带符号大整数类

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <math.h>
#define max(x, y) ((x) < (y) ? (y) : (x))
#define min(x, y) ((x) > (y) ? (y) : (x))
#define abs(x) ((x) < 0 ? (-x) : (x))
#define maxlen 40005
#define ll long long
#define pi (acos(-1))
using namespace std;

struct comp {
    double re, im;
    comp(double r = 0, double i = 0): re(r), im(i) {}
    comp operator + (comp x) { return comp(re + x.re, im + x.im); }
    comp operator - (comp x) { return comp(re - x.re, im - x.im); }
    comp operator * (comp x) {
        return comp(re * x.re - im * x.im, re * x.im + im * x.re);
    }
};

class bint {
public:
    int len, s[maxlen];
    void inc() {
        int flag = neg(); s[0] += 1;
        for (int i = 0; i < len && s[i] == 10; s[i++] = 0) s[i + 1]++;
        if (neg() > flag) s[len++] = 0;
    }
    void dec() {
        int flag = neg(); s[0] -= 1;
        for (int i = 0; i < len && !(~s[i]); s[i++] = 9) s[i + 1]--;
        if (neg() < flag) s[len++] = 9;
    }
    void inv() { for (int i = 0; i < len; i++) s[i] = 9 - s[i]; inc(); }
    int neg() const { return s[len - 1] > 4 ? 9 : 0; }
    void clean() {
        while (len > 2 && s[len - 1] == 0 && s[len - 2] < 5) len--;
        while (len > 2 && s[len - 1] == 9 && s[len - 2] > 4) len--;
    }

public:
    bint(char* raw) {
        memset(s, 0, sizeof s);
        char *str = raw[0] == '-' ? raw + 1 : raw; len = (int)strlen(str) + 1;
        for (int i = 0; str[i]; i++) s[len - i - 2] = str[i] - '0';
        if (raw[0] == '-') inv();
    }
    bint(int v = 0) {
        memset(s, 0, sizeof s);
        int val = abs(v); if (!val) { len = 2; return; }
        for (len = 0; val; val /= 10) s[len++] = val % 10; len++;
        if (v < 0) inv();
    }
    bint(const bint& a) { len = a.len; memcpy(s, a.s, sizeof(int) * len); }
    bint& operator = (const bint& a) {
        len = a.len; memcpy(s, a.s, sizeof(int) * len); return *this;
    }
    int to_int() {
        int ans = 0; bint tmp = *this; if (neg()) tmp.inv();
        for (int i = 0, b = 1; i < len; i++, b *= 10) ans += b * tmp.s[i];
        return neg() ? -ans : ans;
    }
    char* to_str() {
        char *ans = new char[len + 5]; int cnt = 0;
        bint tmp = *this; if (neg()) { tmp.inv(); ans[cnt++] = '-'; }
        for (int i = tmp.len - 1; ~i; i--)
            if (tmp.s[i] || i == 0) {
                for (int j = i; ~j; j--) ans[cnt++] = tmp.s[j] + '0'; break;
        }
        ans[cnt++] = 0; return ans;
    }
    friend bint operator + (bint a, bint b);
    friend bint operator - (bint a, bint b);
    friend bint operator * (bint a, bint b);
    friend bint operator * (bint a, int b);
    friend bint operator / (bint a, bint b);
    friend bint operator / (bint a, int b);
    friend bint operator % (bint a, bint b);
    friend bint operator % (bint a, int b);
    friend bint operator << (bint a, int k);
    friend bint operator >> (bint a, int k);
    bint operator <<= (int k);
```

```cpp
    bint operator >>= (int k);
    bint operator += (bint a)   { *this = *this + a; return *this; }
    bint operator -= (bint a)   { *this = *this - a; return *this; }
    bint operator *= (bint a)   { *this = *this * a; return *this; }
    bint operator *= (int b);
    bint operator /= (bint a)   { *this = *this / a; return *this; }
    bool operator < (bint a)    { return (*this - a).neg() > 0; }
    bool operator > (bint a)    { return (a - *this).neg() > 0; }
    bool operator <= (bint a)   { return (a - *this).neg() <= 0; }
    bool operator >= (bint a)   { return (*this - a).neg() <= 0; }
    bool operator == (bint a)   { bint t = *this - a; return t.len < 10 && !t.to_int(
); }
    bool operator != (bint a)   { return !(*this == a); }
    bint operator ++ (int)      { bint tmp(*this); inc(); return tmp; }
    bint operator ++ ()         { inc(); return *this; }
    bint operator -- (int)      { bint tmp(*this); dec(); return tmp; }
    bint operator -- ()         { dec(); return *this; }
    bint operator - ()          { bint tmp(*this); tmp.inv(); return tmp; }
    friend std::ostream& operator << (std::ostream&, const bint&);
    friend std::istream& operator >> (std::istream&, bint&);
};

bint bint::operator <<= (int k) {
    for (int i = len - 1; ~i; i--) s[i + k] = s[i];
    len += k; clean(); return *this;
}

bint bint::operator >>= (int k) {
    for (int i = 0; i < len - k; i++) s[i] = s[i + k];
    len -= k; if (len <= 0) *this = 0; clean(); return *this;
}

bint operator + (bint a, bint b) {
    bint ans; ans.len = max(a.len, b.len) + 1;
    for (int i = a.len, ta = a.neg(); i < ans.len; i++) a.s[i] = ta;
    for (int i = b.len, tb = b.neg(); i < ans.len; i++) b.s[i] = tb;
    for (int i = 0; i <= a.len || i <= b.len; i++) {
        ans.s[i] += a.s[i] + b.s[i];
        if (ans.s[i] > 9) { ans.s[i] -= 10; ans.s[i + 1]++; }
    }
    ans.clean(); return ans;
}

