#include <deque>

#include <vector>

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

class DividedByZeroException {};

class BigInteger

{

private:

vector<char> digits;

bool sign; // true for positive, false for negitive

void trim(); // remove zeros in tail, but if the value is 0, keep only one:)

public:

BigInteger(int); // construct with a int integer

BigInteger(string&) ;

BigInteger();

BigInteger (const BigInteger&);

BigInteger operator=(const BigInteger& op2);

BigInteger abs() const;

BigInteger pow(int a);

//binary operators

friend BigInteger operator+=(BigInteger&,const BigInteger&);

friend BigInteger operator-=(BigInteger&,const BigInteger&);

friend BigInteger operator\*=(BigInteger&,const BigInteger&);

friend BigInteger operator/=(BigInteger&,const BigInteger&) throw(DividedByZeroException);

friend BigInteger operator%=(BigInteger&,const BigInteger&) throw(DividedByZeroException);

friend BigInteger operator+(const BigInteger&,const BigInteger&);

friend BigInteger operator-(const BigInteger&,const BigInteger&);

friend BigInteger operator\*(const BigInteger&,const BigInteger&);

friend BigInteger operator/(const BigInteger&,const BigInteger&) throw(DividedByZeroException);

friend BigInteger operator%(const BigInteger&,const BigInteger&) throw(DividedByZeroException);

//uniary operators

friend BigInteger operator-(const BigInteger&); //negative

friend BigInteger operator++(BigInteger&); //++v

friend BigInteger operator++(BigInteger&,int); //v++

friend BigInteger operator--(BigInteger&); //--v

friend BigInteger operator--(BigInteger&,int); //v--

friend bool operator>(const BigInteger&,const BigInteger&);

friend bool operator<(const BigInteger&,const BigInteger&);

friend bool operator==(const BigInteger&,const BigInteger&);

friend bool operator!=(const BigInteger&,const BigInteger&);

friend bool operator>=(const BigInteger&,const BigInteger&);

friend bool operator<=(const BigInteger&,const BigInteger&);

friend ostream& operator<<(ostream&,const BigInteger&); //print the BigInteger

friend istream& operator>>(istream&, BigInteger&); // input the BigInteger

public:

static const BigInteger ZERO;

static const BigInteger ONE;

static const BigInteger TEN;

};

// BigInteger.cpp

const BigInteger BigInteger::ZERO=BigInteger(0);

const BigInteger BigInteger::ONE =BigInteger(1);

const BigInteger BigInteger::TEN =BigInteger(10);

BigInteger::BigInteger()

{

sign=true;

}

BigInteger::BigInteger(int val){// construct with a int integer

if (val >= 0)

sign = true;

else{

sign = false;

val \*= (-1);

}

do{

digits.push\_back( (char)(val%10) );

val /= 10;

} while ( val != 0 );

}

BigInteger::BigInteger(string& def){

sign=true;

for ( string::reverse\_iterator iter = def.rbegin() ; iter < def.rend(); iter++){

char ch = (\*iter);

if (iter == def.rend()-1){

if ( ch == '+' )

break;

if(ch == '-' ){

sign = false;

break;

}

}

digits.push\_back( (char)((\*iter) - '0' ) );

}

trim();

}

void BigInteger::trim(){

vector<char>::reverse\_iterator iter = digits.rbegin();

while(!digits.empty() && (\*iter) == 0){

digits.pop\_back();

iter=digits.rbegin();

}

if( digits.size()==0 ){

sign = true;

digits.push\_back(0);

}

}

BigInteger::BigInteger(const BigInteger& op2){

sign = op2.sign;

digits=op2.digits;

}

BigInteger BigInteger::operator=(const BigInteger& op2){

digits = op2.digits;

sign = op2.sign;

return (\*this);

}

BigInteger BigInteger::abs() const {

if(sign) return \*this;

else return -(\*this);

}

BigInteger BigInteger::pow(int a)

{

BigInteger res(1);

for(int i=0; i<a; i++)

res\*=(\*this);

return res;

}

//binary operators

BigInteger operator+=(BigInteger& op1,const BigInteger& op2){

if( op1.sign == op2.sign ){ //只处理相同的符号的情况，异号的情况给-处理

vector<char>::iterator iter1;

vector<char>::const\_iterator iter2;

iter1 = op1.digits.begin();

iter2 = op2.digits.begin();

char to\_add = 0; //进位

while ( iter1 != op1.digits.end() && iter2 != op2.digits.end()){

(\*iter1) = (\*iter1) + (\*iter2) + to\_add;

to\_add = ((\*iter1) > 9); // 大于9进一位

(\*iter1) = (\*iter1) % 10;

iter1++; iter2++;

