1　注意点

1.1　コードを書く前に

- 計算量をきちんと検討した上でコードを書くこと．
- （最初にかかる問題を除いて）少なくとも 1 人には問題の内容を伝えること．
- 問題文は行間もよく読むこと．ただし勝手な思いこみはしないように．
- 時間があれば問題文は最初（背景）から読むほうがよい．

1.2　コードを書いている間

- 配列の要素数は（必要）十分に確保すること．
- 配列の添字の下限値や順序には十分に気を配ること．
- 浮動小数点で表現された数値は単純な比較演算子で比較しないこと．

1.3　提出の前に

- 出力は**必ず**複数人で確認すること（特に英単語のスペル）．
- できれば Sample 以外の入力値（特に境界値）に対しても動作を確認すること．

1.4　浮動小数点のためのマクロ

浮動小数点は誤差の問題があるので単純な比較演算子は使用すべきでない．代わりに次のマクロを使用すること．

```
#define EPSILON (1e-12)
#define iszero(x) (fabs(x) < EPSILON)

#define EQ(x,y) (fabs((x) - (y)) <  EPSILON)
#define NE(x,y) (fabs((x) - (y)) >= EPSILON)
#define LT(x,y) ((x) - (y) <= -EPSILON)
#define LE(x,y) ((x) - (y) <   EPSILON)
#define GT(x,y) ((x) - (y) >=  EPSILON)
#define GE(x,y) ((x) - (y) >  -EPSILON)
```

2　C++標準ライブラリ

2.1　入出力

HEADER FILE(S): iostream, fstream

```
cin >> var;
cout [<< manip] << vals [<< endl];        // manip については下記参照
chr = cin.get();                          // ファイル終端は EOF
cin.getline(var, length[, delim]);        // var は char * 型
getline(cin, var[, delim]);               // var は string 型
cin >> ws;                                // 空白文字を読み飛ばす
bool = cin.eof();
ifstream cin(filename);
ofstream cout(filename);
```

出力操作子（output manipulator）は次のとおり．setw(n) はすぐ次の値に対してのみ有効．ほかの操作子はそれ以降に出力するすべての値に対して効力をもつ．

```
setw(n)
setprecision(n)
setfill(chr)                              // setfill(' ') でもとに戻る
internal, left, right
dec, hex, oct
generic, fixed, scientific
uppercase, nouppercase                    // 16進数(A〜F)などを大文字で表記
showpos, noshowpos                        // 正符号を出力
showpoint, noshowpoint                    // 常に小数点を出力
```

g++ で必要になるコードは次のとおり．

```
ostream& operator <<(ostream &o, string s) { return o << s.c_str(); }

ios& internal(ios &i)
{ i.setf(ios::internal, ios::adjustfield); return i; }
ios& left(ios &i)
{ i.setf(ios::left, ios::adjustfield); return i; }
inline ios& right(ios &i)
{ i.setf(ios::right, ios::adjustfield); return i; }

inline ios& generic(ios &i)
{ i.unsetf(ios::floatfield); return i; }
inline ios& fixed(ios &i)
{ i.setf(ios::fixed, ios::floatfield); return i; }
inline ios& scientific(ios &i)
{ i.setf(ios::scientific, ios::floatfield); return i; }

inline ios& uppercase(ios &i)   { i.setf  (ios::uppercase); return i; }
inline ios& nouppercase(ios &i) { i.unsetf(ios::uppercase); return i; }
inline ios& showpos(ios &i)     { i.setf  (ios::showpos);   return i; }
inline ios& noshowpos(ios &i)   { i.unsetf(ios::showpos);   return i; }
inline ios& showpoint(ios &i)   { i.setf  (ios::showpoint); return i; }
inline ios& noshowpoint(ios &i) { i.unsetf(ios::showpoint); return i; }
```

2.2 文字列

HEADER FILE(S): string

```
string s [= str];
string::iterator p;                    // ランダムイテレータ

s = s1 + s2;
s += str;
s1 == s2, s1 != s2, s1 < s2, ...       // 辞書式比較
bool = s.empty();                      // s == "" と同じ

n = s.length();                        // s.size() でも同じ
str = s.substr(pos[, length]);         // 部分文字列
str = string(first*, last*);           // 部分文字列（イテレータによる指定）
str = string(n, chr);                  // n 文字の chr からなる文字列
cstr = s.c_str();                      // char * に変換

pos = s.find(str[, startpos]);         // str がないときは string::npos（以下同様）
pos = s.rfind(str[, startpos]);
pos = s.find_first_of(str[, startpos]);
pos = s.find_first_not_of(str[, startpos]);
pos = s.find_last_of(str[, startpos]);

s.insert(pos, str);                    // 破壊的（以下同様）
s.erase(pos, length);
s.replace(pos, length, str);

reverse(s.begin(), s.end());           // #include <algorithm> が必要（以下同様）
rotate(s.begin(), s.begin() + n, s.end());
```

1 文字ずつ処理する場合については次のコードを参照.

```
// with an indexer (bracket operator)
for(i = 0; i < s.length(); i++) {
    do_something(s[i]);
}

// with an iterator
for(string::iterator p = s.begin(); p != s.end(); p++) {
    do_something(*p);
}
```