bint operator - (bint a, bint b) { b.inv(); return a + b; }
```

```
void FFT(comp* a, int* g, int n, int f) {
    for (int i = 0; i < n; i++)
        if (g[i] > i) swap(a[i], a[g[i]]);
    for (int i = 1; i < n; i <<= 1) {
        comp wn1(cos(pi / i), f * sin(pi / i));
        for (int j = 0; j < n; j += (i << 1)) {
            comp wnk(1, 0);
            for (int k = 0; k < i; k++, wnk = wnk * wn1) {
                comp x = a[j + k], y = wnk * a[j + k + i];
                a[j + k] = x + y; a[j + k + i] = x - y;
            }
        }
    }
    if (!(~f)) for (int i = 0; i < n; i++) a[i].re /= n;
}

bint operator * (bint a, bint b) {
    if (b.len < 9) return a * b.to_int();
    bool flag = (a.neg() > 0) ^ (b.neg() > 0);
    if (a.neg()) a.inv(); if (b.neg()) b.inv();
    int n = max(a.len, b.len) << 1, _n = 1, t = -1;
    while (_n < n) { _n <<= 1; t++; }
    int* g = new int[_n]; g[0] = 0;
    comp *tmpa = new comp[_n], *tmpb = new comp[_n];
    for (int i = 1; i < _n; i++)
        g[i] = (g[i >> 1] >> 1) | ((i & 1) << t);
    for (int i = 0; i < a.len; i++) tmpa[i] = comp(a.s[i], 0);
    for (int i = 0; i < b.len; i++) tmpb[i] = comp(b.s[i], 0);
    FFT(tmpa, g, _n, 1); FFT(tmpb, g, _n, 1);
    for (int i = 0; i < _n; i++) tmpa[i] = tmpa[i] * tmpb[i];
    FFT(tmpa, g, _n, -1);
    bint ans; ans.len = a.len + b.len + 1;
    for (int i = 0; i < min(_n, ans.len); i++) ans.s[i] = tmpa[i].re + 0.01;
    for (int i = 0; i < ans.len - 2; ans.s[i++] %= 10)
        ans.s[i + 1] += ans.s[i] / 10;
    ans.clean(); return flag ? -ans : ans;
}

bint operator * (bint a, int b) {
    if (b > 99999999) return a * bint(b);
    bool flag = (a.neg() > 0) ^ (b < 0);
    if (a.neg()) a.inv(); if (b < 0) b = -b;
    bint ans; ans.len = a.len + 10;
    for (int i = 0; i < a.len; i++)
        ans.s[i] += a.s[i] * b;
    for (int i = 0; i < ans.len - 2; ans.s[i++] %= 10)
        ans.s[i + 1] += ans.s[i] / 10;
    ans.clean(); return flag ? -ans : ans;
}

bint bint::operator *= (int b) {
    if (b > 99999999) return (*this) * bint(b);
    bool flag = (neg() > 0) ^ (b < 0);
    if (neg()) inv(); if (b < 0) b = -b;
    for (int i = 0; i < len; i++) s[i] *= b;
    len += 10;
    for (int i = 0; i < len - 2; s[i++] %= 10)
        s[i + 1] += s[i] / 10;
    clean(); return flag ? -(*this) : *this;
}

bint operator / (bint a, bint b) {
    if (b.len < 10) return a / b.to_int();
    bool flag = (a.neg() > 0) ^ (b.neg() > 0);
    if (a.neg()) a.inv(); if (b.neg()) b.inv();
    bint invb = 1; int lim = a.len + 5, fac = b.len;
    for (int i = 0; i < 100; i++) {
        bint two(0), last = invb; two.len = fac + 1; two.s[fac] = 2;
        invb = invb * (two - b * invb); fac <<= 1;
        if (invb.len > lim) {
            fac -= invb.len - lim;
            for (int i = 0; i < lim; i++)
                invb.s[i] = invb.s[i + invb.len - lim];
            for (int i = lim; i < invb.len; i++)
                invb.s[i] = 0;
            invb.len = lim;
        }
        if (last == invb) break;
    }
    bint ans = a * invb; if (ans.len < fac) return 0;
    for (int i = 0; i < ans.len - fac; i++)
        ans.s[i] = ans.s[i + fac];
    for (int i = ans.len - fac; i < ans.len; i++)
        ans.s[i] = 0;
    ans.len -= fac; if (ans.len == 0) ans = 0;
    bint mod = a - b * ans;
    while (mod >= b) { ans++; mod -= b;}
    ans.clean(); return flag ? -ans : ans;
}
```

```
bint operator / (bint a, int b) {
    bool flag = (a.neg() > 0) ^ (b < 0);
    if (a.neg()) a.inv(); if (b < 0) b = -b;
    bint ans; ans.len = a.len;
    long long mod = 0;
    for (int i = a.len - 1; ~i; i--) {
        mod *= 10; mod += a.s[i];
        ans.s[i] = (int)(mod / (ll)b); mod %= b;
    }
    ans.clean(); return flag ? -ans : ans;
}

