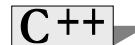


Create a template for the Array class developed in chapter 8.

```
// Simple class Array converted to a template
#ifndef ARRAY1_H
#define ARRAY1_H

#include <iostream.h>
#include <iomanip.h>
#include <assert.h>

template< class T >
class ArrayT {
    friend ostream &operator << ( ostream &, const ArrayT<T> & );
    friend istream &operator >> ( istream &, ArrayT<T> & );
public:
    ArrayT( int = 10 );
    ArrayT( const ArrayT<T> & ); // copy constructor
    ~ArrayT();
    int getSize() const;
    const ArrayT<T> &operator=(const ArrayT<T> &);
    int operator==( const ArrayT<T> & ) const;
    int operator!=( const ArrayT<T> & right ) const
        { return ! ( *this == right ); }
    T &operator[]( int );
    const T &operator[]( int ) const;
    static int getArrayCount();
private:
    int size; // size of the array
    T *ptr; // pointer to first element of array
    static int arrayCount; // # of Arrays instantiated
};
```



```
// Member function definitions for class ArrayT TEMPLATE
// Initialize static data member at file scope
template <class T>
int ArrayT<T>::arrayCount = 0; // no objects yet

// Default constructor for class ArrayT (default size 10)
template <class T>
ArrayT<T>::ArrayT( int arraySize ) {
    size = ( arraySize > 0 ? arraySize : 10 );
    ptr = new T[ size ]; // create space for array
    assert( ptr != 0 ); // exit if memory not allocated
    ++arrayCount; // count one more object
    for ( int i = 0; i < size; i++ )
        ptr[ i ] = 0; // initialize array
}

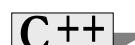
// Copy constructor for class ArrayT
// must receive a reference to prevent infinite recursion
template <class T>
ArrayT<T>::ArrayT( const ArrayT<T> &init ) {
    Size= init.size;
    ptr = new T[ size ]; // create space for array
    assert( ptr != 0 ); // exit if memory not allocated
    ++arrayCount; // count one more object
    for ( int i = 0; i < size; i++ )
        ptr[ i ] = init.ptr[ i ]; // copy init into object
}
```



```
// Destructor for class ArrayT
template <class T>
ArrayT<T>::~ArrayT() {
    delete [] ptr; // reclaim space for array
    --arrayCount; // one fewer objects
}

// Get the size of the array
template <class T>
int ArrayT<T>::getSize() const { return size; }

// Overloaded assignment operator const return avoids: ( a1 = a2 ) = a3
template <class T>
const ArrayT<T> &ArrayT<T>::operator=
    ( const ArrayT<T> &right ) {
    if ( &right != this ) { // check for self-assignment
        // for arrays of different sizes, deallocate original
        // left side array, then allocate new left side array.
        if ( size != right.size ) {
            delete [] ptr; // reclaim space
            size = right.size; // resize this object
            ptr = new T[ size ]; // create space for array copy
            assert( ptr != 0 ); // exit if not allocated
        }
        for ( int i = 0; i < size; i++ )
            ptr[i] = right.ptr[i]; // copy array into object
    }
    return *this; // enables x = y = z;
}
```



```
// Determine if two arrays are equal and return 1, otherwise return 0.
template <class T>
int ArrayT<T>::operator==
    ( const ArrayT<T> &right ) const {
    if ( size != right.size )
        return 0; // arrays of different sizes
    for ( int i = 0; i < size; i++ )
        if ( ptr[ i ] != right.ptr[ i ] )
            return 0; // arrays are not equal
    return 1; // arrays are equal
}

// Overloaded subscript operator for non-const Arrays
// reference return creates an lvalue
template <class T>
T &ArrayT<T>::operator[]( int subscript ) {
    // check for subscript out of range error
    assert( 0 <= subscript && subscript < size );
    return ptr[ subscript ]; // reference return
}

// Overloaded subscript operator for const Arrays
// const reference return creates an rvalue
template <class T>
const T &ArrayT<T>::operator[]( int subscript ) const {
    // check for subscript out of range error
    assert( 0 <= subscript && subscript < size );
    return ptr[ subscript ]; // const reference return
}
```

C++

Exercice #12-1
5

```
// Return the number of ArrayT objects instantiated
// static functions cannot be const
template <class T>
int ArrayT<T>::getArrayCount() {
    return arrayCount;
}

// Overloaded input operator for class ArrayT;
// inputs values for entire array.
template <class T>
istream&operator>>( istream &input, ArrayT<T> &a) {
    for ( int i = 0; i < a.size; i++ )
        input >> a.ptr[ i ];
    return input; // enables cin >> x >> y;
}

// Overloaded output operator for class ArrayT
template <class T>
ostream&operator<<( ostream &output,
                      const ArrayT<T> &a){
    int i;
    for ( i = 0; i < a.size; i++ ) {
        output << setw( 12 ) << a.ptr[ i ];
        if ( ( i + 1 ) % 4 == 0 ) // 4 numbers per row
            output << endl;
    }
    if ( i % 4 != 0 )
        output << endl;
    return output; // enables cout << x << y;
}
#endif
```

C++

Exercice #12-1
6

```
// fig08_04.cpp Driver for simple class ArrayT.
#include "array.h"

int main() {
    // no objects yet
    cout << "# of arrays instantiated = "
        << ArrayT<int>::getArrayCount() << '\n';

    // create two arrays and print ArrayT count
    ArrayT <int>integers1( 7 ), integers2;
    cout << "# of arrays instantiated = "
        << ArrayT<int>::getArrayCount() << "\n\n";

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

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

    // input and print integers1 and integers2
    cout << "Input 17 integers:\n";
    cin >> integers1 >> integers2;
    cout << "After input, the arrays contain:\n"
        << "integers1:\n" << integers1
        << "integers2:\n" << integers2 << '\n';

    // use overloaded inequality (!=) operator
    cout << "Evaluating: integers1 != integers2\n";
    if ( integers1 != integers2 )
        cout << "They are not equal\n";
```

C++

Exercice #12-1
7

```
// create array integers3 using integers1 as an initializer; print size
// and contents
ArrayT <int>integers3( integers1 );
cout << "\nSize of array integers3 is "
    << integers3.getSize()
    << "\nArray after initialization:\n"
    << integers3 << '\n';

// use overloaded assignment (=) operator
cout << "Assigning integers2 to integers1:\n";
integers1 = integers2;
cout << "integers1:\n" << integers1
    << "integers2:\n" << integers2 << '\n';

// use overloaded equality (==) operator
cout << "Evaluating: integers1 == integers2\n";
if ( integers1 == integers2 )
    cout << "They are equal\n\n";

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

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

// attempt to use out of range subscript
cout << "Attempt to assign 1000 to integers1[15]" << endl;
integers1[ 15 ] = 1000; // ERROR: out of range
return 0;
```