2.3 コンテナ

HEADER FILE(S): list, vector, deque

```
deque<type> c;                          // list, vector も同様
deque<type>::iterator p;                // list は双方向，それ以外はランダム

x = c.front();
x = c.back();
c.push_front(x); c.push_back(x);        // vector は back のみ
c.pop_front(); c.pop_back();            // vector は back のみ

c1 == c2, c1 != c2, c1 < c2, ...        // 辞書式比較
bool = c.empty();
size = c.size();

c.clear();
c.erase(pos*);
c.erase(first*, last*);
c.insert(dest*, x);                     // c に値 x を挿入
c.insert(dest*, first*, last*);         // c に first* から last* の内容を挿入
c.assign(first*, last*);                // c を first* から last* の内容に置き換え

/* list のみ */
ls.remove(x);                           // 値が x である要素をすべて削除
ls.remove_if(func);                     // func(x) が真になる要素をすべて削除
ls.splice(pos*, src-list[, first*, last*]);  // src-list の全部または一部を ls に「移動」
ls.sort();                              // ls の要素を昇順にソート
ls.reverse();                           // ls の要素順を反転させる
ls.unique();                            // 隣接かつ重複する要素を削除

/* deque, vector のみ */
sort(c.begin(), c.end());               // #include <algorithm> が必要
stable_sort(c.begin(), c.end());        // #include <algorithm> が必要

/* 可変長配列 */
vector<type> v(size);                   // 括弧の種類に注意（ちなみに deque でも可）
value = v[index];                       // もちろん代入も可能
vector<type>::iterator p = v.begin();   // ポインタのように扱う場合
v.resize(size);                         // 要素数を変更
```

逆順にソートする場合は次のように reverse を利用するか，sort 関数の引数に greater<*type*>() を追加する（後者の方法は #include <functional> が必要）.

```
// list
ls.sort();
ls.reverse();

// deque, vector
sort(c.begin(), c.end());
reverse(c.begin(), c.end());
```

補足 deque（両端キュー）は両端への要素挿入が $O(1)$，途中への要素挿入が $O(n)$.

2.4 マップ（map）

HEADER FILE(S): map

```
map<key-type,value-type> m;
map<key-type,value-type>::iterator p;        // 双方向

value = m[key];                              // もちろん代入も可能
m.insert(make_pair(key,value));              // key が存在するときは何もしない
m.erase(key);
m.erase(pos*);                               // イテレータ pos が指す要素を削除
m.erase(first*, last*);                      // 範囲指定による削除
pos* = m.find(key);                          // 見つからないときは m.end()
m.clear();
bool = m.empty();
n = m.size();
```

補足 insert の戻り値はイテレータと論理値のペア．イテレータは挿入された要素を指す．論理値は要素が挿入されたかどうか．

2.5 集合（set）

HEADER FILE(S): set

```
set<type> A;
set<type>::iterator p;                       // 双方向

A.insert(x);
A.erase(x);
A.erase(pos*);                               // イテレータ pos が指す要素を削除
A.erase(first*, last*);                      // 範囲指定による削除
pos* = A.find(x);                            // 見つからないときは A.end()
A.clear();

/* 集合演算（#include <algorithm> が必要）
   結果を集合 C に格納する場合 output* は inserter(C, C.begin()) を指定 */
bool = includes(A.begin(), A.end(), B.begin(), B.end());
set_union(A.begin(), A.end(), B.begin(), B.end(), output*);
set_intersection(A.begin(), A.end(), B.begin(), B.end(), output*);
set_difference(A.begin(), A.end(), B.begin(), B.end(), output*);
set_symmetric_difference(A.begin(), A.end(), B.begin(), B.end(), output*);
```

補足 symmetric_difference は対称差集合 $(A \cup B) - (A \cap B)$.

2.6 キューとスタック

HEADER FILE(S): queue , stack

```
queue<type> q;
q.push(x);
q.pop();
x = q.front();
x = q.back();
bool = q.empty();
size = q.size();

stack<type> s;                          // priority_queue も使い方は同じ
s.push(x);
s.pop();
x = s.top();
bool = s.empty();
size = s.size();
```

補足 queue, stack, priority_queue には clear がない.

2.7 ペア（pair）

HEADER FILE(S): utility

```
pair<type1,type2> p;
p = make_pair(x1, x2);
x1 = p.first;
x2 = p.second;
p1 == p2, p1 != p2, p1 < p2, ...        // 辞書式比較（x1 を先に比較）
```

2.8 複素数（complex）

HEADER FILE(S): complex

```
complex<double> z;
complex<double> z(x, y);                // 初期値 x + yi を指定

x = real(z); y = imag(z);               // 実部と虚部
z = polar(r, theta);                    // 極形式表現
z-conj = conj(z);                       // z の共役複素数
abs(z), norm(z), arg(z)                 // 順に z の大きさ，ノルム，偏角

/* 実関数と同じ */
sqrt(z), pow(z,w), exp(z), log(z), log10(z), sin(z), cos(z), tan(z),
sinh(z), cosh(z), tanh(z)
```

補足 演算子は+, -, *, /, ==, !=のみが用意されている．<, >, <=, >=は**用意されていない**ので注意（そもそも複素数に大小関係は存在しない）．また cin, cout を介した入出力においては，例えば $2+3i$ ならば (2,3) のような表現になる．

2.9 アルゴリズム

HEADER FILE(S): `algorithm`

$*$ は双方向またはランダムイテレータ（実際には単方向でよいものも多い）．$**$ は要素がソートされている必要あるもの．† はランダムイテレータ．通常のポインタはランダムイテレータとして指定可能．*output** に `inserter(c, c.begin())` のような挿入イテレータを指定すればコンテナが自動的に拡張される．

