다항식 (Polynomial)

다항식(polynomial) 은 변수와 상수를 덧셈, 곱셈, 뺄셈 연산만을 이용하여 표현한 식을 말한다. 예를 들어 x^3+2x^2-5x+7 은 다항식이지만, $3x^2-5/x+7x^{2/3}$ 은 다항식이 아니다. 다항식 x^3+2x^2-5x+7 에서 "x^3", "$2x^2$", "$-5x$", "7"을 각각 항(term)이라고 부르고, 여러 개의 항으로 이루어졌기 때문에 다항식이라고 부른다. 한 개의 항, 즉 단항, cx^e 에서 c 를 상수(coefficient)라 하고, 음수가 아닌 정수 e 를 지수(exponent)라 한다.

한 개의 변수 x 와 $n+1$ 개의 상수 $c_0, c_1, \ldots, c_n$ 으로 다음과 같이 표현되는 식은 다항식이다.

$$P(x) = c_n x^n + c_{n-1}x^{n=1} + \cdots + c_1 x + c_0 \quad (c_n \neq 0)$$

위 식에서, 음이 아닌 정수 n 을 차수(degree)라고 부르며, 각 상수 $c_0, c_1, \ldots, c_n$ 을 다항식의 계수(coefficient)라고 부른다. 또한 $c_i x^i \ (0 \le i \le n)$ 를 i 차 항이라고 하고, 특히 0 차항 c_0 을 상수항이라고 한다. 0 차 다항식은 상수항 밖에 없는 다항식을 말한다.

다항식은 다음과 같은 특징을 가지고 있다.

(1) 다항식의 합(sum)은 다항식이다.
(2) 다항식의 곱(product)은 다항식이다.
(3) 다항식의 미분(derivative)은 다항식이다. 다항식 $c_n x^n + c_{n-1}x^{n=1} + \cdots + c_1 x + c_0$ 의 미분 다항식은 $nc_n x^{n-1} + (n-1)c_{n-1}x^{n=2} + \cdots + 2c_2 x + c_1$ 이다.

또한, 변수 x 에 특정한 값 a 을 지정하여 다항식의 값 $P(a) = c_n a^n + c_{n-1}a^{n=1} + \cdots + c_1 a + c_0$ 을 계산할 수 있다.

다항식의 모든 계수가 정수인 정수 다항식을 정의하고, 다항식에 적용 가능한 연산자를 구현한 C++ 클래스 "myPolynomial"을 작성하시오. 단, 정수 다항식 클래스는 다음과 같은 연산이 가능하도록 만들어 져야 한다. 단, 다항식(polynomial)을 구현한 "myPolynomial" 클래스는 항(term)을 구현한 "myTerm"클래스를 이용하여 구현하여야 한다. 계수가 정수인 "myTerm" 클래스는 다음과 같은 연산이 가능하도록 만들어 져야 한다.

<myTerm 클래스>

Constructors (생성자)

(1) myTerm(int c=0, unsigned e=0);
Default constructor 로서 단항(term) cx^e 를 만든다.

(2) myTerm (const myTerm &term);
Copy constructor 이다.

Accessor Functions (접근자), Mutator Functions(수정자)

(1) int getCoeff() const;

단항의 계수를 리턴한다.

(2) unsigned getExp() const;

단항의 지수를 리턴한다.

(3) void setCoeff(int c);

단항의 계수를 수정한다.

(4) void setExp(unsigned e);

단항의 지수를 수정한다.

Member Functions

(1) myTerm ddx() const;

단항 cx^e 의 미분인 ecx^{e-1} 을 리턴한다. 단 지수 e 가 0 인 경우는 상수 단항 0 를 리턴한다.

Overloaded Operators

myTerm 에서 가능한 연산자는 다음과 같다.

(1) 비교연산 (<, ==, !=)

두 개의 단항이 같은 단항인지를 판단한다. 또한 두 개의 단항을 비교하는 경우에는 지수가 큰 단항을 두 단항 중에서 작은 단항으로 판단한다.

(2) Unary -

단항의 계수의 부호를 반대로 한다.

(3) Output Opeartors (<<)

단항 cx^e 를 cx^e 형식으로 출력한다. 단 지수가 1 인 경우에는 지수를 출력하지 않으며, 지수가 0 인 경우에는 계수만 출력한다. 또한 상수인 경우를 제외하고 계수가 1 인 경우에는 계수를 출력하지 않는다. 각 문자들 사이에는 빈칸이 없다. 예를 들어 단항 $3x^2$ 는 3x^2 로, 단항 $-5x$ 는 -5x 로, 단항 7 은 7 로 출력한다.

