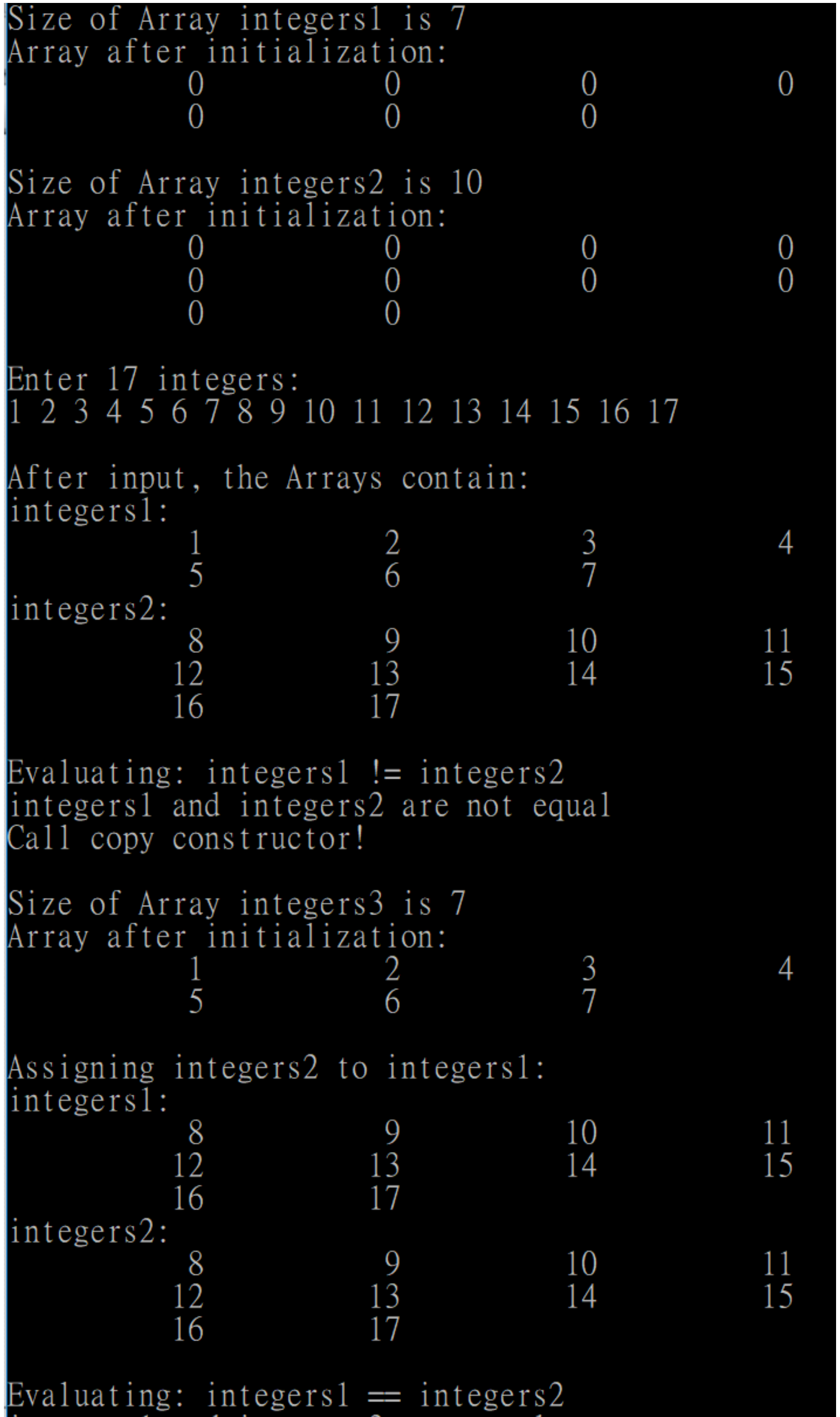
Main() Function

- The main function in Fig. 11.8 should remain the same.