引数のうち *pred* は条件判定関数，*cmp* は比較関数．これらの引数には関数オブジェクト（関数へのポインタでも可）を指定する．

```
/* 要素操作 */
fill(first*, last*, value);
fill_n(first*, count, value);
generate(first*, last*, gen-func);
generate_n(first*, count, gen-func);
output-last* = copy(first*, last*, output*);
output-first* = copy_backward(first*, last*, output-last*);
reverse(first*, last*);
output-last* = reverse_copy(first*, last*, output*);
rotate(first*, middle*, last*);
output-last* = rotate_copy(first*, middle*, last*, output*);
new-last* = remove(first*, last*, value);
output-last* = remove_copy(first*, last*, value, output*);
new-last* = remove_if(first*, last*, pred);
output-last* = remove_copy_if(first*, last*, pred, output*);
new-last* = unique(first*, last*);
output-last* = unique_copy(first*, last*, output*);
replace(first*, last*, old-value, new-value);
output-last* = replace_copy(first*, last*, old-value, new-value, output*);
replace_if(first*, last*, pred, new-value);
output-last* = replace_copy_if(first*, last*, pred, new-value, output*);
boundary* = partition(first*, last*, pred);
boundary* = stable_partition(first*, last*, pred);
random_shuffle(first†, last†);

/* 探索・係数 */
pos* = find(first*, last*, value);
pos* = find_if(first*, last*, pred);
pos* = find_first_of(first1*, last1*, first2*, last2*);
pos* = find_end(first1*, last1*, first2*, last2*);
pos* = adjacent_find(first*, last*);
pos* = search(first1*, last1*, first2*, last2*);
pos* = search_n(first1*, last1*, count, value);
pos* = mismatch(first1*, last1*, first2*);
pos* = max_element(first*, last*);
pos* = min_element(first*, last*);
count = count(first*, last*, value);
count = count(first*, last*, pred);

/* ヒープ */
make_heap(first†, last† [, cmp]);
push_heap(first†, last† [, cmp]);
pop_heap(first†, last† [, cmp]);
sort_heap(first†, last† [, cmp]);
```

```
/* ソート・二分探索 */
sort(first†, last† [, cmp]);
stable_sort(first†, last† [, cmp]);
partial_sort(first†, middle†, last† [, cmp]);
partial_sort_copy(first*, last*, output-first†, output-last† [, cmp]);
nth_element(first*, n-th*, last* [, cmp]);
bool = binary_search(first**, last**, value [, cmp]);
pos** = lower_bound(first**, last**, value [, cmp]);
pos** = upper_bound(first**, last**, value [, cmp]);
pos**-pair = equal_range(first**, last**, value [, cmp]);
output-last* = merge(first1**, last1**, first2**, last2**, output* [, cmp]);
inplace_merge(first**, middle**, last** [, cmp]);

/* 集合操作 */
bool = includes(first1**, last1**, first2**, last2** [, cmp]);
output-last* = set_difference(first1**, last1**, first2**, last2**, output* [, cmp]);
output-last* = set_intersection(first1**, last1**, first2**, last2**, output* [, cmp]);
output-last* = set_symmetric_difference(first1**, last1**, first2**, last2**, output* [, cmp]);
output-last* = set_union(first1**, last1**, first2**, last2**, output* [, cmp]);

/* 演算 */
value = accumulate(first*, last*, init-value [, binop]);
value = inner_product(first1*, last1*, first2*, init-value [, binop1, binop2]);
output-last* = transform(first*, last*, output*, unop);
output-last* = transform(first1*, last1*, first2*, output*, binop);
output-last* = partial_sum(first*, last*, output* [, binop]);
output-last* = adjacent_difference(first*, last*, output* [, binop]);

/* その他 */
swap(var1, var2);
value = max(value1, value2);
value = min(value1, value2);
for_each(first*, last*, func);
bool = equal(first1*, last1*, first2*);
bool = lexicographical_compare(first1*, last1*, first2*, last2* [, cmp]);
bool = next_permutation(first*, last*);
bool = prev_permutation(first*, last*);
```

3　構文解析

3.1　基本的なこと

CFG（文脈自由文法）における生成規則の各々に対して次のような関数を作る．

```
int rule(const char *s, int *index, int *result);
    s      -- [IN]     解析する文字列
    index  -- [IN/OUT] 解析位置を表すインデックス
    result -- [OUT]    解析結果
```

この関数の役割は次のとおりである．

- 対応する生成規則の範囲で文字列を解析し，結果を `result` に格納する．
- インデックスは解析が終わった文字の**次の文字に**進める．
- 解析に成功したときは真，失敗したときは偽を返す．

なお，どの文法規則も**左再帰的であってはならない**（回避方法は次ページ）．文法規則が次によって与えられているときの実装例を下に記す．

```
<expr>   ::= <factor> | <factor> "+" <expr>
<factor> ::= [0-9][0-9]*
```

```
int expr(const char *s, int *index, unsigned int *result)
{
    unsigned int r1, r2;
    if(!factor(s, index, &r1)) return FALSE;
    r2 = 0;
    if(s[*index] == '+') {
        ++(*index);
        if(!expr(s, index, &r2)) return FALSE;
    }
    *result = r1 + r2;
    return TRUE;
}

int factor(const char *s, int *index, unsigned int *result)
{
    if(!isdigit(s[*index])) return FALSE;
    *result = 0;
    while(isdigit(s[*index])) {
        *result = (*result) * 10 + (int)(s[*index] - '0');
        ++(*index);
    }
    return TRUE;
}
```

3.2 雑多な注意

左再帰性のある文法を単純に再帰降下型の解析器にかけると無限再帰に陥るため，次のように文法を改める必要がある（α や β が複数ある場合についても同様）.

