Khoi Duong

Prof. Yang

CS360

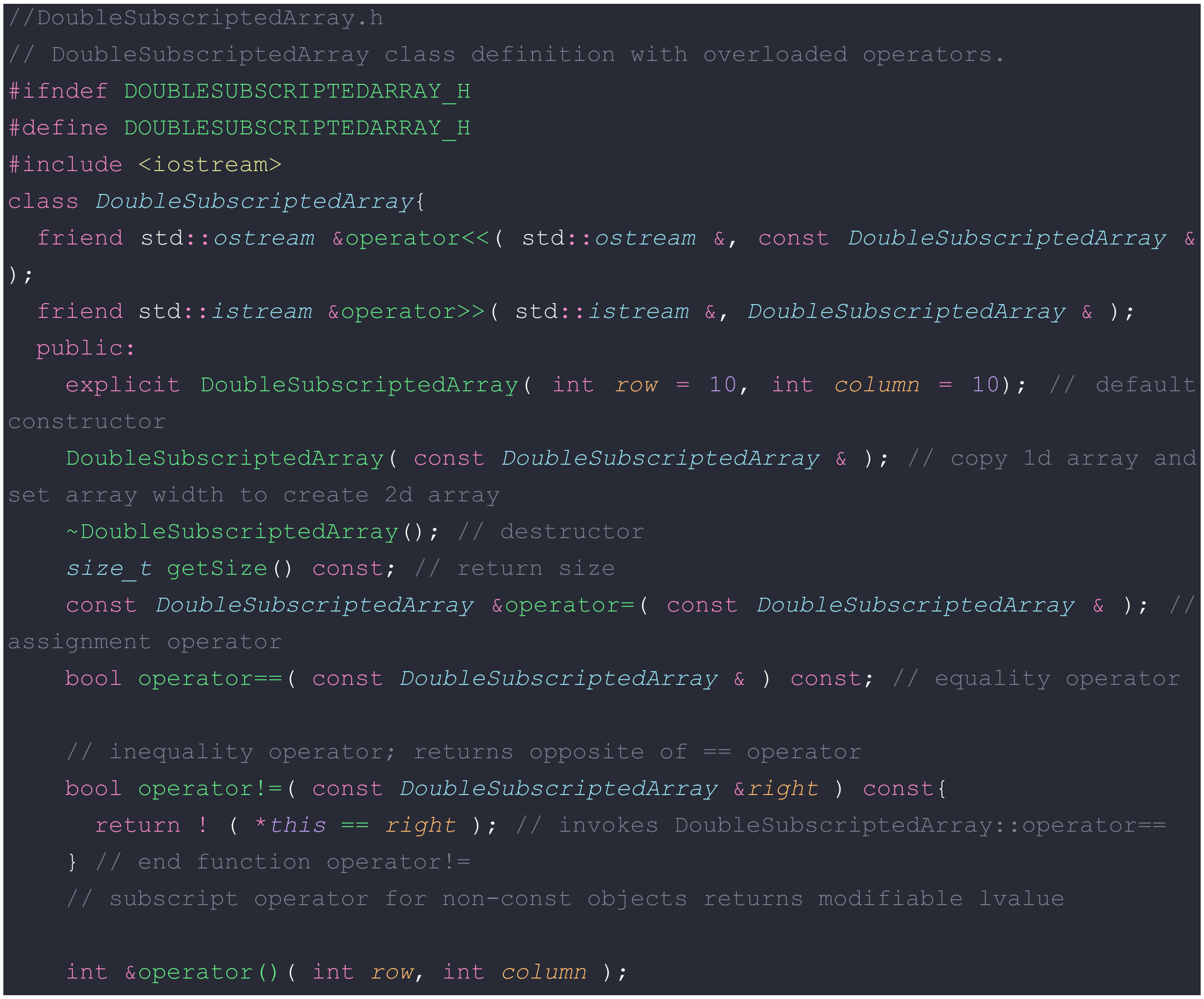
8/2/2022

HW#4

1.

Source code:

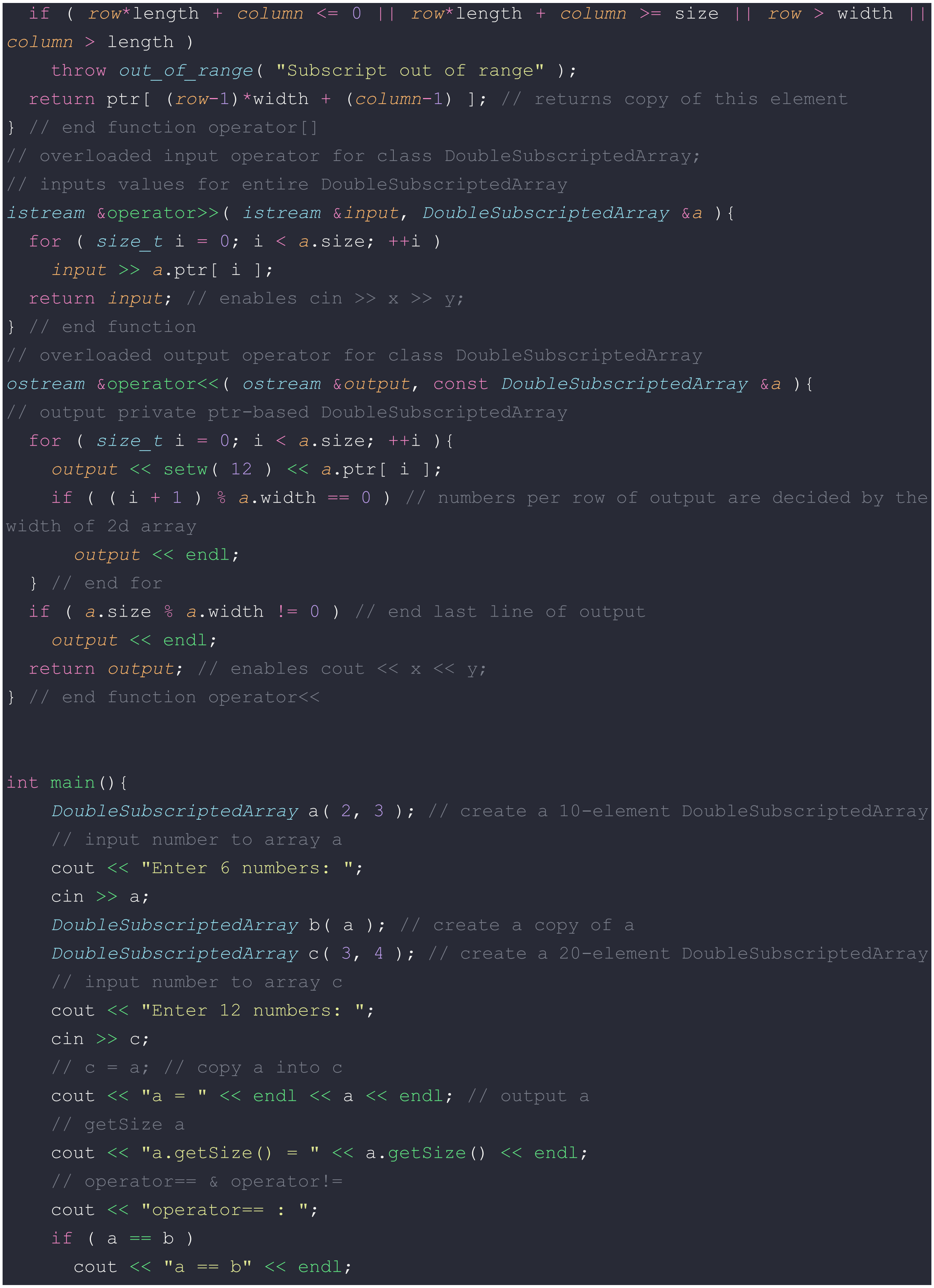
<u>DoubleSubscriptedArray.h</u>



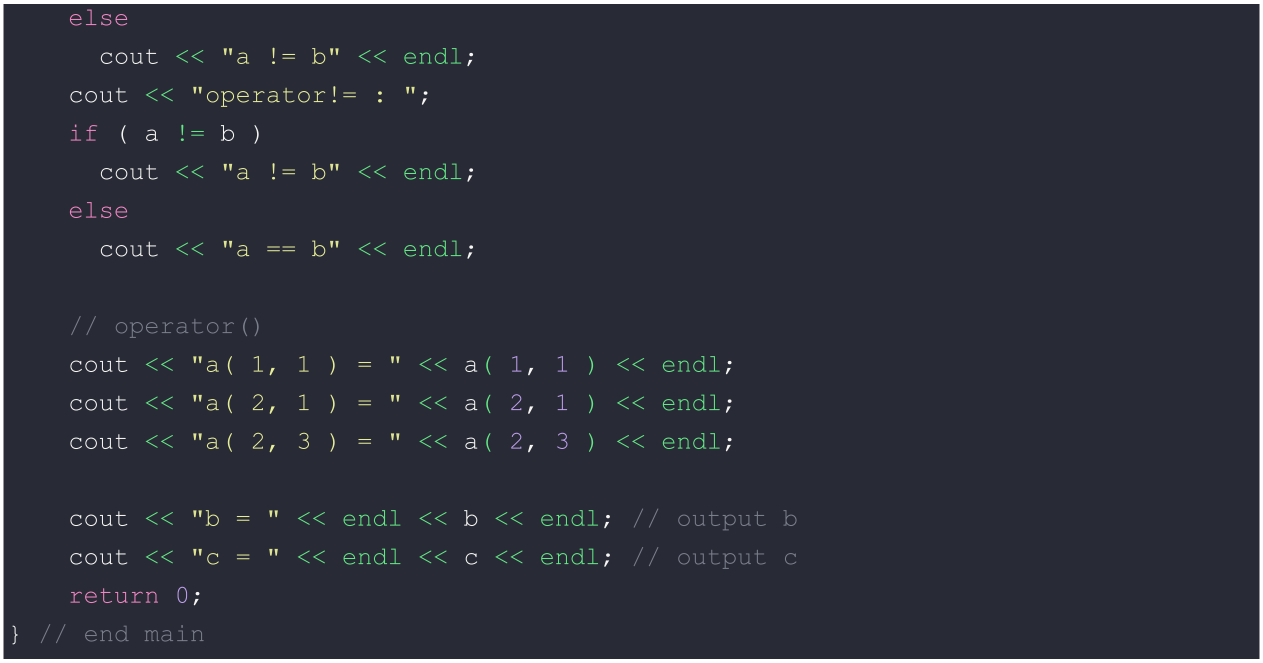
```
//DoubleSubscriptedArray.h
// DoubleSubscriptedArray class definition with overloaded operators.
#ifndef DOUBLESUBSCRIPTEDARRAY_H
#define DOUBLESUBSCRIPTEDARRAY_H
#include <iostream>
class DoubleSubscriptedArray{
  friend std::ostream &operator<<( std::ostream &, const DoubleSubscriptedArray & );
  friend std::istream &operator>>( std::istream &, DoubleSubscriptedArray & );
  public:
    explicit DoubleSubscriptedArray( int row = 10, int column = 10); // default constructor
    DoubleSubscriptedArray( const DoubleSubscriptedArray & ); // copy 1d array and set array width to create 2d array
    ~DoubleSubscriptedArray(); // destructor
    size_t getSize() const; // return size
    const DoubleSubscriptedArray &operator=( const DoubleSubscriptedArray & ); // assignment operator
    bool operator==( const DoubleSubscriptedArray & ) const; // equality operator

    // inequality operator; returns opposite of == operator
    bool operator!=( const DoubleSubscriptedArray &right ) const{
      return ! ( *this == right ); // invokes DoubleSubscriptedArray::operator==
    } // end function operator!=
    // subscript operator for non-const objects returns modifiable lvalue

    int &operator()( int row, int column );
    // subscript operator for const objects returns rvalue
    int operator()( int row, int column ) const;
  private:
    int width, length; // how many rows and columns 2d array has
    size_t size; // pointer-based DoubleSubscriptedArray size
    int *ptr; // pointer to first element of pointer-based DoubleSubscriptedArray
}; // end class DoubleSubscriptedArray
#endif
```

DoubleSubscriptedArray.cpp

```
// DoubleSubscriptedArray class member- and friend-function definitions.
#include <iostream>
#include <iomanip>
#include <stdexcept>
#include "DoubleSubscriptedArray.h" // DoubleSubscriptedArray class definition
using namespace std;
// default constructor for class DoubleSubscriptedArray (default size 10)
DoubleSubscriptedArray::DoubleSubscriptedArray( int r, int c ): size( r > 0 && c >
0? c*r :
  throw invalid_argument( "Array size must be greater than 0" ) ), width(c),
length(r),
  ptr( new int[ size ] )
{
  for ( size_t i = 0; i < size; ++i )
    ptr[ i ] = 0; // set pointer-based array element
} // end DoubleSubscriptedArray default constructor
// copy constructor for class DoubleSubscriptedArray;
// must receive a reference to an DoubleSubscriptedArray
DoubleSubscriptedArray::DoubleSubscriptedArray( const DoubleSubscriptedArray
&DoubleSubscriptedArrayToCopy ): size( DoubleSubscriptedArrayToCopy.size ),