bint operator % (bint a, bint b) {
    if (b.len < 10) return a % b.to_int();
    bool flag = (a.neg() > 0);
    if (a.neg()) a.inv(); if (b.neg()) b.inv();
    bint invb = 1; int lim = a.len + 5, fac = b.len;
    for (int i = 0; i < 100; i++) {
        bint two(0), last = invb; two.len = fac + 1; two.s[fac] = 2;
        invb = invb * (two - b * invb); fac <<= 1;
        if (invb.len > lim) {
            fac -= invb.len - lim;
            for (int i = 0; i < lim; i++)
                invb.s[i] = invb.s[i + invb.len - lim];
            for (int i = lim; i < invb.len; i++)
                invb.s[i] = 0;
            invb.len = lim;
        }
        if (last == invb) break;
    }
    bint ans = a * invb; if (ans.len < fac) return 0;
    for (int i = 0; i < ans.len - fac; i++)
        ans.s[i] = ans.s[i + fac];
    for (int i = ans.len - fac; i < ans.len; i++)
        ans.s[i] = 0;
    ans.len -= fac; if (ans.len == 0) ans = 0;
    bint mod = a - b * ans;
    while (mod >= b) { ans++; mod -= b;}
    mod.clean(); return flag ? -mod : mod;
}

bint operator % (bint a, int b) {
    bool flag = (a.neg() > 0);
    if (a.neg()) a.inv(); if (b < 0) b = -b;
    bint ans; ans.len = a.len;
    long long mod = 0;
    for (int i = a.len - 1; ~i; i--) {
        mod *= 10; mod += a.s[i];
        ans.s[i] = (int)(mod / (ll)b); mod %= b;
    }
    return flag ? -int(mod) : int(mod);
}

bint operator << (bint a, int k) {
    a.len += k;
    for (int i = a.len - 1; ~i; i--) a.s[i + k] = a.s[i];
    a.clean(); return a;
}

bint operator >> (bint a, int k) {
    a.len -= k;
    for (int i = 0; i < a.len; i++) a.s[i] = a.s[i + k];
    if (a.len <= 0) a = 0; a.clean(); return a;
}

std::ostream& operator << (std::ostream& out, const bint& a) {
    bint tmp = a; if (a.neg()) { putchar('-'); tmp.inv(); }
    for (int i = tmp.len - 1; ~i; i--)
        if (tmp.s[i] || i == 0) {
            for (int j = i; ~j; j--) putchar(tmp.s[j] + '0'); break;
        }
    return out;
}

std::istream& operator >> (std::istream& cin, bint& a) {
    char s[maxlen]; scanf("%s", s); a = bint(s);
    return cin;
}