$$A \to A\alpha \mid \beta \implies \begin{array}{l} A \to \beta A' \\ A' \to \alpha A' \mid \epsilon \end{array}$$

文全体の構文適合性を調べる必要がある場合は，文の最後に余計な文字が存在する場合に対する対策として，次のような開始規則を追加する.

```
<start> ::= <expr> <null-terminator>
```

```
int start(const char *s, int *index, int *result)
{
    if(!expr(s, index, result)) return FALSE;
    return (s[*index] == '\0');
}
```

空白文字が許される場合はトークン境界で空白文字を読み飛ばす．多くの場合は一部の `++(*index)` を次の `while` 文に置き換えればよい．ただし，複雑な場合は空白文字も終端文字として扱い，文法を改めたほうが無難.

```
while(isspace(s[*index])) ++(*index);
```

3.3 複雑な場合

複雑な場合はまず字句解析を行い，トークンを単位として処理する．

```
typedef struct {
    int type;      /* gets one of VARIABLE, INTEGER, OPERATOR, ... */
    int content;
} token_t;

token_t nexttoken(const char *s, int *index)
{
    token_t t;
    while(isspace(s[*index])) ++(*index);

    if(isalpha(s[*index])) {
        t.type = VARIABLE;
        t.content = s[*index];
    } else if(isdigit(s[*index])) {
        t.type = INTEGER;
        t.content = 0;
        while(isdigit(s[*index])) {
            t.content = t.content * 10 + (int)(s[*index] - '0');
            ++(*index);
        }
    } else if(s[*index] == '+' || s[*index] == '-') {
        t.type = OPERATOR;
        t.content = s[*index];
    } else if(s[*index] == '\0') {
        t.type = TERMINATOR;
    } else {
        t.type = ERROR;
    }

    return t;
}
```

4　グラフ理論

4.1　トポロジカル・ソート

DAG（有向非循環グラフ）$G = (V, E)$ において，点 u から点 v への有向路があるときは u が先に来るように点の集合 V を並べる．

```
list<int> sorted;

void topological_sort(int current)
{
    for(int i = 0; i < N; i++)
        if(is_adjacent(current, i)) topological_sort(i);
    sorted.push_front(current);
}
```

4.2　最小全域木

Kruskal's Algorithm　グラフを隣接リストで管理しているときに好都合．

1. 辺の集合 E を重みが小さい順に並びかえる．
2. E に含まれる辺を順番に調べる．その辺を最小全域木に加えても閉路が生じないときは，その辺を最小全域木に加える．

Prim's Algorithm　グラフを隣接行列で管理しているときに好都合．最短路探索の Dijkstra 法とよく似ている．

1. 最小全域木の部分木を構成する点の集合 S を空集合で初期化する．
2. 任意の 1 点を S に加える．
3. S に含まれる点から S に含まれない点に接続する辺の中で重みが最小である辺を選び，その辺の行き先を S に加える．
4. すべての点が S に含まれるまで 3. の動作を繰り返す．

4.3　最短路

Dijkstra's Algorithm　計算量 $O(|V|^2)$．開始点 s からの目的点 t への距離を計算．負の重みがあってはならない．

Bellman-Ford Algorithm　計算量 $O(|V| \cdot |E|)$．開始点 s からすべての点 v への距離を計算．負の重みがあってもよい．$d(v)$ は開始点 s から点 v までの距離，$p(v)$ は点 v の先行点．

1. $d(s) \leftarrow 0$，$d(v) \leftarrow \infty \ \ \forall v \neq s$.
2. 次の操作を $(|V| - 1)$ 回繰り返す．
   - 各辺 (u, v)（無向グラフの場合は (u, v)，(v, u) の両方）に対して
     $d(v) > d(u) + a_{u,v}$ ならば $d(v) \leftarrow d(u) + a_{u,v}$，$p(v) \leftarrow u$.
3. $d(v) > d(u) + a_{u,v}$ を満たす辺 (u, v) があるときは負の閉路が存在する．
4. そうでなければ $d(v)$ が最短距離．$p(v)$ をたどれば最短路．

Floyd-Warshall Algorithm　計算量 $O(|V|^3)$．任意の 2 点 u，v 間の距離を一度に計算．負の重みがあってもよい．$d(u,v)$ は 2 点 u，v 間の距離，$w(u,v)$ は辺 (u,v) の重み，$p(u,v)$ は点 u から点 v への路における点 v の先行点．

1. $d(u,v) \leftarrow a_{u,v}$ として初期化．
2. 次の操作を各点 $k \in V$ について繰り返す．
   - 各 $(i,j) \in V^2$ に対して，
     $d(i,j) > d(i,k) + d(k,j)$ ならば $d(i,j) \leftarrow d(i,k) + d(k,j)$，$p(i,j) \leftarrow k$.
3. $d(v,v) < 0$ を満たす点 v があるときは負の閉路が存在する．
4. そうでなければ $d(u,v)$ が最短距離．$p(u,v)$ をたどれば最短路．

4.4　最大流

計算量 $O(|V| \cdot |E|^2)$．無向グラフのときは有向グラフに変換しておく．

1. Dijkstra のアルゴリズムなどで開始点から目的点までの最短路（ここでは辺の数が最小である路）を探索する．
2. 1. で得られた路に流せるだけの物資を流す．
3. 流した物資の量を x とする．物資を流した辺の容量を x だけ減らし，代わりにそれと反対向きの辺の容量を x だけ増やす．
4. 開始点から目的点までの路がなくなるまで繰り返す．