width(DoubleSubscriptedArrayToCopy.width),
  length(DoubleSubscriptedArrayToCopy.length) , ptr( new int[ size ] )
{
  for ( size_t i = 0; i < size; ++i )
    ptr[ i ] = DoubleSubscriptedArrayToCopy.ptr[ i ]; // copy into object
} // end DoubleSubscriptedArray copy constructor
// destructor for class DoubleSubscriptedArray
DoubleSubscriptedArray::~DoubleSubscriptedArray(){
  delete [] ptr; // release pointer-based DoubleSubscriptedArray space
} // end destructor
// return number of elements of DoubleSubscriptedArray
size_t DoubleSubscriptedArray::getSize() const{
  return size; // number of elements in DoubleSubscriptedArray
} // end function getSize
// overloaded assignment operator;
// const return avoids: ( a1 = a2 ) = a3
const DoubleSubscriptedArray &DoubleSubscriptedArray::operator=( const
DoubleSubscriptedArray &right ){
  if ( &right != this )// avoid self-assignment
  {
  // for DoubleSubscriptedArrays of different sizes, deallocate original
  // left-side DoubleSubscriptedArray, then allocate new left-side
DoubleSubscriptedArray
    if ( size != right.size ){
      delete [] ptr; // release space
      size = right.size; // resize this object
      ptr = new int[ size ]; // create space for DoubleSubscriptedArray copy
    } // end inner if
    for ( size_t i = 0; i < size; ++i )
      ptr[ i ] = right.ptr[ i ]; // copy DoubleSubscriptedArray into object
  } // end outer if
  return *this; // enables x = y = z, for example
} // end function operator=
// determine if two DoubleSubscriptedArrays are equal and
// return true, otherwise return false
bool DoubleSubscriptedArray::operator==( const DoubleSubscriptedArray &right )
const{
  if ( size != right.size )
    return false; // DoubleSubscriptedArrays of different number of elements
  for ( size_t i = 0; i < size; ++i )
    if ( ptr[ i ] != right.ptr[ i ] )
      return false; // DoubleSubscriptedArray contents are not equal
  return true; // DoubleSubscriptedArrays are equal
} // end function operator==
// overloaded subscript operator for non-const DoubleSubscriptedArrays;
// reference return creates a modifiable lvalue
int &DoubleSubscriptedArray::operator()( int row, int column ){
// check for subscript out-of-range error
  if ( row*length + column <= 0 || row*length + column >= size || row > width ||
column > length )
    throw out_of_range( "Subscript out of range" );
  return ptr[ (row-1)*width + (column-1)]; // reference return
} // end function operator[]
// overloaded subscript operator for const DoubleSubscriptedArrays
// const reference return creates an rvalue
int DoubleSubscriptedArray::operator()( int row, int column ) const{
// check for subscript out-of-range error
```