- Before line 65 in the main function in Fig. 11.8, you should add the following statements:

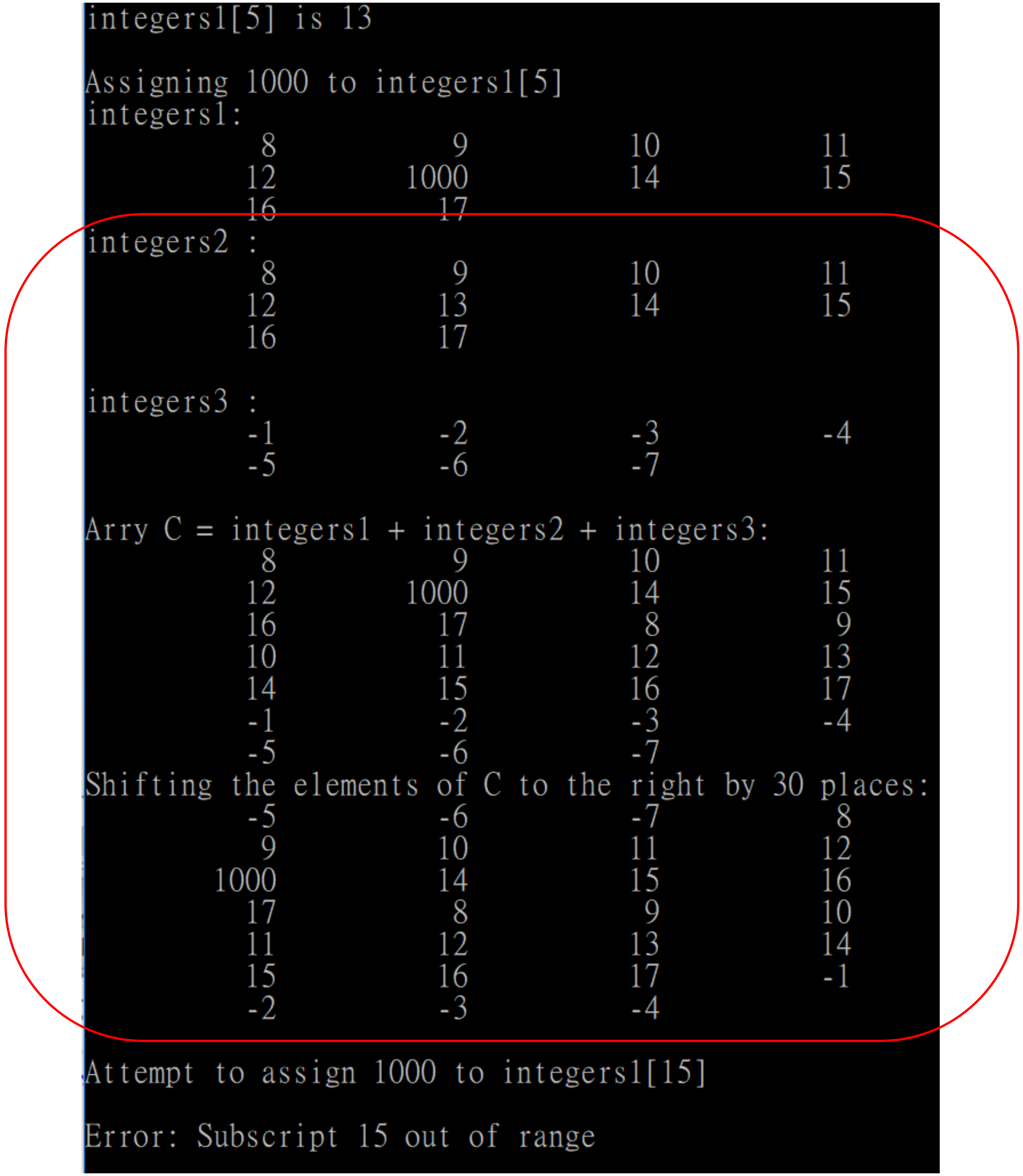


```
integers3 = -integers3;
cout << "integers2 :\n" << integers2 << endl;
cout << "integers3 :\n" << integers3 << endl;
Array C;
C = integers1 +  integers2 + integers3;
cout << "Arry C = integers1 + integers2 + integers3: \n" << C;
int k =30;
 C >> k;
 cout << "Shifting the elements of C to the right by " << k << " places:\n" << C;
```

Output



```
Size of Array integers1 is 7
Array after initialization:
           0           0           0           0
           0           0           0

Size of Array integers2 is 10
Array after initialization:
           0           0           0           0
           0           0           0           0
           0           0

Enter 17 integers:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

After input, the Arrays contain:
integers1:
           1           2           3           4
           5           6           7
integers2:
           8           9          10          11
          12          13          14          15
          16          17

Evaluating: integers1 != integers2
integers1 and integers2 are not equal
Call copy constructor!