手順 1. でラベリング法を利用した場合は計算量が $O(|E|^2 \cdot U)$ になる（U は各辺における初期容量の最大値）．

4.5　最小費用流

Busaker and Gowen's Algorithm　供給点，需要点がともに 1 つである場合は最大流問題と同様の手法で解ける．ただし次の点が異なる．

- 辺の数が最小となる路の代わりに，物資を流すのにかかる費用が最小となる路を探索する．
- 反対向きの辺については費用を符号を反転させる（つまり負にする）．負の重みが生じるため Dijkstra 法は適用できないので注意．

Cycle Cancelling Algorithm　制約条件を満たす適当なフローがわかっている場合は，それを初期フローとする．フローに負の閉路が存在するとき，その閉路に対して流せるだけの物資を流す．負の閉路がなくなったらそれが最小費用流．

4.6 強連結成分

有向グラフの 2 点 u, v に対して，$u \to v$ と $v \to u$ の両方の有向路が存在するとき u と v は**強連結**であるという．強連結である頂点の極大部分集合 $U \subseteq V$ を**強連結成分**という．

探索アルゴリズム　まず G の各頂点を時刻つき深さ優先探索でたどる．次に G^T（G の辺の向きを反転したグラフ）の各頂点を，G を深さ優先探索したときの終了時刻の降順に並べる．今度はその順番で G^T の各頂点を深さ優先探索．そのときに一度の探索で到達した点の集合が 1 つの強連結成分となる．

```
int  finish[N], index[N];
bool visited[N];

void dfs(int current)
{
    static int t = 0;
    ++t;
    for(int i = 0; i < N; i++)
        if(is_adjacent(current, i)) dfs(i);
    finish[current] = ++t;
}

bool finish_descending(int i, int j)
{
    return finish[i] > finish[j];
}

void strongly_connected(void)
{
    for(int i = 0; i < N; i++) {
        fisnish[i] = 0; index[i] = i; visited[i] = false;
    }

    for(int i = 0; i < N; i++) if(finish[i] == 0) dfs(i);
    sort(index, index + N, finish_descending);

    for(int i = 0; i < N; i++) {
        if(!visited[index[i]]) {
            visited[index[i]] = true;
            set<int> R;
            strongly_connected_iter(index[i], R);
            DO_SOMETHING(R);
        }
    }
}

void strongly_connected_iter(int current, set<int> &R)
{
    R.insert(current);
    for(int i = 0; i < N; i++) {
        if(is_adjacent(current, index[i]) && !visited[index[i]]) {
            visited[index[i]] = true;
            strongly_connected_iter(index[i], R);
        }
    }
}
```

5　計算幾何

$\mathrm{P_i}$ は 2 次元 Euclid 空間上の点で $\mathrm{P_i} = (x_i, y_i)$．また $\mathrm{P_i} \times \mathrm{P_j} \equiv x_i y_j - x_j y_i$．また必要に応じて，点を表現する型として次の `point_t`，`fpoint_t` を使用する．

```
struct point_t  { int x, y; };
struct fpoint_t { double x, y; };
```

5.1　曲がる方向の判定

経路 $\mathrm{P_0} \to \mathrm{P_1} \to \mathrm{P_2}$ に対し，$(\mathrm{P_2} - \mathrm{P_1}) \times (\mathrm{P_1} - \mathrm{P_0}) < 0$ ならば $\mathrm{P_1}$ で左に曲がっている．

5.2　CCW

```
int ccw(point_t p0, point_t p1, point_t p2)
{
    int dx1, dx2, dy1, dy2;
    dx1 = p1.x - p0.x; dy1 = p1.y - p0.y;
    dx2 = p2.x - p0.x; dy2 = p2.y - p0.y;
    if(dx1 * dy2 > dy1 * dx2) return +1;
    if(dx1 * dy2 < dy1 * dx2) return -1;
    if(dx1 * dx2 < 0 || dy1 * dy2 < 0) return -2;
    if(dx1*dx1 + dy1*dy1 < dx2*dx2 + dy2*dy2) return +2;
    return 0;
}
```

5.3　線分の交差判定（CCW あり）

```
/* (p1,p2) と (p3,p4) がそれぞれ線分 */
bool intersect(point_t p1, point_t p2, point_t p3, point_t p4)
{
    return (ccw(p1, p2, p3) * ccw(p1, p2, p4)) <= 0 &&
           (ccw(p3, p4, p1) * ccw(p3, p4, p2)) <= 0;
}
```

5.4 線分の交差判定（CCW なし）

1. 2 つの Bounding Box が重なっていることを確かめる．
2. $(P_3 - P_1) \times (P_2 - P_1)$ と $(P_4 - P_1) \times (P_2 - P_1)$ を計算する．
3. 2 つの外積が異符号ならば，線分 P_1P_2 と線分 P_3P_4 は交差している．
4. 外積が 0 になる場合は，P_3，P_4 のいずれかが線分 P_1P_2 に重なっているか，または一方の線分の長さが 0.