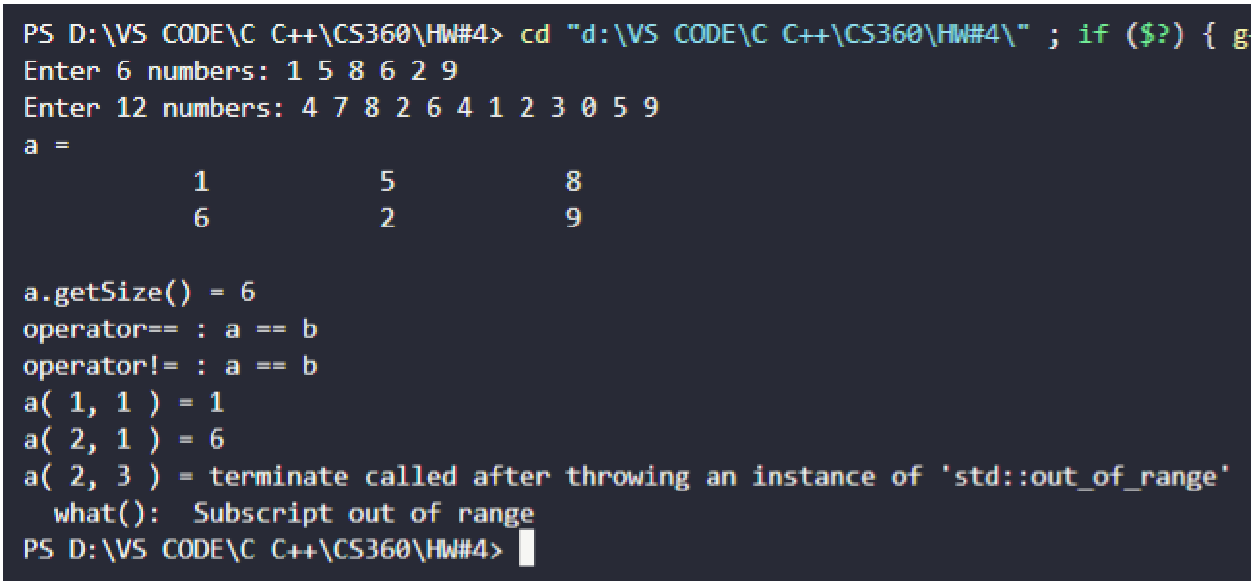
```
  if ( row*length + column <= 0 || row*length + column >= size || row > width ||
column > length )
    throw out_of_range( "Subscript out of range" );
  return ptr[ (row-1)*width + (column-1) ]; // returns copy of this element
} // end function operator[]
// overloaded input operator for class DoubleSubscriptedArray;
// inputs values for entire DoubleSubscriptedArray
istream &operator>>( istream &input, DoubleSubscriptedArray &a ){
  for ( size_t i = 0; i < a.size; ++i )
    input >> a.ptr[ i ];
  return input; // enables cin >> x >> y;
} // end function
// overloaded output operator for class DoubleSubscriptedArray
ostream &operator<<( ostream &output, const DoubleSubscriptedArray &a ){
// output private ptr-based DoubleSubscriptedArray
  for ( size_t i = 0; i < a.size; ++i ){
    output << setw( 12 ) << a.ptr[ i ];
    if ( ( i + 1 ) % a.width == 0 ) // numbers per row of output are decided by the
width of 2d array
      output << endl;
  } // end for
  if ( a.size % a.width != 0 ) // end last line of output
    output << endl;
  return output; // enables cout << x << y;
} // end function operator<<


int main(){
    DoubleSubscriptedArray a( 2, 3 ); // create a 10-element DoubleSubscriptedArray
    // input number to array a
    cout << "Enter 6 numbers: ";
    cin >> a;
    DoubleSubscriptedArray b( a ); // create a copy of a
    DoubleSubscriptedArray c( 3, 4 ); // create a 20-element DoubleSubscriptedArray
    // input number to array c
    cout << "Enter 12 numbers: ";
    cin >> c;
    // c = a; // copy a into c
    cout << "a = " << endl << a << endl; // output a
    // getSize a
    cout << "a.getSize() = " << a.getSize() << endl;
    // operator== & operator!=
    cout << "operator== : ";
    if ( a == b )
      cout << "a == b" << endl;
```



```
    else
      cout << "a != b" << endl;
    cout << "operator!= : ";
    if ( a != b )
      cout << "a != b" << endl;
    else
      cout << "a == b" << endl;

    // operator()
    cout << "a( 1, 1 ) = " << a( 1, 1 ) << endl;
    cout << "a( 2, 1 ) = " << a( 2, 1 ) << endl;
    cout << "a( 2, 3 ) = " << a( 2, 3 ) << endl;

    cout << "b = " << endl << b << endl; // output b
    cout << "c = " << endl << c << endl; // output c
    return 0;
} // end main
```