}

while ( iter1 != op1.digits.end() ){ //

(\*iter1) = (\*iter1) + to\_add;

to\_add = ( (\*iter1) > 9 );

(\*iter1) %= 10;

iter1++;

}

while ( iter2 != op2.digits.end() ){

char val = (\*iter2) + to\_add;

to\_add = (val > 9) ;

val %= 10;

op1.digits.push\_back(val);

iter2++;

}

if( to\_add != 0 )

op1.digits.push\_back(to\_add);

return op1;

}

else{

if (op1.sign)

return op1 -= (-op2);

else

return op1= op2 - (-op1);

}

}

BigInteger operator-=(BigInteger& op1,const BigInteger& op2){

if( op1.sign == op2.sign ){ //只处理相同的符号的情况，异号的情况给+处理

if(op1.sign) {

if(op1 < op2) // 2 - 3

return op1=-(op2 - op1);

}

else {

if(-op1 > -op2) // (-3)-(-2) = -(3 - 2)

return op1=-((-op1)-(-op2));

else // (-2)-(-3) = 3 - 2

return op1= (-op2) - (-op1);

}

vector<char>::iterator iter1;

vector<char>::const\_iterator iter2;

iter1 = op1.digits.begin();

iter2 = op2.digits.begin();

char to\_substract = 0; //借位

while ( iter1 != op1.digits.end() && iter2 != op2.digits.end()){

(\*iter1) = (\*iter1) - (\*iter2) - to\_substract;

to\_substract = 0;

if( (\*iter1) < 0 ){

to\_substract=1;

(\*iter1) += 10;

}

iter1++;

iter2++;

}

while ( iter1 != op1.digits.end() ){

(\*iter1) = (\*iter1) - to\_substract;

to\_substract = 0;

if( (\*iter1) < 0 ){

to\_substract=1;

(\*iter1) += 10;

}

else break;

iter1++;

}

op1.trim();

return op1;

}

else{

if (op1 > BigInteger::ZERO)

return op1 += (-op2);

else

return op1 = -(op2 + (-op1));

}

}

BigInteger operator\*=(BigInteger& op1,const BigInteger& op2){

BigInteger result(0);

if (op1 == BigInteger::ZERO || op2==BigInteger::ZERO)

result = BigInteger::ZERO;

else{

vector<char>::const\_iterator iter2 = op2.digits.begin();

while( iter2 != op2.digits.end() ){

if(\*iter2 != 0){

deque<char> temp(op1.digits.begin() , op1.digits.end());

char to\_add = 0;

deque<char>::iterator iter1 = temp.begin();

while( iter1 != temp.end() ){

(\*iter1) \*= (\*iter2);

(\*iter1) += to\_add;

to\_add = (\*iter1) / 10;

(\*iter1) %= 10;

iter1++;

}

if( to\_add != 0)

temp.push\_back( to\_add );

int num\_of\_zeros = iter2 - op2.digits.begin();

while( num\_of\_zeros--)

temp.push\_front(0);

BigInteger temp2;

temp2.digits.insert( temp2.digits.end() , temp.begin() , temp.end() );

temp2.trim();

result = result + temp2;

}

iter2++;

}

result.sign = ( (op1.sign && op2.sign) || (!op1.sign && !op2.sign) );

}

op1 = result;

return op1;

}

BigInteger operator/=(BigInteger& op1 , const BigInteger& op2 ) throw(DividedByZeroException) {

if( op2 == BigInteger::ZERO )

throw DividedByZeroException();

BigInteger t1 = op1.abs(), t2 = op2.abs();

if ( t1 < t2 ){

op1 = BigInteger::ZERO;

return op1;

}

//现在 t1 > t2 > 0

//只需将 t1/t2的结果交给result就可以了

deque<char> temp;

vector<char>::reverse\_iterator iter = t1.digits.rbegin();

BigInteger temp2(0);

while( iter != t1.digits.rend() ){

temp2 = temp2 \* BigInteger::TEN + BigInteger( (int)(\*iter) );

char s = 0;

while( temp2 >= t2 ){

temp2 = temp2 - t2;

s = s + 1;

}

temp.push\_front( s );

iter++;

}

op1.digits.clear();

op1.digits.insert( op1.digits.end() , temp.begin() , temp.end() );

op1.trim();

op1.sign = ( (op1.sign && op2.sign) || (!op1.sign && !op2.sign) );

return op1;