int main() {
    bint a, b;
    cin >> a >> b;
    cout << a / b << endl;
    return 0;
}
```

无符号大整数类

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <math.h>
#define max(x, y) ((x) < (y) ? (y) : (x))
#define min(x, y) ((x) > (y) ? (y) : (x))
#define maxlen 40005
#define ll long long
#define pi (acos(-1))
using namespace std;

struct comp {
    double re, im;
    comp(double r = 0, double i = 0): re(r), im(i) {}
    comp operator + (comp x) { return comp(re + x.re, im + x.im); }
    comp operator - (comp x) { return comp(re - x.re, im - x.im); }
    comp operator * (comp x) {
        return comp(re * x.re - im * x.im, re * x.im + im * x.re);
    }
};

class ubint {
public:
    int len, s[maxlen];
    void clean() { while (len > 1 && s[len - 1] == 0) len--; }
    void inc() {
        s[0]++; for (int i = 0; i < len && s[i] == 10; s[i++] = 0) s[i + 1]++;
        if (s[len]) len++;
    }
    void dec() {
        s[0]--; for (int i = 0; i < len && !(~s[i]); s[i++] = 9) s[i + 1]--;
        if (s[len - 1] == 0) len--;
    }
    int cmp(const ubint& a) {
        if (len != a.len) return len - a.len;
        for (int i = len - 1; ~i; i--)
            if (s[i] != a.s[i]) return s[i] - a.s[i];
        return 0;
    }

public:
    ubint(char* str) {
        memset(s, 0, sizeof s); len = (int)strlen(str);
        for (int i = 0; str[i]; i++) s[len - i - 1] = str[i] - '0';
    }
    ubint(int v = 0) {
        memset(s, 0, sizeof s);
        if (!v) { len = 1; s[0] = 0; return; }
        for (len = 0; v; v /= 10) s[len++] = v % 10;
    }
    ubint(const ubint& a) {
        len = a.len; memcpy(s, a.s, sizeof s); clean();
    }
    ubint& operator = (const ubint& a) {
        len = a.len; memcpy(s, a.s, sizeof s); clean(); return *this;
    }
    int to_int() {
        int ans = 0;
        for (int i = 0, b = 1; i < len; i++, b *= 10) ans += b * s[i];
        return ans;
    }
    char* to_str() {
        char *ans = new char[len + 3]; int cnt = 0; clean();
        for (int i = len - 1; ~i; i--) ans[cnt++] = s[i] + '0';
        ans[cnt++] = 0; return ans;
    }
    friend ubint operator + (ubint a, ubint b);
    friend ubint operator - (ubint a, ubint b);
    friend ubint operator * (ubint a, ubint b);
    friend ubint operator * (ubint a, int b);
    friend ubint operator / (ubint a, ubint b);
    friend ubint operator / (ubint a, int b);
    friend ubint operator % (ubint a, ubint b);
    friend ubint operator % (ubint a, int b);
    ubint operator += (ubint a) { *this = *this + a; return *this; }
    ubint operator -= (ubint a) { *this = *this - a; return *this; }
    ubint operator *= (ubint a) { *this = *this * a; return *this; }
    ubint operator *= (int b);
    ubint operator /= (ubint a) { *this = *this / a; return *this; }
    bool operator < (ubint a)   { return cmp(a) < 0; }
    bool operator > (ubint a)   { return cmp(a) > 0; }
    bool operator <= (ubint a)  { return cmp(a) <= 0; }
    bool operator >= (ubint a)  { return cmp(a) >= 0; }
    bool operator == (ubint a)  { return cmp(a) == 0; }
    bool operator != (ubint a)  { return cmp(a) != 0; }
    ubint operator ++ (int)     { ubint tmp(*this); inc(); return tmp; }
    ubint operator ++ ()        { inc(); return *this; }
    ubint operator -- (int)     { ubint tmp(*this); dec(); return tmp; }
    ubint operator -- ()        { dec(); return *this; }
    friend std::ostream& operator << (std::ostream&, const ubint&);
    friend std::istream& operator >> (std::istream&, ubint&);
};

ubint operator + (ubint a, ubint b) {
    ubint ans; ans.len = max(a.len, b.len) + 1;
    for (int i = 0; i < a.len || i < b.len; i++) {
        ans.s[i] += a.s[i] + b.s[i];
        if (ans.s[i] > 9) { ans.s[i] -= 10; ans.s[i + 1]++; }
    }
    ans.clean(); return ans;
}

ubint operator - (ubint a, ubint b) {
    ubint ans; ans.len = a.len + 1;
    for (int i = 0; i < a.len; i++) {
        ans.s[i] += a.s[i] - b.s[i];
        if (ans.s[i] < 0) { ans.s[i] += 10; ans.s[i + 1]--; }
    }
    ans.clean(); return ans;
}

void FFT(comp* a, int* g, int n, int f) {
    for (int i = 0; i < n; i++)
        if (g[i] > i) swap(a[i], a[g[i]]);
    for (int i = 1; i < n; i <<= 1) {
        comp wn1(cos(pi / i), f * sin(pi / i));
        for (int j = 0; j < n; j += (i << 1)) {
            comp wnk(1, 0);
            for (int k = 0; k < i; k++, wnk = wnk * wn1) {
                comp x = a[j + k], y = wnk * a[j + k + i];
                a[j + k] = x + y; a[j + k + i] = x - y;
            }
        }
    }
    if (!(~f)) for (int i = 0; i < n; i++) a[i].re /= n;
}

ubint operator * (ubint a, ubint b) {
    if (b.len < 9) return a * b.to_int();
    int n = max(a.len, b.len) << 1, _n = 1, t = -1;
    while (_n < n) { _n <<= 1; t++; }
    int* g = new int[_n]; g[0] = 0;
    comp *tmpa = new comp[_n], *tmpb = new comp[_n];
    for (int i = 1; i < _n; i++)
        g[i] = (g[i >> 1] >> 1) | ((i & 1) << t);
    for (int i = 0; i < a.len; i++) tmpa[i] = comp(a.s[i], 0);
    for (int i = 0; i < b.len; i++) tmpb[i] = comp(b.s[i], 0);
    FFT(tmpa, g, _n, 1); FFT(tmpb, g, _n, 1);
    for (int i = 0; i < _n; i++) tmpa[i] = tmpa[i] * tmpb[i];
    FFT(tmpa, g, _n, -1);
    ubint ans; ans.len = a.len + b.len + 1;
    for (int i = 0; i < min(_n, ans.len); i++) ans.s[i] = tmpa[i].re + 0.01;
    for (int i = 0; i < ans.len - 2; ans.s[i++] %= 10)
        ans.s[i + 1] += ans.s[i] / 10;
    ans.clean(); return ans;
}

ubint operator * (ubint a, int b) {
    if (b > 99999999) return a * ubint(b);
    ubint ans; ans.len = a.len + 10;
    for (int i = 0; i < a.len; i++)
        ans.s[i] += a.s[i] * b;
    for (int i = 0; i < ans.len - 2; ans.s[i++] %= 10)
        ans.s[i + 1] += ans.s[i] / 10;
    ans.clean(); return ans;
}

ubint ubint::operator *= (int b) {
    if (b > 99999999) return (*this) * ubint(b);
    for (int i = 0; i < len; i++) s[i] *= b;
    len += 10;
    for (int i = 0; i < len - 2; s[i++] %= 10)
        s[i + 1] += s[i] / 10;
    clean(); return *this;
}

ubint operator / (ubint a, ubint b) {
    if (b.len < 10) return a / b.to_int();
    ubint invb = 1; int lim = a.len + 5, fac = b.len;
    for (int i = 0; i < 100; i++) {
        ubint two(0), last = invb; two.len = fac + 1; two.s[fac] = 2;
        invb = invb * (two - b * invb); fac <<= 1;
        if (invb.len > lim) {
            fac -= invb.len - lim;
            for (int i = 0; i < lim; i++)
                invb.s[i] = invb.s[i + invb.len - lim];
            for (int i = lim; i < invb.len; i++)
                invb.s[i] = 0;
            invb.len = lim;
        }
        if (last == invb) break;
    }
    ubint ans = a * invb; if (ans.len < fac) return 0;
    for (int i = 0; i < ans.len - fac; i++)
        ans.s[i] = ans.s[i + fac];
    for (int i = ans.len - fac; i < ans.len; i++)
        ans.s[i] = 0;
    ans.len -= fac; if (ans.len == 0) ans = 0;
    ubint mod = a - b * ans;
    while (mod >= b) { ans++; mod -= b;}
    ans.clean(); return ans;
}

ubint operator / (ubint a, int b) {
    ubint ans; ans.len = a.len; ll mod = 0;
    for (int i = a.len - 1; ~i; i--) {
        mod *= 10; mod += a.s[i];
        ans.s[i] = (int)(mod / (ll)b); mod %= b;
    }
    ans.clean(); return ans;
}

ubint operator % (ubint a, ubint b) {
    if (b.len < 10) return a % b.to_int();
    ubint invb = 1; int lim = a.len + 5, fac = b.len;
    for (int i = 0; i < 100; i++) {
        ubint two(0), last = invb; two.len = fac + 1; two.s[fac] = 2;
        invb = invb * (two - b * invb); fac <<= 1;
        if (invb.len > lim) {
            fac -= invb.len - lim;
            for (int i = 0; i < lim; i++)
                invb.s[i] = invb.s[i + invb.len - lim];
            for (int i = lim; i < invb.len; i++)
                invb.s[i] = 0;
            invb.len = lim;
        }
        if (last == invb) break;
    }
    ubint ans = a * invb; if (ans.len < fac) return 0;
    for (int i = 0; i < ans.len - fac; i++)
        ans.s[i] = ans.s[i + fac];
    for (int i = ans.len - fac; i < ans.len; i++)
        ans.s[i] = 0;
    ans.len -= fac; if (ans.len == 0) ans = 0;
    ubint mod = a - b * ans;
    while (mod >= b) { ans++; mod -= b;}
    ans.clean(); return mod;
}

ubint operator % (ubint a, int b) {
    ubint ans; ans.len = a.len; ll mod = 0;
    for (int i = a.len - 1; ~i; i--) {
        mod *= 10; mod += a.s[i];
        ans.s[i] = (int)(mod / (ll)b); mod %= b;
    }
    return int(mod);
}

std::ostream& operator << (std::ostream& out, const ubint& a) {
    for (int i = a.len - 1; ~i; i--) putchar(a.s[i] + '0');
    return out;
}

std::istream& operator >> (std::istream& cin, ubint& a) {
    char s[maxlen]; scanf("%s", s); a = ubint(s);
    return cin;
}

int main() {
    ubint a, b; cin >> a >> b;
    cout << a % b << endl;
    return 0;
}
```

