#define \_CRT\_SECURE\_NO\_WARNINGS

//

// Created by LeeKi on 2023/3/10.

//

//Copyright [2023] <Copyright leeki>

#include <cmath>

#include <iostream>

#include <cctype>

#include <cstring>

using namespace std;

class TeamWorkBigInt

{

public:

static const int digits = 1000; //最大位数

//explicit关键字 指定构造函数为显式

TeamWorkBigInt() {

sign = 0;

for (int i = 0; i < 1000; i++) {

integer[i] = 0;

}

}

explicit TeamWorkBigInt(int a ); //参数为整数的构造函数

explicit TeamWorkBigInt(const char\*); //参数为字符串的构造函数

TeamWorkBigInt(const TeamWorkBigInt& a);

int getLength() const;

bool isNegative() const;

void operator=(const int);

void operator=(const TeamWorkBigInt);

void operator-();

TeamWorkBigInt operator+(const TeamWorkBigInt&) const;

TeamWorkBigInt operator+(const int) const;

TeamWorkBigInt operator+(const char\*) const;

TeamWorkBigInt operator-(const TeamWorkBigInt&) const;

TeamWorkBigInt operator-(const int) const;

TeamWorkBigInt operator\*(const TeamWorkBigInt&) const;

TeamWorkBigInt operator\*(const int) const;

TeamWorkBigInt operator/(const TeamWorkBigInt&) const;

TeamWorkBigInt operator/(const int) const;

void operator+=(const TeamWorkBigInt& a);

void operator-=(const TeamWorkBigInt& a);

void operator/=(const int) ;

void operator/=(const TeamWorkBigInt&);

bool operator>(const TeamWorkBigInt&) const;

bool operator<(const TeamWorkBigInt&) const;

bool operator>=(const TeamWorkBigInt&) const;

bool operator<=(const TeamWorkBigInt&) const;

bool operator==(const TeamWorkBigInt&) const;

bool operator!=(const TeamWorkBigInt&) const;

//友元函数

friend TeamWorkBigInt operator<<(const TeamWorkBigInt& a, int b);

friend TeamWorkBigInt pow(TeamWorkBigInt, int);

friend ostream& operator<<(ostream&, const TeamWorkBigInt&); //输出操作符重载

private:

int integer[digits]; //大数位数

int sign;

};

//-------------------------构造函数--------------------------------

//参数为整数的构造函数

TeamWorkBigInt::TeamWorkBigInt(int a) {

if (a < 0)

sign = -1, a = -a;

//else if (a == 0)

// sign = 0;

else

sign = 1;

int i = 0;

int temp = a;

//判断位数

while (temp) {

temp /= 10;

i++;

}

//大数倒序依次存入数组 例：4742938566342存入数组后为[2,4,3,6,6,5,8,3,9,2,4,7,4,0,0,0,...]

//输出的时候需要跳过前导0，倒序输出

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

integer[m] = a % 10;

a /= 10;

}

//空位补0

for (int j = i; j < digits; j++) {

integer[j] = 0;

}

}

//参数为字符串的构造函数

TeamWorkBigInt::TeamWorkBigInt(const char\* a) {

int i = 0;

while (\*a != '\0') {

a++;

i++;

}

a--;//此时i记录位数，指针a指向字符串最后一位

if (a == "-")

sign = -1;

else

sign = 1;

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

integer[m] = (\*a - '0');

a--;

}//大数倒序依次存入数组

for (int j = i; j < digits; j++) {

integer[j] = 0;

}//空位补0

}

TeamWorkBigInt::TeamWorkBigInt(const TeamWorkBigInt& a) {

sign = a.sign;

for (int m = 0; m < digits; m++) {

integer[m] = a.integer[m];

}

}

//-------------------------构造函数--------------------------------

//=重载

void TeamWorkBigInt::operator=(const int a)

{

TeamWorkBigInt c(a);

//memcpy(integer, c.integer,c.digits);

int count = 0;

for (int i = 0; i < digits - 1; i++)

{

integer[i] = c.integer[i];

}

}

void TeamWorkBigInt::operator=(const TeamWorkBigInt a)

{

//TeamWorkBigInt c(a);

//memcpy(integer, c.integer,c.digits);

int count = 0;

for (int i = 0; i < digits - 1; i++)

{

integer[i] = a.integer[i];

}

}

//输出操作符重载

ostream& operator<<(ostream& out, const TeamWorkBigInt& a) {

int l = a.getLength();