}

BigInteger operator%=(BigInteger& op1,const BigInteger& op2) throw(DividedByZeroException) {

return op1 -= ((op1 / op2)\*op2);

}

BigInteger operator+(const BigInteger& op1,const BigInteger& op2){

BigInteger temp(op1);

temp += op2;

return temp;

}

BigInteger operator-(const BigInteger& op1,const BigInteger& op2){

BigInteger temp(op1);

temp -= op2;

return temp;

}

BigInteger operator\*(const BigInteger& op1,const BigInteger& op2){

BigInteger temp(op1);

temp \*= op2;

return temp;

}

BigInteger operator/(const BigInteger& op1,const BigInteger& op2) throw(DividedByZeroException) {

BigInteger temp(op1);

temp /= op2;

return temp;

}

BigInteger operator%(const BigInteger& op1,const BigInteger& op2) throw(DividedByZeroException) {

BigInteger temp(op1);

temp %= op2;

return temp;

}

//uniary operators

BigInteger operator-(const BigInteger& op){ //negative

BigInteger temp = BigInteger(op);

temp.sign = !temp.sign;

return temp;

}

BigInteger operator++(BigInteger& op){ //++v

op += BigInteger::ONE;

return op;

}

BigInteger operator++(BigInteger& op,int x){ //v++

BigInteger temp(op);

++op;

return temp;

}

BigInteger operator--(BigInteger& op){ //--v

op -= BigInteger::ONE;

return op;

}

BigInteger operator--(BigInteger& op,int x){ //v--

BigInteger temp(op);

--op;

return temp;

}

bool operator<(const BigInteger& op1,const BigInteger& op2){

if( op1.sign != op2.sign )

return !op1.sign;

else{

if(op1.digits.size() != op2.digits.size())

return (op1.sign && op1.digits.size()<op2.digits.size())

|| (!op1.sign && op1.digits.size()>op2.digits.size());

vector<char>::const\_reverse\_iterator iter1,iter2;

iter1 = op1.digits.rbegin();iter2 = op2.digits.rbegin();

while( iter1 != op1.digits.rend() ){

if( op1.sign && \*iter1 < \*iter2 ) return true;

if( op1.sign && \*iter1 > \*iter2 ) return false;

if( !op1.sign && \*iter1 > \*iter2 ) return true;

if( !op1.sign && \*iter1 < \*iter2 ) return false;

iter1++;

iter2++;

}

return false;

}

}

bool operator==(const BigInteger& op1,const BigInteger& op2){

if( op1.sign != op2.sign || op1.digits.size() != op2.digits.size() )

return false;

vector<char>::const\_iterator iter1,iter2;

iter1 = op1.digits.begin();

iter2 = op2.digits.begin();

while( iter1!= op1.digits.end() ){

if( \*iter1 != \*iter2 ) return false;

iter1++;

iter2++;

}

return true;

}

bool operator!=(const BigInteger& op1,const BigInteger& op2){

return !(op1==op2);

}

bool operator>=(const BigInteger& op1,const BigInteger& op2){

return (op1>op2) || (op1==op2);

}

bool operator<=(const BigInteger& op1,const BigInteger& op2){

return (op1<op2) || (op1==op2);

}

bool operator>(const BigInteger& op1,const BigInteger& op2){

return !(op1<=op2);

}

ostream& operator<<(ostream& stream,const BigInteger& val){ //print the BigInteger

if (!val.sign)

stream << "-";

for ( vector<char>::const\_reverse\_iterator iter = val.digits.rbegin(); iter != val.digits.rend() ; iter++)

stream << (char)((\*iter) + '0');

return stream;

}

istream& operator>>(istream& stream, BigInteger& val){ //Input the BigInteger

string str;

stream >> str;

val=BigInteger(str);

return stream;

}

////////////////////////////////////////////////////////////

int main()

{

int n;

cin>>n;

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

BigInteger A;

BigInteger B;

BigInteger C=888;

cin>>A>>B;

cout<<"A-B:"<<A-B<<endl;

cout<<"A+B:"<<A+B<<endl;

cout<<"A\*B:"<<A\*B<<endl;

cout<<"A/B:"<<A/B<<endl;

cout<<"A%B:"<<A%B<<endl;

cout<<"A.pow(5)"<<A.pow(5)<<endl;

A++;

cout<<"A++:"<<A<<endl;

A--;

cout<<"A--:"<<A<<endl;

cout<<"++B:"<<++B<<endl;

cout<<"--B:"<<--B<<endl;

cout<<"C:"<<C<<endl;

}

return 0;

}