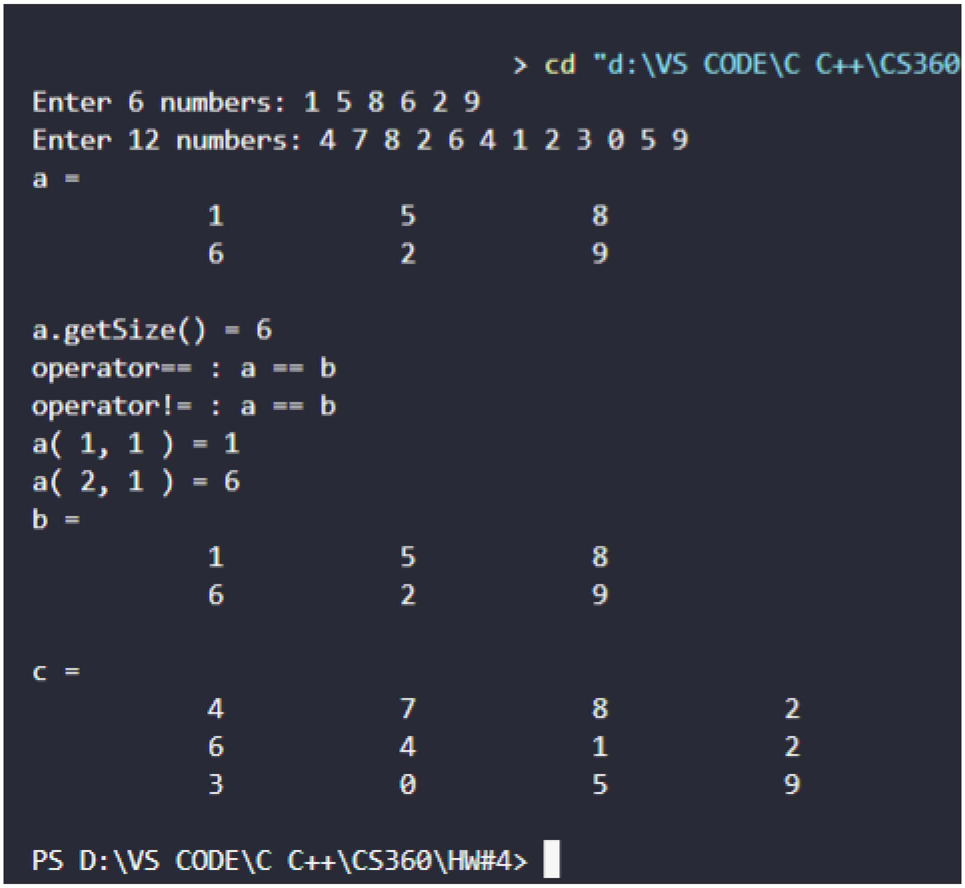
Run program & result:

1st try (with throw_out argument)



```
PS D:\VS CODE\C C++\CS360\HW#4> cd "d:\VS CODE\C C++\CS360\HW#4\" ; if ($?) { g
Enter 6 numbers: 1 5 8 6 2 9
Enter 12 numbers: 4 7 8 2 6 4 1 2 3 0 5 9
a =
           1           5           8
           6           2           9

a.getSize() = 6
operator== : a == b
operator!= : a == b
a( 1, 1 ) = 1
a( 2, 1 ) = 6
a( 2, 3 ) = terminate called after throwing an instance of 'std::out_of_range'
  what():  Subscript out of range
PS D:\VS CODE\C C++\CS360\HW#4> █
```

2nd try (without throw_out argument)



```
                               > cd "d:\VS CODE\C C++\CS360
Enter 6 numbers: 1 5 8 6 2 9
Enter 12 numbers: 4 7 8 2 6 4 1 2 3 0 5 9
a =
           1           5           8
           6           2           9

a.getSize() = 6
operator== : a == b
operator!= : a == b
a( 1, 1 ) = 1
a( 2, 1 ) = 6
b =
           1           5           8
           6           2           9

c =
           4           7           8           2
           6           4           1           2
           3           0           5           9

PS D:\VS CODE\C C++\CS360\HW#4> 
```

2.