<myPolynomial 클래스>

Constructors (생성자)

(1) myPolynomial (int c=0, unsigned e=0);

Default constructor 로서 단항(term) cx^e 으로 만들어진 다항식을 만든다.

(2) myPolynomial (int nTerms, int mono[]);

nTerms 개의 단항으로 만들어진 다항식을 만든다. 각 단항의 계수와 지수는 1 차원 배열 mono[] 에 각 단항의 계수와 지수 순서로 저장되어 있다. 예를 들어, nTerms 가 5 이고,

mono[10] = { 1, 0, -2, 1, 3, 2, -4, 3, 5, 4 }

인 경우에는 다항식 $5x^4 - 4x^3 + 3x^2 - 2x + 1$ 을 만든다.

(3) myPolynomial (const myPolynomial &poly);

Copy constructor 이다.

Accessor Functions (접근자), Mutator Functions(수정자)

(1) int getDegree() const;

다항식의 차수를 리턴한다.

(2) int getNumTerms() const;

다항식에서 단항의 개수를 리턴한다.

(3) void setCoeff(int c);

단항의 계수를 수정한다.

Member Functions

(1) myPolynomial ddx() const;

다항식의 미분 다항식을 리턴한다.

Overloaded Operators

myPolynomial 에서 가능한 연산자는 다음과 같다.

(1) 다항식끼리의 덧셈, 뺄셈, 곱셈연산 (+, -, *)

다항식+ 다항식, 다항식-다항식, 다항식*다항식 연산이 가능하다.

(2) 다항식과 정수와의 곱셈연산 (*)

다항식*정수, 정수*다항식 연산이 가능하다.

(3) 다항식 계산 연산자 (())

다항식의 변수 x 에 특정한 값 a 을 지정하여 다항식의 값 $p(a) = c_n a^n + c_{n-1} a^{n=1} + \cdots + c_1 a + c_0$ 을 계산하여 리턴한다.

(4) Unary -

모든 단항의 계수의 부호를 반대로 한다.

(5) Comparison Operators (==, !=)

두개의 다항식이 같은지를 비교할 수 있다.

(6) Assignment Operators (=, + =, -=, *=)

위 연산 중에서 *= 연산의 경우에는 다항식과 곱셈이 가능하며, 또한 정수와의 곱셈도 가능하다.

(7) Output Opeartors (<<)

다항식 $c_n x^n + c_{n-1} x^{n=1} + \cdots + c_1 x + c_0$ 를 각 단항을 지수가 큰 단항부터 상수항까지 순서대로 출력한다. 상수항을 제외하고 각 단항에서 계수가 1 인 경우에 그 계수는 출력하지 않는다. 각 출력되는 문자들 사이에는 빈칸이 없다. 예를 들어 다항식 $5x^4 - 4x^3 + 3x^2 - 2x + 1$ 는 5x^4-4x^3+ 3x^2=2x+ 1 로 출력한다.

다음 네 개의 파일 MyTerm.h, MyTerm.cpp, MyPolynomial.h, MyPolynomial.cpp 에 위에서 설명한 모든 연산자를 추가하여 클래스 myTerm, myPolynomial 를 완전히 구현하여 아래 테스트 프로그램인 TestMyPolynomial.cpp 가 정확하게 동작하도록 하시오. 단, 테스트 프로그램의 testDataFromFile() 함수는 입력 파일에서 다항식을 입력하여 계산하는 함수이다.

MyTerm.h

```
#ifndef _MYTERM_H_
#define _MYTERM_H_

#include <iostream>
using namespace std;

class myPolynomial;

class myTerm
{
public:
    myTerm(int c = 0, unsigned e = 0);

    // copy constructor
    myTerm(const myTerm &term);

    // overloaded operators
    bool operator == (const myTerm& term) const;
    bool operator != (const myTerm& term) const;
    bool operator < (const myTerm& term) const;
    myTerm operator - () const;

    // accessor functions
    int getCoeff() const { return coeff; }
    unsigned getExp() const { return exp; }

    // mutator functions
    void setCoeff(int c) { coeff = c; }
    void setExp(unsigned e) { exp = e; }

    // member functions
    myTerm ddx() const;        // derivative of a term

    // friend functions and classes
    friend ostream& operator <<(ostream &outStream, const myTerm& term);

    friend myPolynomial;

private:
    int coeff;      // integer coefficient
    unsigned exp;   // exponent (non-negative integer)
};

#endif _MYTERM_H_
```