5.5 疑似 arctan

atan2(x,y) に対して単調増加する関数．atan2 よりも高速．ただし範囲は $[-\pi, \pi]$ ではなく $[-2, 2]$．引数は atan2 と同様に y，x の順なので注意すること．

```
double pseudo_atan2(double y, double x)
{
    double t, ax, ay;
    ax = fabs(x); ay = fabs(y);
    t = (ax + ay == 0.0) ? 0.0 : (y / (ax + ay));
    if(x < 0.0) t = (y < 0.0 ? -2.0 : 2.0) - t;
    return t;
}
```

5.6 三角形の面積

$\triangle OP_1P_2 = \frac{1}{2}|x_1y_2 - x_2y_1|$．ただし O は原点 $(0, 0)$.

```
double area_triangle(fpoint_t p0, fpoint_t p1, fpoint_t p2)
{
    double w = (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
    return fabs(w) / 2.0;
}
```

5.7 凸多角形の面積

点の配列 p は時計回り（または反時計回り）の順序で与えること．

```
double area_convex(const vector<fpoint_t> &p)
{
    int n = p.size();
    double S = 0.0;
    for(int i = 1; i < n; i++) S += p[i-1].x * p[i].y - p[i].x * p[i-1].y;
    S += p[n-1].x * p[0].y - p[0].x * p[n-1].y;
    return fabs(S) / 2.0;
}
```

5.8 三角形における点の包含判定（CCW 方式）

```
bool inside_triangle(fpoint_t p0, fpoint_t p1, fpoint_t p2, fpoint_t q)
{
    p0.x -= q.x; p0.y -= q.y;
    p1.x -= q.x; p1.y -= q.y;
    p2.x -= q.x; p2.y -= q.y;

    double v1 = p0.x * p1.y - p1.x * p0.y;
    double v2 = p1.x * p2.y - p2.x * p1.y;
    double v3 = p2.x * p0.y - p0.x * p2.y;

    /* 辺の上 */
    if(iszero(v1) || iszero(v2) || iszero(v3)) return true;
    /* 辺の上ではない */
    return (v1 > 0 && v2 > 0 && v3 > 0) || (v1 < 0 && v2 < 0 && v3 < 0);
}
```

5.9 三角形における点の包含判定（面積法）

```
bool inside_triangle(fpoint_t p0, fpoint_t p1, fpoint_t p2, fpoint_t q)
{
    double r = 0.0;
    r += area_triangle(p0, p1, q);
    r += area_triangle(p1, p2, q);
    r += area_triangle(p2, p0, q);
    r -= area_triangle(p0, p1, p2);
    return iszero(r);
}
```

5.10 最近点対問題

1. すべての点を x 座標の順に並びかえる．
2. 点集合を左右半分に分割する．
3. このとき点集合全体に対する最近点対は次のいずれか．
   - 左側の集合の最近点対
   - 右側の集合の最近点対
   - 左右両方の集合にそれぞれ 1 点ずつ含まれるような 2 点の最近点対

   これらのうちで問題になるのは一番下の場合．
4. 左側，右側の各集合における最近点対を（再帰的に）探索する．得られた点対のより近いほうの距離を m とする．
5. 両側にまたがった 2 点について距離 m 未満の点対を探す．
   - そのとき y 座標の値について点をソートしておくと効率がよい（左右に分割しているのでマージソートが可能）．

5.11 多角形における点の包含判定（CCW あり）

```
/* the number large enough but INT_MAX is not acceptable */
#define LARGE_INT (46340)

int inside(point_t p[N], point_t t)
{
    int i, count = 0, j = 0;
    line_t lt, lp;
    p[0] = p[N]; p[N+1] = p[1];
    lt.p1 = t; lt.p2 = t; lt.p2.x = LARGE_INT;
    for(i = 1; i <= N; i++) {
        lp.p1 = p[i]; lp.p2 = p[i];
        if(cc2(lt.p1, lt.p2, p[i]) != 0) {
            if(i == j + 1) {
                lp.p2 = p[j];
                if(intersect(lp, lt)) count++;
            }
            else {
                if(ccw(lt.p1, lt.p2, p[i]) * ccw(lt.p1, lt.p2, p[j]) < 0)
                    count++;
            }
            j = i;
        }
    }
    return count % 2;
}
```

5.12 凸包

ここでは周の長さを計算している.

```
#define where if

double dist2(point_t p1, point_t p2)
{
    return (p2.x - p1.x) * (p2.x - p1.x) + (p2.y - p1.y) * (p2.y - p1.y);
}

double convex_hull(const vector<fpoint_t> &p)
{
    int start = 0;
    for(int i = 1; i < p.size(); i++) {
        if(LT(p[start].y, p[i].y)) start = i;
        if(EQ(p[start].y, p[i].y) && LT(p[start].x, p[i].x)) start = i;
    }

    int curr = start;
    double len = 0.0, theta = 0.0;

    while(true) {
        double phi_min = 2.0 * M_PI;
        int next = curr;
        for(int i = 0; i < p.size(); i++) where (i != curr) {
            double phi;
            phi = atan2(p[i].y - p[curr].y, p[i].x - p[curr].x) - theta;
            while(LT(phi, 0.0)) phi += 2.0 * M_PI;
            if(EQ(phi, 0.0)) continue;

            if(EQ(phi, phi_min)) {
                if(LT(dist2(p[curr], p[next]), dist2(p[curr], p[i]))) {
                    next = i;
                }
            } else if(LT(phi, phi_min)) {
                next = i;
                phi_min = phi;
            }
        }
        len += sqrt(dist2(p[curr], p[next]));
        theta = atan2(p[next].y - p[curr].y, p[next].x - p[curr].x);
        curr = next;
        if(curr == start) break;
    }

    return len;
}
```

6　整数論

6.1　拡張最大公約数

自然数 a，b の最大公約数 d と $ax + by = d$ を満たす x，y の組を計算.

```
int exgcd(int a, int b, int &x, int &y)
{
    if(b == 0) {
        x = 1; y = 0;
        return a;
    } else {
        int d = exgcd(b, a % b, y, x);
        y -= a / b * x;
        return d;
    }
}
```