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

out << a.integer[a.getLength() - 1 - i];

}

return out;

}

//返回数位长度

int TeamWorkBigInt::getLength() const {

int count = 0;

for (int i = digits-1; i > 0; i--) {

if (integer[i] != 0) {

break;

}

else {

count++;

}

}

return digits - count;

}

//判断是否为负值

bool TeamWorkBigInt::isNegative() const {

return (sign == -1);

}

void TeamWorkBigInt::operator-(){

sign \*= (-1);

}

//-------------------------逻辑判断符重载--------------------------------

//==,!=,>,<,>=,<=只需要完整写出==,>的重载函数即可，其余逻辑判断符号重载函数可以通过两者的组合得到结果

bool TeamWorkBigInt::operator==(const TeamWorkBigInt& a) const {

if (this->getLength() == a.getLength()) {

for (int i = a.getLength() - 1; i >= 0; i--) {

if (this->integer[i] > a.integer[i]) {

return false;

}

else if (this->integer[i] < a.integer[i]) {

return false;

}

else {

if (i == 0) {

return true;

}

}

}

}

else if (this->getLength() > a.getLength()) {

return false;

}

else {

return false;

}

return 0;

}

bool TeamWorkBigInt::operator!=(const TeamWorkBigInt& a) const {

return !(\*this == a);

}

bool TeamWorkBigInt::operator>(const TeamWorkBigInt& a) const {

//先比较长度，长度长的数字大，长度相等再进一步比较

if (this->getLength() == a.getLength()) { //从最高位开始比较，最高位数组下标是length - 1,只要有一位数字比被比较的数字更大该数字就大

for (int i = a.getLength() - 1; i >= 0; i--) {

if (this->integer[i] > a.integer[i]) {

return true;

}

else if (this->integer[i] < a.integer[i]) {

return false;

}

else {

if (i == 0) {

return false;

}

}

}

}

else if (this->getLength() > a.getLength()) {

return true;

}

else {

return false;

}

return 0;

}

bool TeamWorkBigInt::operator>=(const TeamWorkBigInt& a) const {

if (\*this > a)

return true;

if (\*this == a)

return true;

return false;

}

bool TeamWorkBigInt::operator<(const TeamWorkBigInt& a) const {

return !(\*this > a);

}

bool TeamWorkBigInt::operator<=(const TeamWorkBigInt& a) const {

return !(\*this >= a);

}

//-------------------------逻辑操作符重载--------------------------------

//-------------------------加法操作符重载√--------------------------------

TeamWorkBigInt TeamWorkBigInt:: operator+(const TeamWorkBigInt& a) const

{

//保证b中储存位数更大的加数

TeamWorkBigInt b(a);

if (a.getLength() <= this->getLength())

b = \*this;

int l = min(a.getLength(), this->getLength());

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

b.integer[i] = integer[i] + a.integer[i];

}

//进位

for (int i = 0; i < b.getLength(); i++) {

if (b.integer[i] > 9) {

b.integer[i] -= 10;

b.integer[i + 1]++;

}

}

return b;

}

void TeamWorkBigInt:: operator+=(const TeamWorkBigInt& a) {

\*this = \*this + a;

}

TeamWorkBigInt TeamWorkBigInt:: operator+(const int b) const {

return \*this + TeamWorkBigInt(b);

}

TeamWorkBigInt TeamWorkBigInt:: operator+(const char\* b) const {

return \*this + TeamWorkBigInt(b);

}

//-------------------------加法操作符重载--------------------------------

//-------------------------减法操作符重载√--------------------------------

TeamWorkBigInt TeamWorkBigInt:: operator-(const TeamWorkBigInt& a) const {

TeamWorkBigInt b = \*this, c;

if (\*this > a) //结果为正数

{

//b = \*this;//被减数大于减数

for (int i = 0; i < a.getLength(); i++) {

if (b.integer[i] < a.integer[i]) {//如果不够减就向前结借位

b.integer[i] = 10 + b.integer[i] - a.integer[i];

b.integer[i + 1] -= 1; //前一位减1

}

else {

b.integer[i] -= a.integer[i];

}

}

return b;

}

else if (\*this == a)

return TeamWorkBigInt(0);

else //结果为负数，方法为c=-|b-a|

{

c = a - b;

c.integer[c.getLength()] \*= -1; //最高位设置为负数

return c;