MyTerm.cpp

```
#include "MyTerm.h"

// Constructor
myTerm::myTerm(int c, unsigned e) : coeff(c), exp(e)
{
}

// copy constructor
myTerm::myTerm(const myTerm &term) : coeff(term.coeff), exp(term.exp)
```

```
{
}

// overloaded operators
bool myTerm::operator == (const myTerm& term) const
{
}

// overloaded operators
bool myTerm::operator != (const myTerm& term) const
{
}

// overloaded operators
bool myTerm::operator < (const myTerm& term) const
{
    return exp > term.exp;
}

myTerm myTerm::operator - () const
{
}

// derivative of a term
myTerm myTerm::ddx() const
{
    if (exp == 0)
        return myTerm(0, 0);

    return myTerm(exp*coeff, exp-1);
}

// output operator
ostream& operator <<(ostream &outStream, const myTerm& term)
{
    if (term.exp == 0)
        if (term.coeff == 0)           // nothing to output
            return outStream;
        else
            return outStream << term.coeff;

    if (term.coeff == 1)
        outStream << "x";
    else if (term.coeff == -1)
        outStream << "-x";
    else
        outStream << term.coeff << "x";

    if (term.exp == 1)
        return outStream;
    else
        return outStream << "^" << term.exp;
}
```

MyPolynomial.h

```
#ifndef _MYPOLYNOMIAL_H_
#define _MYPOLYNOMIAL_H_

#include <iostream>
#include <list>
#include "MyTerm.h"

using namespace std;

class myPolynomial
{
```

```
public:
    myPolynomial(int c = 0, unsigned e = 0);
    myPolynomial(int nTerms, int mono[]);

    // copy constructor
    myPolynomial(const myPolynomial &poly);

    // overloaded operators
    bool operator == (const myPolynomial &poly) const;
    bool operator != (const myPolynomial &poly) const;
    myPolynomial& operator += (const myPolynomial &poly);
    myPolynomial& operator -= (const myPolynomial &poly);
    myPolynomial& operator *= (const myPolynomial &poly);
    myPolynomial& operator *= (int k);

    myPolynomial operator -() const;
    myPolynomial operator *(int k) const;
    myPolynomial operator +(const myPolynomial &poly) const;
    myPolynomial operator -(const myPolynomial &poly) const;
    myPolynomial operator *(const myPolynomial &poly) const;

    long operator() (int x) const; // evaluate the polynomial
     int getDegree() const;     // get a degree of the polynomial
    unsigned getNumTerms() const;      // number of terms in the polynomial
    myPolynomial ddx() const;          // derivative of a polynomial

    // friend operators and functions
    friend myPolynomial operator *(int k, const myPolynomial &poly);
    friend ostream& operator <<(ostream &outStream, const myPolynomial &poly);

    static const myPolynomial ZERO;   // P(x) = 0
    static const myPolynomial ONE;    // P(x) = 1
    static const myPolynomial X;           // P(x) = x

private:
    int degree;     // maximum expnent

    /********************* add your code here ********************************/

};

#endif _MYPOLYNOMIAL_H_
```

MyPolynomial.cpp

```
#include "MyPolynomial.h"

    /********************* add your code here ********************************/

// output operator
ostream& operator <<(ostream &outStream, const myPolynomial& poly)
{
    if (poly == myPolynomial::ZERO)
        return outStream << 0;

    /********************* add your code here ********************************/


    return outStream;
}

const myPolynomial myPolynomial::ZERO(0);      // the zero polynomial P(x) = 0
const myPolynomial myPolynomial::ONE(1, (unsigned)0); // the monomial P(x) = 1
const myPolynomial myPolynomial::X(1, 1);      // the monomial P(x) = x
```