Size of Array integers3 is 7
Array after initialization:
           1           2           3           4
           5           6           7

Assigning integers2 to integers1:
integers1:
           8           9          10          11
          12          13          14          15
          16          17
integers2:
           8           9          10          11
          12          13          14          15
          16          17

Evaluating: integers1 == integers2
```

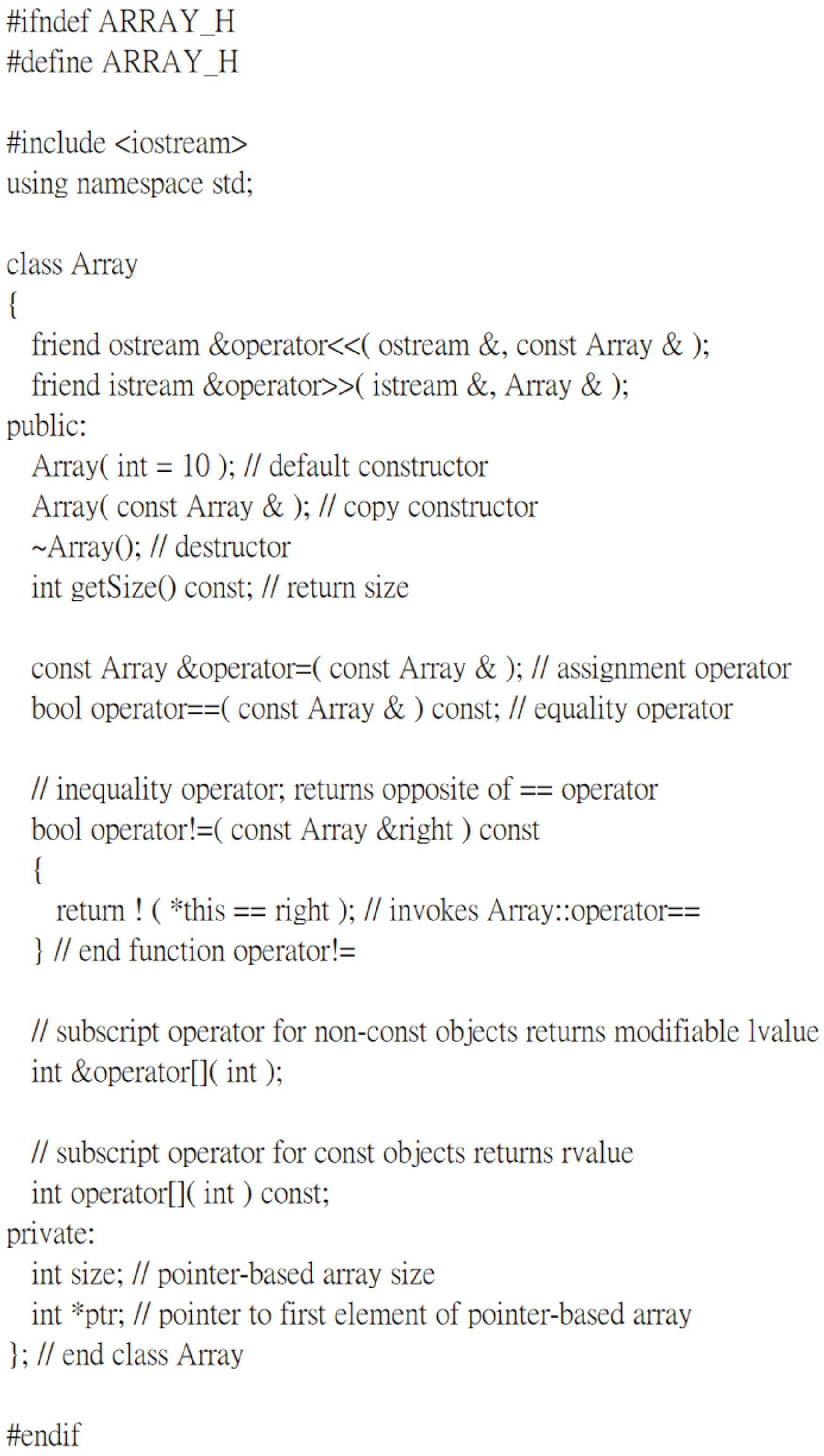


```
integers1[5] is 13

Assigning 1000 to integers1[5]
integers1:
           8           9          10          11
          12        1000          14          15
          16          17
integers2 :
           8           9          10          11
          12          13          14          15
          16          17

integers3 :
          -1          -2          -3          -4
          -5          -6          -7