分数类

```
#define sign(x) ((x) > 0 ? 1 : -1)

inline int gcd(int a, int b) {
    while (b) { int t = a % b; a = b; b = t; }
    return a;
}

inline int lcm(int a, int b) { return (a * b) / gcd(a, b); }

struct frac {
    int x, y; // x / y
    frac(int val = 0) { x = val; y = 1; }
    frac(int a, int b) {
        int t = gcd(abs(a), abs(b));
        x = (a / t) * sign(b);
        y = (x == 0 ? 1 : (abs(b) / t));
    }
    frac(char* s) {
        int len = strlen(s), a, b, t;
        for (int i = 0; i < len; i++)
            if (s[i] == '/') {
                sscanf(s, "%d/%d", &a, &b); *this = frac(a, b); return;
            } else if (s[i] == '.') {
                sscanf(s, "%d.%d", &a, &t);
                for (int j = 0; j < len - i - 1; j++) b *= 10;
                a *= b; a += t; *this = frac(a, b); return;
            }
        sscanf(s, "%d", &a); x = a; y = 1;
    }
    frac operator + (const frac& f) { return frac(x * f.y + f.x * y, y * f.y); }
    frac operator - (const frac& f) { return frac(x * f.y - f.x * y, y * f.y); }
    frac operator * (const frac& f) { return frac(x * f.x, y * f.y); }
    frac operator / (const frac& f) { return frac(x * f.y, y * f.x); }
};

inline void print(const frac& f) {
    printf("%d", f.x); if (f.y != 1) printf("/%d", f.y);
}
```

矩阵类

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <cmath>
#define MAXN 100
#define MAXM 200
//N * M
using namespace std;

template <class T>
struct matrix {
    int n, m;
    T dat[MAXN][MAXM];//both start from 1
    matrix(int nn = 0, int mm = 0): n(nn), m(mm) {}
    T* operator[](int i) { return dat[i]; }
};

template <class T>
void matcpy(matrix<T>& a, matrix<T>& b) {
    b.n = a.n; b.m = a.m;
    memcpy(b.dat, a.dat, sizeof(T) * MAXM * MAXN);
}

template <class T>
void matadd(matrix<T>& a, matrix<T>& b, matrix<T>& ans) {
    ans.n = a.n; ans.m = a.m;
    for (int i = 1; i <= a.n; i++)
        for (int j = 1; j <= a.m; j++)
            ans[i][j] = a[i][j] + b[i][j];
}

template <class T>
void matsub(matrix<T> &a, matrix<T> &b, matrix<T>& ans) {
    ans.n = a.n; ans.m = a.m;
    for (int i = 1; i <= a.n; i++)
        for (int j = 1; j <= a.m; j++)
            ans[i][j] = a[i][j] - b[i][j];
}

template <class T>
void matmul(matrix<T> &a, matrix<T> &b, matrix<T> &ans) {
    ans.n = a.n; ans.m = b.m;
    for (int i = 1; i <= a.n; i++)
        for (int j = 1; j <= b.m; j++) {
            T sum = 0;
            for (int k = 1; k <= a.m; k++)
                sum += a[i][k] * b[k][j];
            ans[i][j] = sum;
        }
}

template <class T>
void matmul(matrix<T> &a, int k, matrix<T> &ans) {
    ans.n = a.n; ans.m = a.m;
    for (int i = 1; i <= a.n; i++)
        for (int j = 1; j <= a.m; j++)
            ans[i][j] = a[i][j] * k;
}