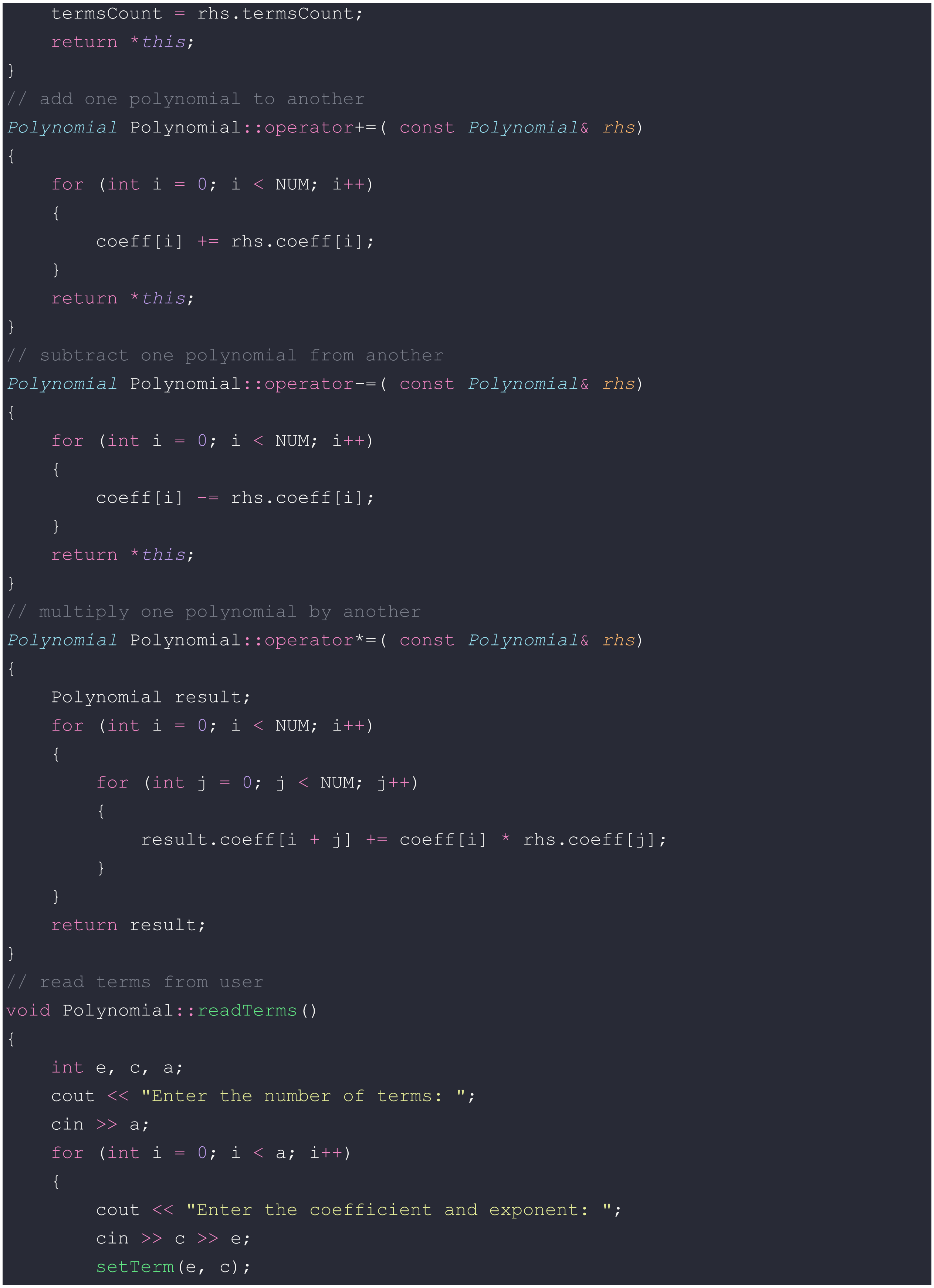
Source code:

<u>Polynomial.h</u>

```
#ifndef POLYNOMIAL_H
#define POLYNOMIAL_H
#include <iostream>
class Polynomial{
    public:
    static const int NUM = 100;
    Polynomial();
    ~Polynomial();
    int getCoefficient(int);
    void setTerm(int, int);
    Polynomial operator+ (const Polynomial&) const;
    Polynomial operator- (const Polynomial&) const;
    Polynomial operator* (const Polynomial&);
    Polynomial operator= (const Polynomial&);
    Polynomial operator+= (const Polynomial&);
    Polynomial operator-= (const Polynomial&);
    Polynomial operator*= (const Polynomial&);
    void readTerms();
    int getTermsCount(Polynomial&);
    void print();
    int getDegree();

    private:
        int coeff[NUM];
        int termsCount;
};
#endif // POLYNOMIAL_H
```