}

}

TeamWorkBigInt TeamWorkBigInt:: operator-(int a) const {

//int a = b;

return \*this - TeamWorkBigInt(a);

}

void TeamWorkBigInt:: operator-=(const TeamWorkBigInt& a) {

return \*this = \*this - TeamWorkBigInt(a);

}

//-------------------------减法操作符重载--------------------------------

//-------------------------乘法操作符重载--------------------------------

TeamWorkBigInt TeamWorkBigInt::operator\*(const TeamWorkBigInt& b) const {

TeamWorkBigInt a; //功能先储存，然后计算得出结果

int l2, l1, l3;

l2 = b.getLength(); //记录第二个乘数的长度

l1 = this->getLength(); //记录第一个乘数的长度

if (\*this > b) {

for (int i = 0; i < l2; i++) {

for (int j = 0; j < l1; j++) {

a.integer[j + i] += this->integer[j] \* b.integer[i]; //存储乘法结果，可以实现错位相加求和

}

}

}

//进位

l3 = a.getLength(); //记录得数的位数

for (int i = 0; i < l3; i++) {

if (a.integer[i] > 9) {

a.integer[i + 1] += a.integer[i] / 10;

a.integer[i] = a.integer[i] % 10;

}

}

return a;

}

TeamWorkBigInt TeamWorkBigInt::operator\*(const int a) const {

TeamWorkBigInt b(a);

return (\*this) \*b ;

}

//TeamWorkBigInt operator\*(const TeamWorkBigInt& a, const int b) {

// return a\* TeamWorkBigInt(b);

//}

//-------------------------乘法操作符重载--------------------------------

//-------------------------除法操作符重载--------------------------------

TeamWorkBigInt TeamWorkBigInt::operator/(const TeamWorkBigInt& b) const {

TeamWorkBigInt q, r(\*this), d; //q用来储存按位相除后的商，r指代本身，d暂时储存左移操作后的除数

bool negative = (r.isNegative() != b.isNegative());

if (r.isNegative())

{

-r;

}

if (b.isNegative())

{

d = b\*(-1);

}

else {

d = b;

}

int l1 = this->getLength(); //被除数的位数

int l2 = b.getLength(); //除数的位数

int ld = l1 - l2; //位数差，商的最大位数

int i = 0;

for (i = ld; i >= 0; i--) //从最大位数差开始除

{

d = b << i;

while (r >= d) //如果该位上能至少商1

{

r -= d; //一直减，减到不能再减

q.integer[i]++; //减的次数就是该位的商

}

}

return q;

}

//左移操作符重载，左移n位相当于乘以2的n次幂，溢出则舍弃

//左移操作数的效果是扩大倍数，但是因为本算法中数位的存储顺序是颠倒的，所以表面上看起来是右移

TeamWorkBigInt operator<<(const TeamWorkBigInt& a, int b)

{

TeamWorkBigInt c;

int l = a.getLength();

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

{

c.integer[i + b] = a.integer[i];

}

return c;

}

TeamWorkBigInt TeamWorkBigInt::operator/(const int a) const {

return \*this / TeamWorkBigInt(a);

}

void TeamWorkBigInt::operator/=(const TeamWorkBigInt& a) {

\*this = \*this / a;

}

void TeamWorkBigInt::operator/=(const int a) {

TeamWorkBigInt b(a);

\*this =\*this/ b;

}

//-------------------------除法操作符重载--------------------------------

//幂函数

TeamWorkBigInt pow(TeamWorkBigInt a,int n) {

TeamWorkBigInt temp(1);

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

temp = temp \* a;

return temp;

}

//StudybarCommentBegin

int main()

{

//int i, N;

//TeamWorkBigInt n (10), b, x1, x2, s, t, pi;

TeamWorkBigInt n;

//TeamWorkBigInt n5("10000");

TeamWorkBigInt n5("5000000003456");

cout << n5 << endl;

TeamWorkBigInt n6(-50);

cout << n6 << endl;

n5 /= n6;

//n =n5/ n6;

cout << n5 << endl;

//cin >> N;

//N--;

//b = ;

//cout << n\*n;

/\* x1 = b \* 4 / 5;

x2 = b / -239;

s = x1 + x2;

for (i = 3; i <= N \* 2; i += 2)

{

x1 /= -25;

x2 /= -57121;

t = (x1 + x2) / i;

s += t;

}

pi = s \* 4;

cout << (pi / pow(n, 10)) << endl;\*/

return 0;

}

//StudybarCommentEnd