6.2　中国の剰余定理

互いに素な m_1，m_2 に対して $x \equiv a_1 \pmod{m_1}$，$x \equiv a_2 \pmod{m_2}$ を同時に満たす $0 \leq x < m_1 m_2$ を求める．m_1 と m_2 が互いに素でないときは -1 を戻す.

```
int chinese_reminder(int a1, int a2, int m1, int m2)
{
    int x, y;
    if(exgcd(m1, m2, x, y) != 1) return -1;
    x = a1 + (a2 - a1) * x * m1;
    while(x < 0) x += m1 * m2;
    return x;
}
```

6.3　Josephus 数

n は全体の人数，p は何人ごとに殺すか．最後に残る人物が戻り値.

```
int josephus(int n, int p)
{
    int k = p - 1;
    while (k < (p - 1) * n) k = (p * k) / (p - 1) + 1;
    return (p * n - k);
}
```

7　公式

7.1　三角関数

基本公式

$$\sin^2\theta + \cos^2\theta = 1$$
$$1 + \tan^2\theta = \frac{1}{\cos^2\theta}$$

余弦定理，正弦定理

$$a^2 = b^2 + c^2 - 2bc\cos A,\quad b^2 = c^2 + a^2 - 2ca\cos B,\quad c^2 = a^2 + b^2 - 2ab\cos C$$
$$\cos A = \frac{b^2 + c^2 - a^2}{2bc},\quad \cos B = \frac{c^2 + a^2 - b^2}{2ca},\quad \cos C = \frac{a^2 + b^2 - c^2}{2ab}$$
$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R \quad (R\text{ は外接円の半径})$$
$$S = \frac{abc}{4R} = \frac{2r}{a+b+c} \quad (R,\ r\text{ はそれぞれ外接円，内接円の半径})$$

加法定理

$$\sin(\alpha+\beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta$$
$$\cos(\alpha+\beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$$
$$\tan(\alpha+\beta) = \frac{\tan\alpha + \tan\beta}{1 - \tan\alpha\tan\beta}$$

倍角，半角

$$\sin 2\theta = 2\sin\theta\cos\theta$$
$$\cos 2\theta = \cos^2\theta - \sin^2\theta = 2\cos^2\theta - 1 = 1 - 2\sin^2\theta$$
$$\tan 2\theta = \frac{2\tan\theta}{1 - \tan^2\theta}$$
$$\sin 3\theta = 3\sin\theta - 4\sin^3\theta$$
$$\cos 3\theta = 4\cos^3\theta - 3\cos\theta$$
$$\sin^2\frac{\theta}{2} = \frac{1 - \cos\theta}{2}$$
$$\cos^2\frac{\theta}{2} = \frac{1 + \cos\theta}{2}$$

合成公式

$$a\sin\theta + b\cos\theta = \sqrt{a^2 + b^2}\sin(\theta + \alpha) \quad (\alpha\text{ は }x\text{ 軸と }(b, a)\text{ のなす角})$$

和積，積和変換

$$\sin\alpha\cos\beta = \frac{1}{2}\{\sin(\alpha+\beta)+\sin(\alpha-\beta)\}$$

$$\cos\alpha\cos\beta = \frac{1}{2}\{\cos(\alpha+\beta)+\cos(\alpha-\beta)\}$$

$$\sin\alpha\sin\beta = -\frac{1}{2}\{\cos(\alpha+\beta)-\cos(\alpha-\beta)\}$$

$$\sin A+\sin B = 2\sin\frac{A+B}{2}\cos\frac{A-B}{2}$$

$$\sin A-\sin B = 2\cos\frac{A+B}{2}\sin\frac{A-B}{2}$$

$$\cos A+\cos B = 2\cos\frac{A+B}{2}\cos\frac{A-B}{2}$$

$$\cos A-\cos B = -2\sin\frac{A+B}{2}\sin\frac{A-B}{2}$$

その他

$$1+\sin\theta = \left(\cos\frac{\theta}{2}+\sin\frac{\theta}{2}\right)^2$$

$$1-\sin\theta = \left(\cos\frac{\theta}{2}-\sin\frac{\theta}{2}\right)^2$$

$$\frac{1-\cos\theta}{\sin\theta} = \frac{\sin\theta}{1+\cos\theta} = \tan\frac{\theta}{2}$$

7.2 幾何学

三角不等式 $a < b+c$ かつ $b < c+a$ かつ $c < a+b$.

- $a^2 < b^2+c^2$ かつ $b^2 < c^2+a^2$ かつ $c^2 < a^2+b^2$ ならば鋭角三角形.
- $a^2 = b^2+c^2$ または $b^2 = c^2+a^2$ または $c^2 = a^2+b^2$ ならば直角三角形.
- $a^2 > b^2+c^2$ かつ $a < b+c$（a は最大辺）ならば鈍角三角形.

