class BigInt {

private:

string Bnum;

short sign;

public:

BigInt(string str = "0") {

Bnum = str;

prepare();

}

BigInt(int x) {

ostringstream s;

s << x;

Bnum = s.str();

prepare();

}

void operator=(string number) {

Bnum = number;

prepare();

}

//used for delete heading zero and taking sign

void prepare() {

int i = 0;

Bnum[0] == '-' ? i++, sign = -1 : sign = 1;

Bnum = Bnum.substr(i, Bnum.size() - i);

headzero();

}

void headzero() {

int i = 0;

while(Bnum[i] == '0')

i++;

Bnum = Bnum.substr(i, Bnum.size() - i);

if(!Bnum.size()) {

Bnum = "0";

sign = 1;

}

}

void signchange() {

sign \*= -1;

}

//printout of big integer

friend ostream &operator<<(ostream &output, const BigInt &number) {

if(number.sign == -1)

output << "-";

return output << number.Bnum, output;

}

//taking input of big integer

friend istream &operator>>(istream &input, BigInt &number) {

return input >> number.Bnum, number.prepare(), input;

}

//less operator declared

bool operator<(const BigInt &number) const;

//less equal operator declared

bool operator<=(const BigInt &number) const;

//equal operator used

bool operator==(const BigInt &number) const {

return Bnum == number.Bnum && sign == sign;

}

bool operator!=(const BigInt &number) const {

return Bnum != number.Bnum || sign != sign;

}

//greater equal operator declared

bool operator>=(const BigInt &number) const;

//greater operator declared

bool operator>(const BigInt &numberconst) const;

//all arithmetic operator

BigInt operator+(BigInt number);

BigInt operator-(BigInt number);

BigInt operator/(BigInt number);

BigInt operator%(BigInt number);

BigInt operator\*(BigInt number);

//increment ++x

BigInt operator++();

//increment x++

BigInt operator++(int);

//decrement --x

BigInt operator--();

//decrement x++

BigInt operator--(int);

//add to own overloading

BigInt operator+=(const BigInt number);

//substract from own overloding

BigInt operator-=(const BigInt number);

//divide from own

BigInt operator/=(const BigInt number);

//multiple by own

BigInt operator\*=(const BigInt number);

//vagsesh

BigInt operator%=(const BigInt number);

//minus

BigInt operator-();

};

//return negative of this

BigInt BigInt::operator-() {

signchange();

return \*this;

}

//multiple by own

BigInt BigInt::operator\*=(const BigInt number) {

\*this = \*this \* number;

return (\*this);

}

//vagsesh by own

BigInt BigInt::operator%=(const BigInt number) {

\*this = \*this % number;

return (\*this);

}

//divide by own

BigInt BigInt::operator/=(const BigInt number) {

\*this = \*this / number;

return (\*this);

}

//substract from own

BigInt BigInt::operator-=(const BigInt number) {

\*this = \*this - number;

return (\*this);

}

//add to own

BigInt BigInt::operator+=(const BigInt number) {

\*this = \*this + number;

return (\*this);

}

// increment

BigInt BigInt::operator++(int) {

BigInt nn;

nn = \*this;

if(Bnum[Bnum.size() - 1] != '9' and sign == 1)

Bnum[Bnum.size() - 1]++;

else if(Bnum[Bnum.size() - 1] != '0' and sign == -1)

Bnum[Bnum.size() - 1]--, headzero();

else {

BigInt number("1");

\*this = \*this + number;

}

return nn;

}

BigInt BigInt::operator++() {

if(Bnum[Bnum.size() - 1] != '9' and sign == 1)

Bnum[Bnum.size() - 1]++;

else if(Bnum[Bnum.size() - 1] != '0' and sign == -1)

Bnum[Bnum.size() - 1]--, headzero();

else {

BigInt number("1");

\*this = \*this + number;

}

return (\*this);

}

//decrement

BigInt BigInt::operator--(int) {

BigInt nn;

nn = \*this;

if(Bnum[Bnum.size() - 1] != '0' and sign == 1) {

Bnum[Bnum.size() - 1]--, headzero();

} else if(Bnum[Bnum.size() - 1] != '9' and sign == -1) {

Bnum[Bnum.size() - 1]++;

} else {

BigInt number("1");

\*this = \*this - number;

}

return nn;

}

BigInt BigInt::operator--() {

if(Bnum[Bnum.size() - 1] != '0' and sign == 1) {

Bnum[Bnum.size() - 1]--, headzero();

} else if(Bnum[Bnum.size() - 1] != '9' and sign == -1) {

Bnum[Bnum.size() - 1]++;

} else {

BigInt number("1");

\*this = \*this - number;

}

return (\*this);

}

//multipication

BigInt BigInt ::operator\*(BigInt number) {

BigInt ans;

ans.sign = sign \* number.sign;

int l = Bnum.size() + number.Bnum.size();

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

ans.Bnum += '0';

}

int carry = 0;

int k, x = 0;

for(int i = Bnum.size() - 1; i >= 0; i--) {

k = x;

for(int j = number.Bnum.size() - 1; j >= 0; j--) {

carry += (Bnum[i] - '0') \* (number.Bnum[j] - '0') + (ans.Bnum[k] - '0');

ans.Bnum[k++] = carry % 10 + '0';

carry /= 10;

}

while(carry) {

ans.Bnum[k++] = carry % 10 + '0';

carry /= 10;

}

x++;

}

reverse(ans.Bnum.begin(), ans.Bnum.end());

ans.headzero();

return ans;