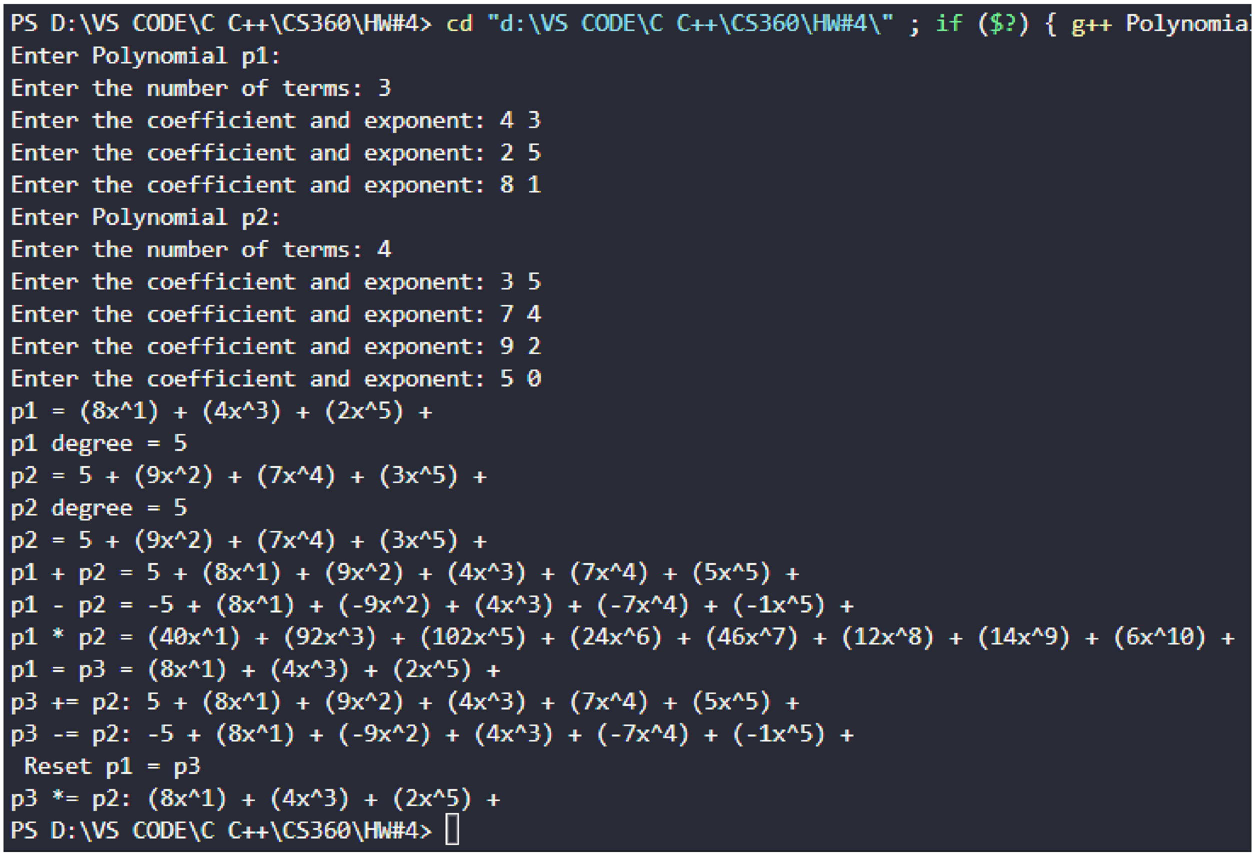
Polynomial.cpp

```
#include <iostream>
#include <iomanip>
#include "Polynomial.h"
using namespace std;
// default constructor for class Polynomial
Polynomial::Polynomial()
{
    for (int i = 0; i < NUM; i++)
    {
        coeff[i] = 0;
    }
    termsCount = 0;
}
// destructor for class Polynomial
Polynomial::~Polynomial(){}

// get a coeff of specific exp in polynomial
int Polynomial::getCoefficient(int i)
{
    return coeff[i];
}
// set a specific term of polynomial
void Polynomial::setTerm(int e, int c)
{
    if (c == 0)
        cout << "Invalid coeff" << endl;
    else
    {
        coeff[e] = c;
        termsCount++;
    }
}
// add two polynomials
Polynomial Polynomial::operator+( const Polynomial& rhs) const
{
    Polynomial result;
    for (int i = 0; i < NUM; i++)
    {
        result.coeff[i] = coeff[i] + rhs.coeff[i];
    }
    return result;
}
// subtract two polynomials
Polynomial Polynomial::operator-( const Polynomial& rhs) const
{
    Polynomial result;
    for (int i = 0; i < NUM; i++)
    {
        result.coeff[i] = coeff[i] - rhs.coeff[i];
    }
    return result;
}
// multiply two polynomials
Polynomial Polynomial::operator*( const Polynomial& rhs)
{
    Polynomial result;
    for (int i = 0; i < NUM; i++)
    {
        for (int j = 0; j < NUM; j++)
        {
            result.coeff[i + j] += coeff[i] * rhs.coeff[j];
        }
    }
    return result;
}
// assign one polynomial to another
Polynomial Polynomial::operator=( const Polynomial& rhs)
{
    for (int i = 0; i < NUM; i++)
    {
        coeff[i] = rhs.coeff[i];
    }
```



```
    termsCount = rhs.termsCount;
    return *this;
}
// add one polynomial to another
Polynomial Polynomial::operator+=( const Polynomial& rhs)
{
    for (int i = 0; i < NUM; i++)
    {
        coeff[i] += rhs.coeff[i];
    }
    return *this;
}
// subtract one polynomial from another
Polynomial Polynomial::operator-=( const Polynomial& rhs)
{
    for (int i = 0; i < NUM; i++)
    {
        coeff[i] -= rhs.coeff[i];
    }
    return *this;
}
// multiply one polynomial by another
Polynomial Polynomial::operator*=( const Polynomial& rhs)
{
    Polynomial result;
    for (int i = 0; i < NUM; i++)
    {
        for (int j = 0; j < NUM; j++)
        {
            result.coeff[i + j] += coeff[i] * rhs.coeff[j];
        }
    }
    return result;
}
// read terms from user
void Polynomial::readTerms()
{
    int e, c, a;
    cout << "Enter the number of terms: ";
    cin >> a;
    for (int i = 0; i < a; i++)
    {
        cout << "Enter the coefficient and exponent: ";
        cin >> c >> e;
        setTerm(e, c);
    }
}
// get the number of terms in polynomial
int Polynomial::getTermsCount(Polynomial& polynomial)
{
    termsCount = 0;
    for (int i = 0; i < NUM; i++)
    {
        if (polynomial.coeff[i] != 0)
            termsCount++;
    }
    return termsCount;
}
// print the polynomial
void Polynomial::print()
{
    if (coeff[0] != 0)
    {
        cout << coeff[0] << " + ";
    }
    for (int i = 1; i < NUM; i++)
    {
        if (coeff[i] != 0)
        {
            cout << "(" << coeff[i] << "x^" << i << ") + ";
        }
    }
    cout << endl;
}
// get the degree of polynomial
int Polynomial::getDegree()
{
    int degree = 0;
    for (int i = 0; i < NUM; i++)
    {
        if (coeff[i] != 0)
        {
            degree = i;
        }
    }
    return degree;
}

int main(){
    Polynomial p1, p2, p3;
    cout << "Enter Polynomial p1: " << endl;
    p1.readTerms();
    cout << "Enter Polynomial p2: " << endl;
    p2.readTerms();
    cout << "p1 = " ; p1.print();
    cout << "p1 degree = ";
    cout << p1.getDegree();
    cout << endl;
    cout << "p2 = " ; p2.print();
    cout << "p2 degree = ";
    cout << p2.getDegree();
    cout << endl;
    cout << "p2 = " ; p2.print();
    p3 = p1 + p2;
    cout << "p1 + p2 = " ; p3.print();
    p3 = p1 - p2;
    cout << "p1 - p2 = " ; p3.print();
    p3 = p1 * p2;
    cout << "p1 * p2 = " ; p3.print();
    p3 = p1;
    cout << "p1 = p3 = " ; p3.print();
    p3 += p2;
    cout << "p3 += p2: " ; p3.print();
    p3 = p1;
    p3 -= p2;
    cout << "p3 -= p2: " ; p3.print();
    p3 = p1;
    cout << " Reset p1 = p3 " << endl;
    p3 *= p2;
    cout << "p3 *= p2: " ; p3.print();
    return 0;
}
```