TestMyPolynomial.cpp

```
#include <iostream>
#include "MyPolynomial.h"

void testSimpleCase();
void testDataFromFile();

void main(void)
{
    testSimpleCase();
     testDataFromFile();
}

void testSimpleCase()
{

    // test static variables
    cout << myPolynomial::ZERO << endl;
    cout << myPolynomial::ONE << endl;
    cout << myPolynomial::X << endl;

    myPolynomial p0, p1(1), p2(1, 1), p3(3, 5);

    int testData4[10] = {1, 0, 1, 1, 1, 2, 1, 3, 1, 4};
    int testData5[10] = {-1, 0, -1, 1, -1, 2, -1, 3, -1, 4};
    int testData6[10] = {1, 0, -1, 1, 1, 2, -1, 3, 1, 4};
    int testData7[10] = {2, 2, 5, 5, 4, 4, 1, 1, 3, 3};
    int testData8[12] = {1, 1000000000, 1, 100000000, 1, 1000000, 1, 10000, 1,
100, 1, 0};

    myPolynomial p4(5, testData4);
    myPolynomial p5(5, testData5);
    myPolynomial p6(5, testData6);
    myPolynomial p7(5, testData7);
    myPolynomial p8(6, testData8);
    myPolynomial p9(p7);

    // test constructor
    cout << p0 << endl << p1 << endl << p2 << endl;
    cout << p4 << endl << p8 << endl;

    // test copy constructor
    cout << p9 << endl;

    // test accessor function
    cout << p8.getDegree() << " " << p8.getNumTerms() << endl;

    // test evaluation function
    cout << p1(2) << " " << p2(2) << " " << p3(2) << " " << p4(2) << " " << endl;
    cout << p5(3) << " " << p6(3) << " " << p7(3) << " " << p9(3) << " " << endl;

    // test comparison operators
    cout << (p7 == p9) << " " << (p7 != p9) << endl;

    // test derivative function
    cout << myPolynomial::ZERO.ddx() << endl;
    cout << myPolynomial::ONE.ddx() << endl;
    cout << myPolynomial::X.ddx() << endl;
    cout << p4.ddx() << endl;
    cout << p8.ddx() << endl;

    // test unary operator -
    cout << -myPolynomial::ZERO << endl;
    cout << -p4 << endl;

  // test k*p(x) or p(x) * k
    cout << 3*myPolynomial::ZERO << endl;
    cout << 3*myPolynomial::ONE << endl;
    cout << myPolynomial::X*3 << endl;
```

```
    cout << 3*p4 << " " << p4*3 << endl;
    cout << 0*p5 << " " << p5*5 << endl;

    // test binary operator +
    cout << p4 + p5 << " " << p4 + p6 << endl;
    cout << p4 + p8 << " " << p8 + p8 << endl;

    // test binary operator -
    cout << p4 - p5 << " " << p4 - p6 << endl;
    cout << p4 - p8 << " " << p8 - p8 << endl;

    // test binary operator *
    cout << p4 * p5 << " " << p4 * p6 << endl;
    cout << p4 * p8 << " " << p8 * p8 << endl;

    myPolynomial tmp1(p4), tmp2, tmp3, tmp4;
    tmp4 = tmp3 = tmp2 = tmp1;

    cout << (tmp1 += p5) << endl;
    cout << (tmp2 -= p5) << endl;
    cout << (tmp3 *= p5) << endl;
    cout << (tmp4 *= 3) << endl;

    int t;
    cin >> t;
}


void testDataFromFile()
{
    int numTestCases;

    cin >> numTestCases;

    for (int i=0; i<numTestCases; i++)
    {
        int numTerms, terms[100];

        /* read first polynomial */
        cin >> numTerms;
        for (int j=0; j<numTerms; j++)
            cin >> terms[2*j] >> terms[2*j+1];

        myPolynomial p1(numTerms, terms);

        /* read second polynomial */
        cin >> numTerms;
        for (int j=0; j<numTerms; j++)
            cin >> terms[2*j] >> terms[2*j+1];

        myPolynomial p2(numTerms, terms);

        cout << p1 << endl << p2 << endl;
        cout << p1.getDegree() << " " << p2.getNumTerms() << endl;
        cout << p1.ddx() << endl << p2.ddx() << endl;
        cout << (p1 == p2) << " " << (p1 != p2) << endl;
        cout << p1(1) << endl;

        cout << -p1 + p1 * 2 * p2 - p2 * 2 + 3 * p1 << endl;

        myPolynomial p3(myPolynomial::ZERO), p4(myPolynomial::ONE),
p5(myPolynomial::X);
        p3 += p1;
        p4 -= p2;
        p5 *= p4;
        p5 *= 2;

        cout << p5 << endl;
```

```
    }
}
```