template <class T>
void matdiv(matrix<T> &a, int k, matrix<T> &ans) {
    ans.n = a.n; ans.m = a.m;
    for (int i = 1; i <= a.n; i++)
        for (int j = 1; j <= a.m; j++)
            ans[i][j] = a[i][j] / k;
}

template <class T>
T det(matrix<T> &a) {
    if (a.m - a.n) exit(1);//error
    T ans = 1;
    for (int px = 1; px <= a.n; px++) {
        if (a[px][px] == 0)
            for (int i = px + 1; i <= a.n; i++)
                if (a[i][px] != 0) {
                    for (int j = px; j <= a.n; j++) swap(a[i][j], a[px][j]);
                    ans = -ans; break;
                }
        if (a[px][px] == 0) return 0;
        ans *= a[px][px];
        for (int i = px + 1; i <= a.n; i++) {
            T k = a[i][px] / a[px][px];
            for (int j = px; j <= a.n; j++) a[i][j] -= a[px][j] * k;
        }
    }
    return ans;
```

```
}

template <class T>
void inv(matrix<T> &a, matrix<T> &ans) {
    if (a.n - a.m) exit(1);
    a.m <<= 1;
    for (int i = 1; i <= a.n; i++) {
        for (int j = a.n + 1; j <= a.m; j++) a[i][j] = 0;
        a[i][i + a.n] = 1;
    }
    for (int px = 1; px <= a.n; px++) {
        if (a[px][px] == 0)
            for (int i = px + 1; i <= a.n; i++)
                if (a[i][px] != 0) {
                    for (int j = px; j <= a.m; j++) swap(a[i][j], a[px][j]);
                    break;
                }
        if (a[px][px] == 0) { ans.n = -1; return; }
        for (int j = px + 1; j <= a.m; j++) a[px][j] /= a[px][px];
        a[px][px] = 1;
        for (int i = px + 1; i <= a.n; i++) {
            T k = a[i][px];
            for (int j = px; j <= a.m; j++) a[i][j] -= a[px][j] * k;
        }
    }
    for (int px = a.n; px >= 1; px--)
        for (int i = px - 1; i >= 1; i--)
            if (a[i][px] != 0) {
                T k = a[i][px];
                for (int j = px; j <= a.m; j++) a[i][j] -= a[px][j] * k;
            }
    ans.n = ans.m = a.n;
    for (int i = 1; i <= a.n; i++)
        for (int j = 1; j <= a.n; j++)
            ans[i][j] = a[i][j + a.n];
}
```

计算几何

```
#include <iostream>
#include <cmath>
#include <stdio.h>
#include <algorithm>
#include <vector>

//////////////////////////////////////////////////////
//L = line, C = circle, S = segment, P = point, F = flat
//////////////////////////////////////////////////////

using namespace std;
const double eps = 1e-8;
const double pi = acos(-1.0);

//////////////////////////////////////////////////////
//点和向量的定义
//////////////////////////////////////////////////////

struct point {
    double x, y;
    point(double x = 0, double y = 0):x(x), y(y) {}
};
typedef point vec;//实现上，vector不过是point的别名
vec operator + (vec a, vec b) { return vec(a.x + b.x, a.y + b.y); }
vec operator - (point a, point b) { return vec(a.x - b.x, a.y - b.y); }
vec operator * (vec a, double p) { return vec(a.x * p, a.y * p); }
vec operator / (vec a, double p) { return vec(a.x / p, a.y / p); }
bool operator < (const point &a, const point &b) {//不要使用>,而使用<
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}
int dcmp(double x) {//三态函数
    if (fabs(x) < eps)
        return 0;
    else
        return x < 0 ? -1 : 1;
}
bool operator == (const point &a, const point &b) {
    return dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y == 0);
}

//////////////////////////////////////////////////////
```

```
//点和向量的基本运算
/////////////////////////////////////////////////////

double dot(vec a, vec b) { return a.x*b.x + a.y*b.y; };//点积
double length(vec a) { return sqrt(dot(a, a)); }
double angle(vec a, vec b) {//俩向量夹角
    double ans = dot(a, b) / length(a) / length(b);
    if (dcmp(ans - 1) == 0) ans = 1;
    if (dcmp(ans + 1) == 0) ans = -1;
    return acos(ans);
}
double angle(vec a) { return atan2(a.y, a.x); }//向量的极角
double cross(vec a, vec b) { return a.x * b.y - a.y * b.x; }//叉积
double area(point a, point b, point c) {//以a,b,c为顶点的三角形的面积
    return cross(b - a, c - a) / 2;
}
vec rotate(vec a, double rad) {//将向量a绕起点逆时针旋转rad角
    return vec(a.x*cos(rad) - a.y*sin(rad), a.x*sin(rad) + a.y*cos(rad));
}
vec normal(vec a) {//a的单位法线(即左转90度,再长度归一)
    double l = length(a);
    return vec(-a.y / l, a.x / l);
}