Arry C = integers1 + integers2 + integers3:
           8           9          10          11
          12        1000          14          15
          16          17           8           9
          10          11          12          13
          14          15          16          17
          -1          -2          -3          -4
          -5          -6          -7
Shifting the elements of C to the right by 30 places:
          -5          -6          -7           8
           9          10          11          12
        1000          14          15          16
          17           8           9          10
          11          12          13          14
          15          16          17          -1
          -2          -3          -4

Attempt to assign 1000 to integers1[15]

Error: Subscript 15 out of range
```

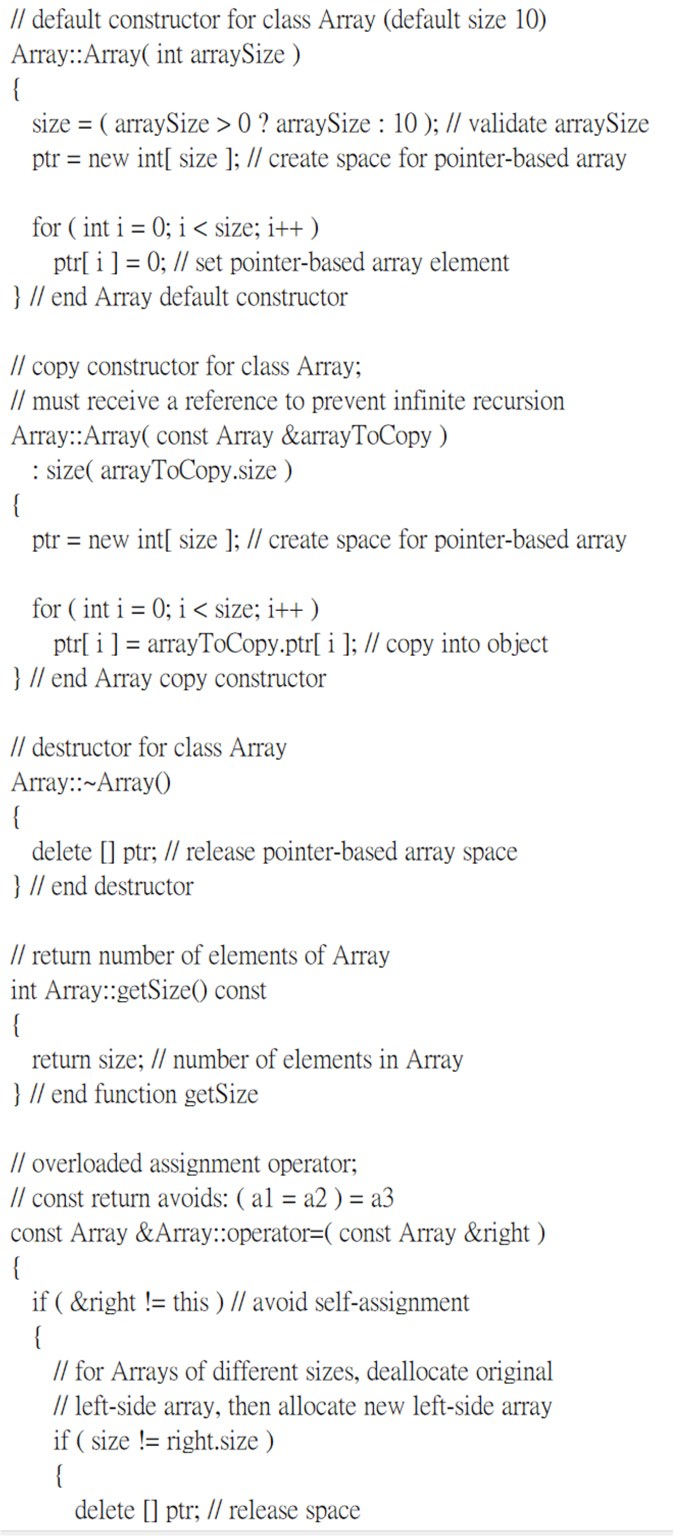
The output marked in the red region should be correct.

