C++講習解説

松野研　M2　小池椋介

C++講習の概要

- https://github.com/r-koike/kosyu　にアクセス.

「良いプログラム」とは？

- バグらない（数値計算で誤差を出さない）．
- 時間計算量が小さい（計算が速い）．

ロボットやそのシミュレーションで重要.
→このC++講習のテーマ.

- 空間計算量が小さい（メモリを消費しない）．

- 可読性，拡張性が高い.
  →ほかの人や未来の自分が見たときにすぐに理解し，改造できる.

チームでの開発で重要.

高速なプログラムを作る時の一般的なコツ

- 二分探索やその応用で，時間計算量を対数オーダー落とせることがある.
- 時間計算量は空間計算量で置き換えられることがある.

各問題のテーマ

- 問題A　→　イントロダクション.
- 問題B
- 問題C

　　バグらないコードを書く.

- 問題D
- 問題E

　　時間計算量が小さい(速い)コードを書く.

- 問題F
- 問題G

　　C++の言語仕様を理解する.

問題B

B: 数値処理

自然数Nが与えられます． 整数Aについて，$N!$が3桁以下なら$A = N!$として，3桁よりも大きいならAは$N!$の左から3文字とします． つまり，$1 \leq A \leq 999$です． Aを出力してください．

制約

- $1 \leq N \leq 100$
- 入力は全て整数
- 上記制約のもとで$1 \leq N! < 10^{200}$であることが保証される

入力

入力は以下の形式で標準入力から与えられます．

> N

出力

整数Aを出力してください．

整数型で大きな数を表現できるか？

https://project-flora.net/2015/07/21/cc%E3%81%AB%E3%81%8A%E3%81%91%E3%82%8B%E6%95%B4%E6%95%B0%E5%9E%8B%E3%81%AB%E3%81%AF%E6%B0%97%E3%82%92%E3%81%A4%E3%81%91%E3%82%88/

各変数と実際のビットサイズ表

処理系（OSみたいなもの）によって違う.
筆者の環境（Win10, g++ 9.2.0）ではLLP64.

整数型	LP32	ILP32	LLP64	LP64	ILP64
char	8	8	8	8	8
short	16	16	16	16	16
int	16	32	32	32	64
long	32	32	32	64	64
long long	64	64	64	64	64

だいたい
16bit整数 → $-32768 \sim 32767$
32bit整数 → $-10^9 \sim 10^9$
64bit整数 → $-10^{18} \sim 10^{18}$
を表現できる.

これより大きい数の表現には向かない.

解答例①　精確さは捨て，上位の数字だけを保持する

だいたい

- float　→　$-10^{38} \sim 10^{38}$
- double　→　$-10^{308} \sim 10^{308}$
- long double　→　$-10^{4932} \sim 10^{4932}$

を表現できる.

doubleで計算すればよい.

解答例①　小数で計算する


```cpp
#include <cmath>
#include <cstdio>

int main() {
    int n;
    scanf("%d", &n);

    // 階乗をdouble型で計算する
    double kaijo = 1;
    for (int i = 1; i <= n; i++) {
        kaijo *= i;
    }

    // kaijoの左から3桁を求める
    // adが3桁よりも大きい限りは続行する
    double double_a = kaijo;
    while (floor(log10(double_a) + 1) > 3) {
        double_a /= 10;
    }

    // double型をint型へ暗黙に変換するとき小数点以下は切り捨てられる
    int a = double_a;
    printf("%d\n", a);

    return 0;
}
```

解答例② logで計算する


```cpp
#include <algorithm>
#include <cmath>
#include <cstdio>

int main() {
    int n;
    scanf("%d", &n);

    // 階乗のlogをdouble型で計算する
    double log_kaijo = 0;
    for (int i = 1; i <= n; i++) {
        log_kaijo += log10(i);
    }

    // 階乗の仮数部と指数部をそれぞれdouble型，int型で計算する
    double base_kaijo = pow(10, log_kaijo - (int)log_kaijo);
    int index_kaijo = (int)log_kaijo;
    for (int i = 0; i < std::min(index_kaijo, 2); i++) {
        base_kaijo *= 10;
    }

    // double型をint型へ丸めこむ
    // base_kaijoは119.99999999999などになる可能性があるので，
    // 微小数値を足してから暗黙的にint型に変換する
    int a = base_kaijo + 0.0000000001;
    printf("%d\n", a);

    return 0;
}
```