/////////////////////////////////////////////////////
//点和直线
/////////////////////////////////////////////////////

struct line {
    point p;
    vec v;
    double rad;//极角
    line() {}
    line(point p, vec v):p(p), v(v) { rad = angle(v); }
    bool operator < (const line& l) const {
        return rad < l.rad;
    }
};
point get_LL_intersection(point p, vec v, point q, vec w) {//俩直线交点
    vec u = p - q;//直线1:p+t*v,直线2:q+t*w
    double t = cross(w, u) / cross(v, w);
    return p + v * t;
}
double dis_of_PL(point p, point a, point b) {//点p到直线AB的距离
    vec v1 = b - a, v2 = p - a;
    return fabs(cross(v1, v2)) / length(v1);
}
double dis_of_PS(point p, point a, point b) {//点p到线段AB的距离
    if (a == b) return length(p - a);
    vec v1 = b - a, v2 = p - a, v3 = p - b;
    if (dcmp(dot(v1, v2)) < 0)
        return length(v2);//返回AP长度
    else if (dcmp(dot(v1, v3)) > 0)
        return length(v3);//返回BP长度
    else
        return fabs(cross(v1, v2)) / length(v1);//计算方法同直线方法
}
bool P_on_L_left(point a, point p, vec v) {//判断点a是否在线段p+tv左侧
    return cross(v, a - p) > 0;
}
point get_PL_projection(point p, point a, point b) {//p点在直线AB上的投影点
    vec v = b - a;
    return a + v*(dot(v, p - a) / dot(v, v));
}
bool SS_intersection(point a1, point a2, point b1, point b2) {//线段相交判定
    double c1 = cross(a2 - a1, b1 - a1), c2 = cross(a2 - a1, b2 - a1),
        c3 = cross(b2 - b1, a1 - b1), c4 = cross(b2 - b1, a2 - b1);
    return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
}
bool P_on_S(point p, point a1, point a2) {//判断点是否在线段上
    return dcmp(cross(a1 - p, a2 - p)) == 0 && dcmp(dot(a1 - p, a2 - p)) < 0;
}

/////////////////////////////////////////////////////
//多边形
/////////////////////////////////////////////////////

double polygon_area(point *p, int n) {
    double area = 0;
    for (int i = 1; i < n - 1; i++)
        area += cross(p[i] - p[0], p[i + 1] - p[0]);
    return area / 2;
}

/////////////////////////////////////////////////////
//圆的相关计算
/////////////////////////////////////////////////////

struct circle {
    point c;
    double r;
    circle(point c, double r):c(c), r(r) {}
    point get_point(double rad) {//获得圆上一点的坐标
        return point(c.x + cos(rad)*r, c.y + sin(rad)*r);
    }
};
int get_LC_intersection(point p, vec v, circle cir, //p+tv为直线参数方程
    double &t1, double &t2, vector<point> &sol) {//t1,t2存放交点在直线上的参数,sol存放交点
    double a = v.x, b = p.x - cir.c.x, c = v.y, d = p.y - cir.c.y;
    double e = a*a + c*c, f = 2 * (a*b + c*d), g = b*b + d*d - cir.r*cir.r;
    double delta = f*f - 4 * e*g;
    if (dcmp(delta) < 0)return 0;//返回交点的个数
    if (dcmp(delta) == 0) {
        t1 = t2 = -f / (2 * e);
        sol.push_back(p + v*t1);
        return 1;
    }
    t1 = (-f - sqrt(delta)) / (2 * e); sol.push_back(p + v*t1);//相交,两个交点
    t2 = (-f + sqrt(delta)) / (2 * e); sol.push_back(p + v*t2);
    return 2;
}
int get_CC_intersection(circle cir1, circle cir2, vector<point> &sol) {//两圆相交
    double d = length(cir1.c - cir2.c);//圆心距
    if (dcmp(d) == 0) {
        if (dcmp(cir1.r - cir2.r) == 0) return -1;//两圆重合
        else return 0;//两圆内含
    }
    if (dcmp(cir1.r + cir2.r - d) < 0) return 0;//两圆相离
    if (dcmp(fabs(cir1.r - cir2.r) - d) > 0) return 0;//两圆内含
    double a = angle(cir2.c - cir1.c);//C1C2的极角
    double da = acos((cir1.r*cir1.r + d*d - cir2.r*cir2.r) / (2 * cir1.r*d));
    point p1 = cir1.get_point(a - da), p2 = cir1.get_point(a + da);
    sol.push_back(p1);
    if (p1 == p2) return 1;
    sol.push_back(p2); return 2;
}
int get_PC_tangents(point p, circle cir, vec *v) {//过点p到圆cir的切线,v[i]是第i条切线的向
量
    vec u = cir.c - p;
    double dis = length(u);
    if (dis < cir.r) return 0;
    else if (dcmp(dis - cir.r) == 0) {
        v[0] = rotate(u, pi / 2);
        return 1;
    } else {
        double rad = asin(cir.r / dis);
        v[0] = rotate(u, -rad);
        v[1] = rotate(u, rad);
        return 2;
    }
}
int get_CC_tangents(circle cir1, circle cir2,//返回切线的条数,-1表示无穷
    point *p1, point *p2) {//p1[i]和p2[i]分别表示第i条切线在圆1与圆2上的切点
    int cnt = 0;
    if (cir1.r < cir2.r) {
        swap(cir1, cir2);
        swap(p1, p2);
    }
    double d2 = (cir1.c.x - cir2.c.x)*(cir1.c.x - cir2.c.x)
        + (cir1.c.y - cir2.c.y)*(cir1.c.y - cir2.c.y);//d^2为圆心距的平方
    double rdiff = cir1.r - cir2.r;//半径之差
    double rsum = cir1.r + cir2.r;//半径之和
    if (d2 < rdiff*rdiff) return 0;//两圆内含
    double base = angle(cir2.c - cir1.c);
    if (d2 == 0 && cir1.r == cir2.r) return -1;//无限条切线
    if (d2 == rdiff*rdiff) {//内切,1条切线
        p1[cnt] = cir1.get_point(base);
        p2[cnt] = cir2.get_point(base);
        cnt++; return 1;
    }
    double rad = acos((cir1.r - cir2.r) / sqrt(d2));//有外公切线,一定为2条
    p1[cnt] = cir1.get_point(base + rad);
    p2[cnt] = cir2.get_point(base + rad);
    cnt++;
    p1[cnt] = cir1.get_point(base - rad);
    p2[cnt] = cir2.get_point(base - rad);
    cnt++;
    if (d2 == rsum*rsum) {//1条内公切线
        p1[cnt] = cir1.get_point(base);
        p2[cnt] = cir2.get_point(base + pi);
        cnt++;
    } else if (d2>rsum*rsum) {//2条内公切线
        double rad = acos((cir1.r + cir2.r) / sqrt(d2));
        p1[cnt] = cir1.get_point(base + rad);
        p2[cnt] = cir2.get_point(base + rad + pi);
        cnt++;
        p1[cnt] = cir1.get_point(base - rad);
        p2[cnt] = cir2.get_point(base - rad + pi);
        cnt++;
    }
    return cnt;
}