Key Points for Grading

- Check whether the overloaded operators +, >>, and - are actually implemented.
- Check whether the added code is actually added in the main() function.
- Check whether the output marked in the red region is indeed correct.



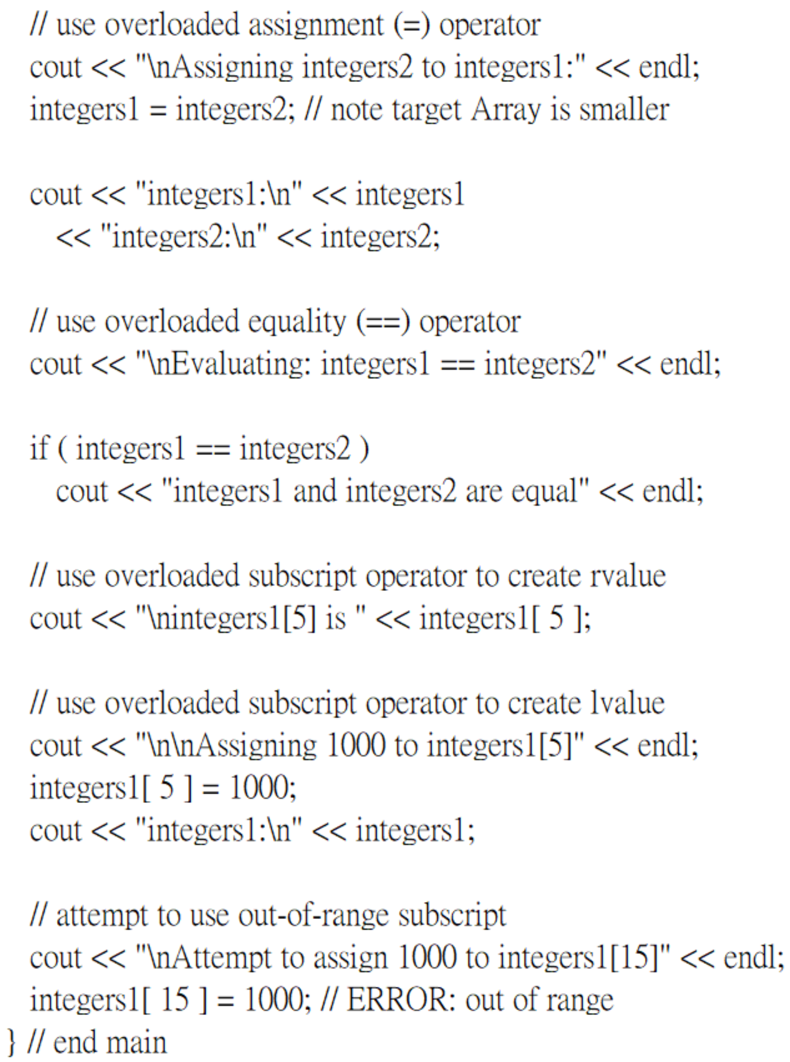
```
#ifndef ARRAY_H
#define ARRAY_H

#include <iostream>
using namespace std;

class Array
{
  friend ostream &operator<<( ostream &, const Array & );
  friend istream &operator>>( istream &, Array & );
public:
  Array( int = 10 ); // default constructor
  Array( const Array & ); // copy constructor
  ~Array(); // destructor
  int getSize() const; // return size

  const Array &operator=( const Array & ); // assignment operator
  bool operator==( const Array & ) const; // equality operator

  // inequality operator; returns opposite of == operator
  bool operator!=( const Array &right ) const
  {
    return ! ( *this == right ); // invokes Array::operator==
  } // end function operator!=

  // subscript operator for non-const objects returns modifiable lvalue
  int &operator[]( int );

  // subscript operator for const objects returns rvalue
  int operator[]( int ) const;
private:
  int size; // pointer-based array size
  int *ptr; // pointer to first element of pointer-based array
}; // end class Array

#endif
```



```
// default constructor for class Array (default size 10)
Array::Array( int arraySize )
{
  size = ( arraySize > 0 ? arraySize : 10 ); // validate arraySize
  ptr = new int[ size ]; // create space for pointer-based array

  for ( int i = 0; i < size; i++ )
    ptr[ i ] = 0; // set pointer-based array element
} // end Array default constructor

// copy constructor for class Array;
// must receive a reference to prevent infinite recursion
Array::Array( const Array &arrayToCopy )
  : size( arrayToCopy.size )
{
  ptr = new int[ size ]; // create space for pointer-based array

  for ( int i = 0; i < size; i++ )
    ptr[ i ] = arrayToCopy.ptr[ i ]; // copy into object
} // end Array copy constructor

// destructor for class Array
Array::~Array()
{
  delete [] ptr; // release pointer-based array space
} // end destructor