입력

입력은 표준입력(standard input)을 사용한다. 입력은 t 개의 테스트 케이스로 주어진다. 입력의 첫 번째 줄에 테스트 케이스의 개수를 나타내는 정수 t 가 주어진다. 두 번째 줄부터 두 줄에 한 개의 테스트 케이스에 해당하는 데이터가 입력된다. 각 줄에서 한 개의 다항식에 관한 데이터가 입력되는데, 입력되는 데이터는 다항식의 모든 항의 계수와 지수이다. 각 줄에서 첫 번째로 입력되는 정수 n ($1 \le n \le 50$)은 다항식의 항의 개수를 나타낸다. 그 다음으로는 2n 개의 정수가 입력되는데, 매 두 개의 정수는 각 항의 계수와 지수를 나타낸다. 계수가 0 인 항은 입력되지 않으며, 지수가 음수인 경우는 없다. 각 정수들 사이에는 한 개의 공백이 있다. 잘못된 데이터가 입력되는 경우는 없다.

출력

출력은 표준출력(standard output)을 사용한다. 입력되는 테스트 케이스의 순서대로 다음 줄에 이어서 각 테스트 케이스의 결과를 출력한다. 출력은 테스트 프로그램에 따른다.

입력과 출력의 예

입력
3 1 0 0 1 1 1000000000 5 1 0 1 1 1 2 1 3 1 4 5 1 4 1 3 1 2 1 1 1 0 5 1 0 1 1 1 2 1 3 1 4 5 -1 4 -1 3 -1 2 -1 1 -1 0

출력

```
0
1
x
0
1
x
x^4+ x^3+ x^2+ x+ 1
x^1000000000+ x^100000000+ x^1000000+ x^10000+ x^100+ 1
5x^5+ 4x^4+ 3x^3+ 2x^2+ x
1000000000 6
1 2 96 31
-121 61 1641 1641
1 0
0
0
1
4x^3+ 3x^2+ 2x+ 1
1000000000x^999999999+ 100000000x^99999999+ 1000000x^999999+ 10000x^9999+ 100x^99
0
-x^4-x^3-x^2-x-1
0
3
3x
3x^4+ 3x^3+ 3x^2+ 3x+ 3 3x^4+ 3x^3+ 3x^2+ 3x+ 3
0 -5x^4-5x^3-5x^2-5x-5
0
-5x^5-3x^4-4x^3-x^2-2x+ 1
5x^1000000005+ 4x^1000000004+ 3x^1000000003+ 2x^1000000002+ x^1000000001+ 5x^100000005+ 4x^10
0000004+ 3x^100000003+ 2x^100000002+ x^100000001+ 5x^1000005+ 4x^1000004+ 3x^1000003+ 2x^10000
02+ x^1000001+ 5x^10005+ 4x^10004+ 3x^10003+ 2x^10002+ x^10001+ 5x^105+ 4x^104+ 3x^103+ 2x^102+ x^
101+ 5x^5+ 4x^4+ 3x^3+ 2x^2+ x
0
2x^4+ 2x^3+ 2x^2+ 2x+ 2
-x^8-2x^7-3x^6-4x^5-5x^4-4x^3-3x^2-2x-1
3x^4+ 3x^3+ 3x^2+ 3x+ 3
0
x^1000000000
0 1
0
1000000000x^999999999
0 1
0
-2x^1000000000
-2x^1000000001+ 2x
x^4+ x^3+ x^2+ x+ 1
x^4+ x^3+ x^2+ x+ 1
4 5
4x^3+ 3x^2+ 2x+ 1
4x^3+ 3x^2+ 2x+ 1
1 0
5
2x^8+ 4x^7+ 6x^6+ 8x^5+ 10x^4+ 8x^3+ 6x^2+ 4x+ 2
-2x^5-2x^4-2x^3-2x^2
x^4+ x^3+ x^2+ x+ 1
-x^4-x^3-x^2-x-1
4 5
4x^3+ 3x^2+ 2x+ 1
-4x^3-3x^2-2x-1
0 1
5
-2x^8-4x^7-6x^6-8x^5-6x^4-4x^3-2x^2+ 2
2x^5+ 2x^4+ 2x^3+ 2x^2+ 4x
```