/////////////////////////////////////////////////////
//二维几何常用算法
/////////////////////////////////////////////////////

int P_in_polygon(point p, point *poly, int size) {//判断点是否在多边形中
    int wn = 0;
    for (int i = 0; i < size; i++) {
        if (P_on_S(p, poly[i], poly[(i + 1) % size])) return -1;
        int k = dcmp(cross(poly[(i + 1) % size] - poly[i], p - poly[i]));
        int d1 = dcmp(poly[i].y - p.y);
        int d2 = dcmp(poly[(i + 1) % size].y - p.y);
        if (k > 0 && d1 <= 0 && d2 > 0)wn++;
        if (k < 0 && d2 <= 0 && d1 > 0)wn--;
    }
    if (wn) return 1;//点在内部
    else return 0;//点在外部
}
int convex_hull(point *p, int n, point *ch) {//n个点组成的p[]的凸包,存放在ch[]中
    sort(p, p + n);
    int m = 0;
    for (int i = 0; i < n; i++) {
        while (m > 1 && cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) <= 0) m--;
        ch[m++] = p[i];
    }
    int k = m;
    for (int i = n - 2; i >= 0; i--) {
        while (m > k && cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) <= 0) m--;
        ch[m++] = p[i];
    }
    if (n > 1) m--;
    return m;
}
int half_plane_intersaction(line *l, int n, point *poly) {
    sort(l, l + n);
    int first, last;//构造双端队列
    point *p = new point[n];//p[i]是q[i] & q[i+1]的交点
    line *q = new line[n];//双端队列
    q[first = last = 0] = l[0];
    for (int i = 1; i < n; i++) {
        while (first < last && !P_on_L_left(p[last - 1], l[i].p, l[i].v)) last--;
        while (first < last && !P_on_L_left(p[first], l[i].p, l[i].v)) first++;
        q[++last] = l[i];
        if (fabs(cross(q[last].v, q[last - 1].v)) < eps) {
            last--;
            if (P_on_L_left(l[i].p, q[last].p, q[last].v)) q[last] = l[i];
        }
        if (first < last) p[last - 1] = get_LL_intersection(q[last].p, q[last].v, q[l
ast - 1].p, q[last - 1].v);
    }
    while (first < last && !P_on_L_left(p[last - 1], q[first].p, q[first].v)) last--;
//删除无用平面
    if (last - first <= 1) return 0;
    p[last] = get_LL_intersection(q[last].p, q[last].v, q[first].p, q[first].v);//计算
首尾交点
    int m = 0;
    for (int i = first; i <= last; i++) poly[m++] = p[i];//拷贝结果
    return m;

}

/////////////////////////////////////////////////////
//三维点的定义
/////////////////////////////////////////////////////

struct point3 {
    double x, y, z;
    point3(double x = 0, double y = 0, double z = 0):x(x), y(y) {}
};
typedef point3 vec3;
vec3 operator + (vec3 a, vec3 b) {
    return vec3(a.x + b.x, a.y + b.y, a.z + b.z);
}
vec3 operator - (vec3 a, vec3 b) {
    return vec3(a.x - b.x, a.y - b.y, a.z - b.z);
}
vec3 operator * (vec3 a, double p) {
    return vec3(a.x * p, a.y * p, a.z * p);
}
vec3 operator / (vec3 a, double p) {
    return vec3(a.x / p, a.y / p, a.z / p);
}

/////////////////////////////////////////////////////
//三维点的运算
/////////////////////////////////////////////////////

double dot(vec3 a, vec3 b) { return a.x*b.x + a.y*b.y + a.z * b.z; }//三维点积
double length(vec3 a) { return sqrt(dot(a, a)); }
double angle(vec3 a, vec3 b) {//俩向量夹角
    double ans = dot(a, b) / length(a) / length(b);
    if (dcmp(ans - 1) == 0)ans = 1;
    if (dcmp(ans + 1) == 0)ans = -1;
    return acos(ans);
}
double dis_of_PF(const point3 &p, const point3 &p0, const point3 &n) {
    return fabs(dot(p - p0, n));
}
point3 get_PF_projection(const point3 &p, const point3 &p0, const point3 &n) {
    return p - n*dot(p - p0, n);
}
point3 get_LF_intersaction(point3 &p1, point3 &p2, point3 &p0, vec3 n) {
    vec3 v = p2 - p1;
    double t = (dot(n, p0 - p1) / dot(n, p2 - p1));
    return p1 + v*t;
}
vec3 cross(vec3 a, vec3 b) {
    return vec3(a.y*b.z - a.z*b.y, a.z*b.x - a.x*b.z, a.x*b.y - a.y*b.x);
}
double area2(point3 a, point3 b, point3 c) {
    return length(cross(b - a, c - a));
}
```