// return number of elements of Array
int Array::getSize() const
{
  return size; // number of elements in Array
} // end function getSize

// overloaded assignment operator;
// const return avoids: ( a1 = a2 ) = a3
const Array &Array::operator=( const Array &right )
{
  if ( &right != this ) // avoid self-assignment
  {
    // for Arrays of different sizes, deallocate original
    // left-side array, then allocate new left-side array
    if ( size != right.size )
    {
      delete [] ptr; // release space
```

```
         size = right.size; // resize this object
         ptr = new int[ size ]; // create space for array copy
      } // end inner if

      for ( int i = 0; i < size; i++ )
         ptr[ i ] = right.ptr[ i ]; // copy array into object
   } // end outer if

   return *this; // enables x = y = z, for example
} // end function operator=

// determine if two Arrays are equal and
// return true, otherwise return false
bool Array::operator==( const Array &right ) const
{
   if ( size != right.size )
      return false; // arrays of different number of elements

   for ( int i = 0; i < size; i++ )
      if ( ptr[ i ] != right.ptr[ i ] )
         return false; // Array contents are not equal

   return true; // Arrays are equal
} // end function operator==

// overloaded subscript operator for non-const Arrays;
// reference return creates a modifiable lvalue
int &Array::operator[]( int subscript )
{
   // check for subscript out-of-range error
   if ( subscript < 0 || subscript >= size )
   {
      cerr << "\nError: Subscript " << subscript
         << " out of range" << endl;
      exit( 1 ); // terminate program; subscript out of range
   } // end if

   return ptr[ subscript ]; // reference return
} // end function operator[]
```

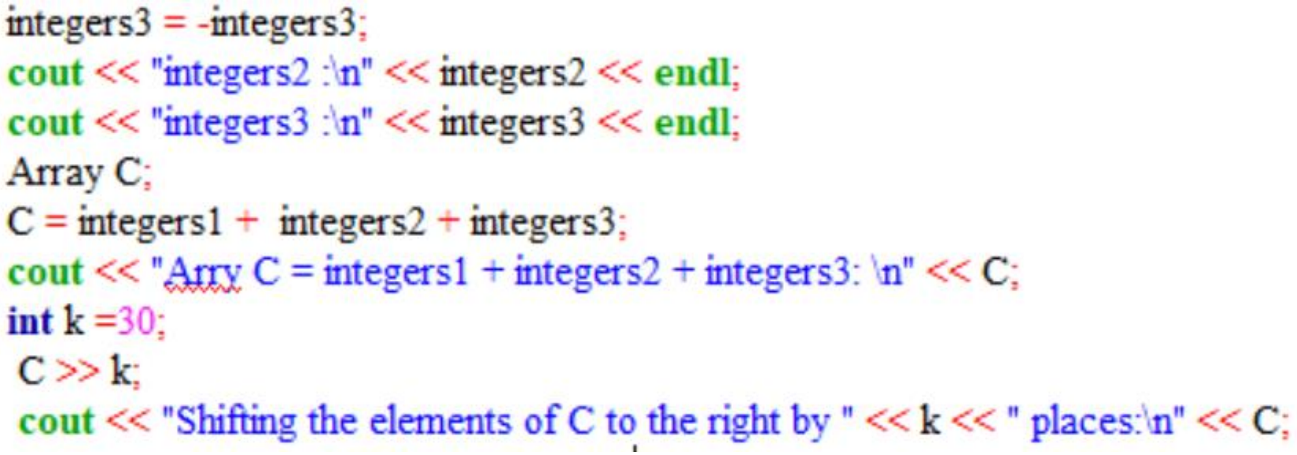
```
int main()
{
   Array integers1( 7 ); // seven-element Array
   Array integers2; // 10-element Array by default

   // print integers1 size and contents
   cout << "Size of Array integers1 is "
      << integers1.getSize()
      << "\nArray after initialization:\n" << integers1;

   // print integers2 size and contents