Run program & result:



```
PS D:\VS CODE\C C++\CS360\HW#4> cd "d:\VS CODE\C C++\CS360\HW#4\" ; if ($?) { g++ Polynomia
Enter Polynomial p1:
Enter the number of terms: 3
Enter the coefficient and exponent: 4 3
Enter the coefficient and exponent: 2 5
Enter the coefficient and exponent: 8 1
Enter Polynomial p2:
Enter the number of terms: 4
Enter the coefficient and exponent: 3 5
Enter the coefficient and exponent: 7 4
Enter the coefficient and exponent: 9 2
Enter the coefficient and exponent: 5 0
p1 = (8x^1) + (4x^3) + (2x^5) +
p1 degree = 5
p2 = 5 + (9x^2) + (7x^4) + (3x^5) +
p2 degree = 5
p2 = 5 + (9x^2) + (7x^4) + (3x^5) +
p1 + p2 = 5 + (8x^1) + (9x^2) + (4x^3) + (7x^4) + (5x^5) +
p1 - p2 = -5 + (8x^1) + (-9x^2) + (4x^3) + (-7x^4) + (-1x^5) +
p1 * p2 = (40x^1) + (92x^3) + (102x^5) + (24x^6) + (46x^7) + (12x^8) + (14x^9) + (6x^10) +
p1 = p3 = (8x^1) + (4x^3) + (2x^5) +
p3 += p2: 5 + (8x^1) + (9x^2) + (4x^3) + (7x^4) + (5x^5) +
p3 -= p2: -5 + (8x^1) + (-9x^2) + (4x^3) + (-7x^4) + (-1x^5) +
 Reset p1 = p3
p3 *= p2: (8x^1) + (4x^3) + (2x^5) +
PS D:\VS CODE\C C++\CS360\HW#4> 
```