}

//vagsesh

BigInt BigInt ::operator%(BigInt number) {

if(number.Bnum == "0")

number.Bnum[0] /= (number.Bnum[0] - 48);

BigInt db;

db.Bnum = "";

short s = number.sign \* sign;

number.sign = db.sign = 1;

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

db.Bnum += Bnum.substr(i, 1);

db.headzero();

while(db >= number)

db = db - number;

}

db.sign = s;

db.headzero();

return db;

}

//divition

BigInt BigInt ::operator/(BigInt number) {

if(number.Bnum == "0")

number.Bnum[0] /= (number.Bnum[0] - 48);

BigInt ans, db;

ans.sign = number.sign \* sign;

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

ans.Bnum += '0';

}

db.Bnum = "";

number.sign = db.sign = 1;

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

db.Bnum += Bnum.substr(i, 1);

db.headzero();

while(db >= number) {

db = db - number;

ans.Bnum[i]++;

}

}

ans.headzero();

ans.Bnum = ans.Bnum.substr(0, ans.Bnum.size() - 1);

if(!ans.Bnum.size())

ans.Bnum = '0';

return ans;

}

//substraction

BigInt BigInt ::operator-(BigInt number) {

if(sign != number.sign) {

number.signchange();

return (\*this) + number;

}

string str1 = Bnum, str2 = number.Bnum;

BigInt ans;

ans.sign = sign;

short s = sign;

sign = number.sign = 1;

if(\*this < number) {

swap(str1, str2);

ans.signchange();

}

ans.Bnum = "";

int l1 = str1.size() - 1;

int l2 = str2.size() - 1;

int l = max(l1, l2) + 1;

int carry = 0;

int sub = 0;

while(l--) {

sub = (l1 >= 0 ? str1[l1] - '0' : 0) - (l2 >= 0 ? str2[l2] - '0' : 0) - carry;

sub < 0 ? sub = sub + 10, carry = 1 : carry = 0;

ans.Bnum += sub + '0';

l1--, l2--;

}

reverse(ans.Bnum.begin(), ans.Bnum.end());

sign = s;

ans.headzero();

return ans;

}

//addition

BigInt BigInt ::operator+(BigInt number) {

if(sign != number.sign) {

number.signchange();

return (\*this) - number;

}

BigInt ans;

ans.Bnum = "";

int l1 = Bnum.size() - 1;

int l2 = number.Bnum.size() - 1;

int l = max(l1, l2) + 1;

int carry = 0;

while(l--) {

carry += (l1 >= 0 ? Bnum[l1] - '0' : 0) + (l2 >= 0 ? number.Bnum[l2] - '0' : 0);

ans.Bnum += carry % 10 + '0';

carry /= 10;

l1--;

l2--;

}

if(carry) {

ans.Bnum += carry + '0';

}

ans.sign = sign;

reverse(ans.Bnum.begin(), ans.Bnum.end());

return ans;

}

// < lesser operator used

bool BigInt::operator<(const BigInt &number) const {

if(sign != number.sign)

return sign < number.sign;

if(Bnum.size() != number.Bnum.size())

return sign == 1 ? Bnum.size() < number.Bnum.size() : Bnum.size() > number.Bnum.size();

int i = 0;

while(i < Bnum.size()) {

if(Bnum[i] != number.Bnum[i]) {

return sign == 1 ? Bnum[i] < number.Bnum[i] : Bnum[i] > number.Bnum[i];

}

i++;

}

return false;

}

// <= operator used

bool BigInt::operator<=(const BigInt &number) const {

if(sign == number.sign and Bnum == number.Bnum)

return true;

if(sign != number.sign)

return sign < number.sign;

if(Bnum.size() != number.Bnum.size())

return sign == 1 ? Bnum.size() < number.Bnum.size() : Bnum.size() > number.Bnum.size();

int i = 0;

while(i < Bnum.size()) {

if(Bnum[i] != number.Bnum[i])

return sign == 1 ? Bnum[i] < number.Bnum[i] : Bnum[i] > number.Bnum[i];

i++;

}

return false;

}

//greater equal operator process

bool BigInt::operator>=(const BigInt &number) const {

if(sign == number.sign and Bnum == number.Bnum)

return true;

if(sign != number.sign)

return sign > number.sign;

if(Bnum.size() != number.Bnum.size())

return sign == 1 ? Bnum.size() > number.Bnum.size() : Bnum.size() < number.Bnum.size();

int i = 0;

while(i < Bnum.size()) {

if(Bnum[i] != number.Bnum[i])

return sign == 1 ? Bnum[i] > number.Bnum[i] : Bnum[i] < number.Bnum[i];

i++;

}

return false;

}

//greater operator process

bool BigInt::operator>(const BigInt &number) const {

if(sign != number.sign)

return sign > number.sign;

if(Bnum.size() != number.Bnum.size())

return sign == 1 ? Bnum.size() > number.Bnum.size() : Bnum.size() < number.Bnum.size();

int i = 0;

while(i < Bnum.size()) {

if(Bnum[i] != number.Bnum[i])

return sign == 1 ? Bnum[i] > number.Bnum[i] : Bnum[i] < number.Bnum[i];

i++;

}

return false;

}

int main() {

//可用运算符: + - \* / % 比大小

BigInt a(123), b("456");

cout << a + b << endl;

cout << a - b << endl;

cout << a \* b << endl;

cout << a / b << endl;

cout << a % b << endl;

}