```
//Filename: poly.cpp
//Name: Tan Qi Hao
/*Synopsis:
This program let user input the coefficient and degree of 2 polynomial. Then,
output the 2 polynomial, their subtraction, their addition and determine whether or
not there are the same polynomial.

 */

#include <cmath>
#include <iostream>

using namespace std;

// Default size of our dynamic coefficient array
const int DEFAULTPOLY = 10;


// Do NOT modify the class header.
class Poly
{
private:

    // Data members
    int arraySize;    // size of array
    int *coeff;       // dynamic array

public:

    // Default Class constructor
    // Allocate an array of DEFAULTPOLY elements and initialize it to the constant
0
    // post: Class object is initialized to degree-0 polynomial of 0
    Poly();


    // Non-default (alternate) Class constructor
    // Allocate an array of 'size' elements and initializes it to the constant 0
    // post: Class object is initialized to degree-0 polynomial of 0
    Poly(int size);


    // Copy constructor
    // Construct a new Poly that is a copy of an existing Poly
    // post: Class object is initialized to be a copy of the argument Poly
    Poly(const Poly& aPoly);


    // Destructor
    // Destroy a poly object by freeing the dynamically allocated array
    ~Poly();


    // Assignment operator
    // Assign 'aPoly' Poly object to 'this' Poly object
    // Note: This function is provided, please do not modify it
    const Poly& operator=(const Poly& aPoly);
    // grow
    // This method will allow us to increase the size of the dynamically allocated
    // array by allocating a new array of the desired size, copying the data from
    // the old array to the new array, and then releasing the old array.
    // If the newSize is less than or equal to the current size, then no actions
    // are taken.
    // Note: the maximum degree of a polynomial is one less than the size of the
    // array. The parameter newSize represents the size of the array.
    void grow(int newSize);


    // degree
    // Finds the degree of a polynomial (the highest power with a non-zero
    // coefficient)
    // pre: Class object exists
    // post: Returns the degree of the polynomial object.
    int degree() const;


    // setCoeff
    // Sets a term, value*x^i, in a polynomial, growing the array if necessary.
    // pre: Class object has been initialized. i is a non-negative integer.
    // post: In the polynomial, the term with power i has coefficient
    //       value. The polynomical was grown if required.
    void setCoeff(int value, int i);


    // getCoeff
    // Finds the coefficient of the x^i term in poly
    // pre: Class object has been initialized. i is a non-negative integer.
    // post: Returns the value of the coefficient of the term with power i
    // note: If the object does not contain a term with power i (e.g.,
    //       i>=arraySize), a coefficient value of zero is returned.
    int getCoeff(int i) const;


    // negate
    // Negate a polynomial
    // pre: The class object has been initialized.
    // post: The polynomial has been changed to represent its
    //       multiplication by -1.
    void negate();


    // addition operator
    // Add two polynomials together and return a new polynomial that is the result
    // pre: Both class objects have been initialized
    // post: The sum of two polynomials is stored in a new polynomial which is
returned.
    //       The parameter polynomials are not changed.
    friend Poly operator+(const Poly& aPoly, const Poly& bPoly);


    // subtraction operator
    // Subtracts one polynomial from another and return a new polynomial that is
the result
    // pre: Both class objects have been initialized
    // post: The difference of two polynomials is stored in a new polynomial which
is returned.
```

```
    //       The parameter polynomials are not changed.
    friend Poly operator-(const Poly& aPoly, const Poly& bPoly);


    // equality operator
    // Compare two polynomials and return true if they are the same, false
otherwise
    // pre: Both class objects have been initialized
    // post: A boolean value indicating whether two polynomials are the same is
returned.
    //       The parameter polynomials are not changed.
    friend bool operator==(const Poly& aPoly, const Poly& bPoly);


    // insertion operator for output
    // Print polynomials
    // pre: The class object has been initialized
    // post: several values representing the polynomial are inserted into the
output stream
    friend ostream& operator<<(ostream& out, const Poly &aPoly);

};



int main(){

    Poly poly1, poly2;
    int numCoeff, coeffValue, coeffDegree, x;

    // prompt user for the number of coefficients
    cout << "How many coefficients for polynomial 1?" << endl;
    cin >> numCoeff;
    for (int i=0; i<numCoeff; ++i){
        cout << "Coefficient " << i+1 << " for polynomial 1:";
        cin >> coeffValue >> coeffDegree;
        poly1.setCoeff(coeffValue, coeffDegree);
    }

    cout << endl << "How many coefficients for polynomial 2?" << endl;
    cin >> numCoeff;
    for (int i=0; i<numCoeff; ++i){
        cout << "Coefficient " << i+1 << " for polynomial 2:";
        cin >> coeffValue >> coeffDegree;
        poly2.setCoeff(coeffValue, coeffDegree);
    }

    // Sample test cases for degree() and operator<<