球の表面積，体積

$$S = 4\pi r^2,\quad V = \frac{4}{3}\pi r^3$$

点と直線との距離 点 $\mathrm{P}(x_0, y_0)$ から直線 $ax+by+c=0$ への距離 d は

$$d = \frac{|ax_0+by_0+c|}{\sqrt{a^2+b^2}}$$

円の方程式 (x_1, y_1)，(x_2, y_2) を直径の両端とする円の方程式は

$$(x-x_1)(x-x_2)+(y-y_1)(y-y_2) = 0$$

円の接線 円 $(x-x_0)^2+(y-y_0)^2 = r^2$ の $\mathrm{P}(x', y')$ を接点とする接線の方程式は

$$(x'-x_0)(x-x_0)+(y'-y_0)(y-y_0) = r^2$$

円 $x^2+y^2+2Ax+2By+C=0$ の $\mathrm{P}(x', y')$ を接点とする接線の方程式は

$$x'x+y'y+Ax'+Ax+By'+By+C = 0$$

2 つの円の関係　2 つの円の半径が r_1，r_2，中心間の距離が d のとき

$$\begin{cases} d < r_1 + r_2 \text{ のとき} & \text{交わっている} \\ d = r_1 + r_2 \text{ のとき} & \text{接している} \\ d > r_1 + r_2 \text{ のとき} & \text{離れている} \end{cases}$$

2 つの円の共通弦　C_1; $x^2 + y^2 + Ax + By + C = 0$，$C_2$; $x^2 + y^2 + A'x + B'y + C' = 0$ の差 $(A - A')x + (B - B')y + (C - C')$ は

$$\begin{cases} C_1 \text{ と } C_2 \text{ が交わるとき} & C_1 \text{ と } C_2 \text{ の共通弦} \\ C_1 \text{ と } C_2 \text{ が接するとき} & C_1 \text{ と } C_2 \text{ の共通接線} \\ C_1 \text{ と } C_2 \text{ が離れるとき} & C_1 \text{ と } C_2 \text{ の接点への長さが等しい点の集合} \end{cases}$$

楕円の方程式　長径 a，短径 b の楕円の方程式は

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \quad (a > b > 0)$$

焦点の座標は $(\pm\sqrt{a^2 - b^2}, 0)$.

7.3　微積分

積分公式

$$\int_a^b f(x)\,dx = \int_\alpha^\beta f(x)\frac{dx}{dt}\,dt = \int_\alpha^\beta f(\phi(t))\phi'(t)\,dt \quad (x = \phi(t),\ a = \phi(\alpha),\ b = \phi(\beta))$$

$$\int f'(x)g(x)\,dx = f(x)g(x) - \int f(x)g'(x)\,dx$$

$$\int \frac{1}{\cos^2 x}\,dx = \tan x$$

$$\int \frac{1}{\sin^2 x}\,dx = -\frac{1}{\tan x}$$

$$\int a^x\,dx = \frac{a^x}{\ln a}$$

$$\int \{f(x)\}^a f'(x)\,dx = \frac{1}{a+1}\{f(x)\}^{a+1} \quad (a \neq 1)$$

$$\int \frac{f'(x)}{f(x)}\,dx = \ln|f(x)|$$

道のり　$x = \varphi(t)$，$y = \psi(t)$ に対し，$t = \alpha$ から $t = \beta$ まで（または $x = a$ から $x = b$ まで）の道のりは

$$\int_\alpha^\beta \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}\,dt = \int_a^b \sqrt{1 + \left(\frac{dx}{dt}\right)^2}\,dx$$

7.4 その他

解と係数の関係 $ax^2 + bx + c = 0$ の解が $x = \alpha, \beta$ のとき

$$-\frac{b}{a} = \alpha + \beta, \quad \frac{c}{a} = \alpha\beta$$

$ax^3 + bx^2 + cx + d = 0$ の解が $x = \alpha, \beta, \gamma$ のとき

$$-\frac{b}{a} = \alpha + \beta + \gamma, \quad \frac{c}{a} = \alpha\beta + \beta\gamma + \gamma\alpha, \quad -\frac{d}{a} = \alpha\beta\gamma$$

級数公式

$$\sum_{k=1}^{n} k = \frac{1}{2}n(n+1)$$

$$\sum_{k=1}^{n} k^2 = \frac{1}{6}n(n+1)(2n+1)$$

$$\sum_{k=1}^{n} k^3 = \frac{1}{4}n^2(n+1)^2$$

$$\sum_{k=1}^{n} ar^{k-1} = \frac{a(r^n - 1)}{r - 1}$$

二項定理

$$(a+b)^n = \sum_{r=0}^{n} {}_n\mathrm{C}_r \, a^{n-r} \, b^r$$

ド・モアブルの定理

$$(\cos\theta + i\sin\theta)^n = \cos n\theta + i\sin n\theta$$

8　トラブルシューティング

プログラムが正常に動作しない

次の点を確認してください.

1. 変数はきちんと初期化されていますか.
2. 配列の要素数は不足していませんか.
3. 配列の添字は 0-base または 1-base などで統一されていますか.
4. 多次元配列における添字の順序は統一されていますか.

制限時間内に計算が終わらない

次の点を確認してください.

1. 計算量は正しく見積もりましたか.
2. プログラムが無限ループに陥ることはありませんか.
3. 同じような計算を何度も行っていませんか.
4. もっと積極的に探索枝を狩ることはできませんか.
5. 動的計画法のようなもっと効率的な計算法はありませんか.

審判団に受理されない（Wrong Answer が戻ってくる）

次の点を確認してください.

1. コード中に記述した英単語のスペルは合っていますか.
2. Sample Input 以外のデータに対して正常に動作しますか.
3. 境界値に対して正常に動作しますか.
4. 問題文は隅から隅まで読みましたか．行間は正確に読めていますか.
5. 問題文に書かれていないことを勝手に仮定していませんか.

頭が熱をもってしまった

次のいずれか（または両方）を実行してください.

1. 休憩スペースで飲食物を口にしながら頭を休めます.
2. トイレで用を足し，気持ちを落ち着けます.