   cout << "\nSize of Array integers2 is "
      << integers2.getSize()
      << "\nArray after initialization:\n" << integers2;

   // input and print integers1 and integers2
   cout << "\nEnter 17 integers:" << endl;
   cin >> integers1 >> integers2;

   cout << "\nAfter input, the Arrays contain:\n"
      << "integers1:\n" << integers1
      << "integers2:\n" << integers2;

   // use overloaded inequality (!=) operator
   cout << "\nEvaluating: integers1 != integers2" << endl;

   if ( integers1 != integers2 )
      cout << "integers1 and integers2 are not equal" << endl;

   // create Array integers3 using integers1 as an
   // initializer; print size and contents
   Array integers3( integers1 ); // invokes copy constructor

   cout << "\nSize of Array integers3 is "
      << integers3.getSize()
      << "\nArray after initialization:\n" << integers3;
```



```
 // use overloaded assignment (=) operator
 cout << "\nAssigning integers2 to integers1:" << endl;
 integers1 = integers2; // note target Array is smaller

 cout << "integers1:\n" << integers1
    << "integers2:\n" << integers2;

 // use overloaded equality (==) operator
 cout << "\nEvaluating: integers1 == integers2" << endl;

 if ( integers1 == integers2 )
    cout << "integers1 and integers2 are equal" << endl;

 // use overloaded subscript operator to create rvalue
 cout << "\nintegers1[5] is " << integers1[ 5 ];

 // use overloaded subscript operator to create lvalue
 cout << "\n\nAssigning 1000 to integers1[5]" << endl;
 integers1[ 5 ] = 1000;
 cout << "integers1:\n" << integers1;


 // attempt to use out-of-range subscript
 cout << "\nAttempt to assign 1000 to integers1[15]" << endl;
 integers1[ 15 ] = 1000; // ERROR: out of range
} // end main
```

Insert the statements here

The statements being inserted:



```
integers3 = -integers3;
cout << "integers2 :\n" << integers2 << endl;
cout << "integers3 :\n" << integers3 << endl;
Array C;
C = integers1 +  integers2 + integers3;
cout << "Arry C = integers1 + integers2 + integers3: \n" << C;
int k =30;
 C >> k;
 cout << "Shifting the elements of C to the right by " << k << " places:\n" << C;
```