    cout << endl << "Polynomial 1 = " << poly1 << endl;
    cout << "Polynomial 1 has degree " << poly1.degree() << endl;
    cout << "Polynomial 2 = " << poly2 << endl;
    cout << "Polynomial 2 has degree " << poly2.degree() << endl;

    // Sample test cases for operator+ and operator-
    cout << endl << "Polynomial 1 + Polynomial 2 = " << poly1 + poly2 << endl;
    cout << "Polynomial 1 - Polynomial 2 = " << poly1 - poly2 << endl << endl;

    // Sample test cases for operator==
    if (poly1==poly2)
```

```
        cout << "Two polynomials are the same." << endl;
    else
        cout << "Two polynomials are different." << endl;

    // Try more test cases to test your class thoroughly



    return 0;
}

// Do not modify this function
const Poly& Poly::operator= (const Poly& aPoly){

    if (this == &aPoly)
        return *this;

    if (coeff)
        delete [] coeff;

    arraySize = aPoly.arraySize;
    coeff = new int[arraySize];
    for (int i=0; i<arraySize; ++i){
        coeff[i] = aPoly.getCoeff(i);
    }

    return *this;
}

/* your code here */

//This is a default constructor
Poly::Poly(){

  coeff = new int[DEFAULTPOLY];

  for(int i = 0; i < DEFAULTPOLY; i++){

    coeff[i] = 0;

  }

}

//This constructor make another coeff array
Poly::Poly(int size){

  coeff = new int[size];

  for(int i = 0; i < size; i++){

    coeff[i] = 0;

  }
  arraySize = size;

}

//This constructor constructor construct a new poly called aPoly
```

```
Poly::Poly(const Poly& aPoly){

  coeff = new int[aPoly.arraySize];
  for(int i = 0; i < aPoly.arraySize; i++){

    coeff[i] = aPoly.coeff[i];

  }

  arraySize = aPoly.arraySize;

}

//Deconstructer to delete array coeff
Poly::~Poly(){

  delete[] coeff;

}

//This function grow the coeff array with newSize
void Poly::grow(int newSize){

  //Make a newCoeff[]
  if(newSize > arraySize){

    int *newCoeff = new int[newSize];
    for(int i = 0; i < newSize; i++){

      newCoeff[i] = 0;

    }

    for(int i = 0; i < arraySize; i++){

      newCoeff[i] = coeff[i];

    }

    //Copy the newCoeff[] to the coeff[]
    arraySize = newSize;
    coeff = new int[arraySize];
    for(int i = 0; i < arraySize; i++){

      coeff[i] = newCoeff[i];

    }

    delete [] newCoeff;

  }


}

//This function used to get the highest degree
int Poly::degree() const{

  int degree = 0;
  for(int i = 0; i < arraySize; i++){

    if(coeff[i] != 0){

      degree = i;

    }

  }

  return degree;


}

//This function set the coefficient of polynomial
void Poly::setCoeff(int value, int i){

  if(i + 2  > arraySize){

    grow(i + 2);

  }
  coeff[i] = coeff[i] + value;

}

//This function is used to get polynomial
int Poly::getCoeff(int i) const{
 if(i >= arraySize){

    return 0;

  }

 return coeff[i];


}

//This function negate the polynomial
void Poly::negate(){

  for(int i = 0; i < arraySize; i++){

    coeff[i] = coeff[i] * -1;

  }

}

//This function add up both polynomial
//Parameter aPoly is polynomial 1 and bPoly is polynomial 2
Poly operator+(const Poly& aPoly, const Poly& bPoly){

  Poly addPoly;

  if(aPoly.arraySize < bPoly.arraySize){
    addPoly.arraySize = aPoly.arraySize;
    addPoly.grow(bPoly.arraySize);

  }else{

    addPoly.arraySize = bPoly.arraySize;
    addPoly.grow(aPoly.arraySize);

  }


  for(int i = 0; i < addPoly.arraySize; i++){

    addPoly.coeff[i] = aPoly.coeff[i] + bPoly.coeff[i];

  }

  return addPoly;

}

//This function subtract both array
Poly operator-(const Poly& aPoly, const Poly& bPoly){
  Poly subPoly;


   if(aPoly.arraySize < bPoly.arraySize){

    subPoly.arraySize = aPoly.arraySize;
    subPoly.grow(bPoly.arraySize);

  }else{

    subPoly.arraySize = bPoly.arraySize;
    subPoly.grow(aPoly.arraySize);

  }


  for(int i = 0; i < subPoly.arraySize; i++){

    subPoly.coeff[i] = aPoly.coeff[i] - bPoly.coeff[i];

  }

  return subPoly;
}

//This operator function test both parameter array aPoly and bPoly are equal or not
bool operator==(const Poly& aPoly, const Poly& bPoly){

  bool equalPoly = true;

  if(aPoly.degree() == bPoly.degree()){
  for(int i = 0; i <= aPoly.arraySize; i++){

    if(aPoly.coeff[i] != bPoly.coeff[i]){

      equalPoly = false;
    }

  }

  }else{

    equalPoly = false;

  }

  return equalPoly;
}

//This operator function display polynomial
ostream& operator<<(ostream& out, const Poly &aPoly){
  //Find the coeff and degree of 1st polynomial
  int firstCoeff = 0;
  int firstDegree = 0;
  for(int i = 0; i < aPoly.arraySize; i++){

    if(aPoly.coeff[i] != 0){

      firstCoeff = aPoly.coeff[i];
      firstDegree = i;
    }

  }

  //output the polynomial
  int degree = aPoly.degree();
  for(int i = degree; i >= 1; i--){

    if(aPoly.coeff[i] != 0){

      //First number
      if(aPoly.coeff[i] == firstCoeff && firstDegree == i){
      out << aPoly.coeff[i] << "x^" << i;

      }else{

      //Positive poly after first number
      if(aPoly.coeff[i] > 0){
      out << "+" << aPoly.coeff[i] << "x^" << i;

      }
      //negative poly after first number
      if(aPoly.coeff[i] < 0){

      out << aPoly.coeff[i] << "x^" << i;
      }
      }
    }

  }

  //Zero power of poly
    if(aPoly.coeff[0] < 0){
    out << aPoly.coeff[0];
    }else{
      out << "+" << aPoly.coeff[0];

    }

  return out;
}
```