$x = N!$ とすると，

$$x = 10^{\log x} = 10^{\sum_{i=1}^{N} \log i}$$

となる．

$\sum_{i=1}^{N} \log i$ はそれほど大きくない値になるので計算可能．その小数部分を使えば，求めたい x の仮数部(=上位の桁)がわかる．

解答例③　長さ200の配列を200桁の整数とみて，ゴリ押しで整数計算する（非推奨）


```cpp
#include <cstdio>

// 階乗計算のためには何桁の整数まで保持すれば十分か
const int MAX_KETA = 200;

// kaijo[i][j]はiの階乗の右から(j+1)個めの数字とする
int kaijo[200][MAX_KETA];

int main() {
    int n;
    scanf("%d", &n);

    // 0の階乗の1桁目は1とする
    kaijo[0][0] = 1;

    for (int i = 1; i <= n; i++) {
        // ここでは(i-1の階乗)×iを計算する
        // そのために，(i-1の階乗)を各桁に分けた物に対して全部にiを掛け，
        // 掛け算の筆算のように加算していく
        for (int j = 0; j < MAX_KETA; j++) {
            // xは何桁になるかわからない
            int x = kaijo[i - 1][j] * i;

            // このkに関するfor文の中身はxの桁数回だけ実行される
            for (int k = 0; x > 0; k++) {
                kaijo[i][j + k] += x % 10;
                x /= 10;

                // kaijoの配列は10以上の数字を保持しないようにする
                x += kaijo[i][j + k] / 10;
                kaijo[i][j + k] %= 10;
            }
        }
    }
```

```cpp
    int ret = 0;
    int keta = 0;
    bool zeroPadding = true;
    for (int i = MAX_KETA - 1; i >= 0 && keta < 3; i--) {
        // 最初の方はゼロで埋まっているはず(ゼロパディング)なので，
        // 最初に0以外の数字が来るまでスルーする
        if (zeroPadding && kaijo[n][i] == 0) {
            continue;
        }
        zeroPadding = false;

        ret *= 10;
        ret += kaijo[n][i];
        keta++;
    }

    printf("%d\n", ret);

    return 0;
}
```

20桁の整数なら，長さ20の配列に押し込む．

21! ＝

5	1	0	9	0	9	4	2	1	7	1	7	0	9	4	4	0	0	0	0

解答例④　計算時間O(1)の最強の方法


```
#include <cstdio>

int ret[101] = {1,   1,   2,   6,   24,  120, 720, 504, 403, 362, 362, 399, 479, 622, 871,
                130, 209, 355, 640, 121, 243, 510, 112, 258, 620, 155, 403, 108, 304, 884,
                265, 822, 263, 868, 295, 103, 371, 137, 523, 203, 815, 334, 140, 604, 265,
                119, 550, 258, 124, 608, 304, 155, 806, 427, 230, 126, 710, 405, 235, 138,
                832, 507, 314, 198, 126, 824, 544, 364, 248, 171, 119, 850, 612, 447, 330,
                248, 188, 145, 113, 894, 715, 579, 475, 394, 331, 281, 242, 210, 185, 165,
                148, 135, 124, 115, 108, 103, 991, 961, 942, 933, 933};

int main() {
    int n;
    scanf("%d", &n);
    printf("%d\n", ret[n]);
    return 0;
}
```

計算時間を減らす代わりに，使用メモリが増える．

時間計算量を**空間計算量**に変換した．

これを**メモ化**という．

問題C

C: 二次方程式

小数$A,\ B,\ C$が与えられます． Xに関する二次方程式$AX^2+BX+C=0$の実数解を小さい順に2つ出力してください． ただし小数のジャッジのため，答えの数値を有効数字8桁の指数表記で出力してください．

制約

- $0 < A,\ B,\ C \leq 10$
- $B^2-4AC>0$
- 入力は指数表記された有効数字8桁の小数
- 上記の制約のもとで二次方程式の実数解が2つ存在することが証明される
- 上記の制約のもとで解Xを8桁目まで正確に計算する方法が存在する

入力

入力は以下の形式で標準入力から与えられます．

> $A\ B\ C$

出力

半角スペース区切りで以下のように出力してください． ただし$X_1 < X_2$とします． また，有効数字8桁の指数表記で出力してください．

どのようなときに計算誤差が生まれるか？


```cpp
1  #include <cmath>
2  using namespace std;
3  const double INF = 1e200;
4
5  // |x * y + x / y| を計算する関数
6  double function(double x, double y) {
7      // ゼロ割りを防ぐ
8      if (abs(y) < 0.0001) {
9          return INF;
10     }
11
12     double a0 = x * y;
13     double a1 = x / y;
14     double ret = a0 + a1;
15
16     return abs(ret);
17 }
```

x,yはあらゆる小数を想定する.
12～14行目で，一番誤差が
生じやすい計算はどれか？

14行目の足し算（引き算）が圧倒的に危険.

12，13桁目はオーバーフローにさえ気を付ければよい.

近い数値同士の足し算（引き算）は危険

floatは約7桁の小数を保持できる.

- float a = 3.141592…
- float b = 3.141583…
- float c = a – b

このとき,

- c = 9.… * 10^-6

になり，実質1桁の情報しか保持されない（桁落ち誤差）.
プログラム上の足し算は引き算にもなり得ると考え，注意するべき.

実用的には,
二次方程式の解の公式，行列式，差分から微分を求める式,
の計算などでよく起こる.

桁落ち誤差の対処法

- ない.

二次方程式に対しては，計算の順番を入れ替えることで奇跡的に回避できる.

二次方程式の解は,

$$x=\frac{-b-\sqrt{b^2-4ac}}{2a},\frac{-b+\sqrt{b^2-4ac}}{2a}$$

ここの桁落ちは結果に直接影響するので，対処する必要がある.

二つ目だけを変形すると,

$$x=\frac{-b-\sqrt{b^2-4ac}}{2a},\frac{-2c}{b+\sqrt{b^2-4ac}}$$

ここの桁落ちは問題にはならない．もしb^2-4acで桁落ちするなら，$\sqrt{b^2-4ac}$はbよりもかなり小さくなる．よって分母である$b+\sqrt{b^2-4ac}$はbに大きく支配されるので，全体として誤差はない.

b>0という仮定があるので同符号同士の計算になり，桁落ちしない.

解答例


```cpp
#include <cmath>
#include <cstdio>

int main() {
    double a, b, c;
    scanf("%lf%lf%lf", &a, &b, &c);

    double d = sqrt(b * b - 4 * a * c);
    double x1 = (-b - d) / (2 * a);
    double x2 = -2 * c / (b + d);

    printf("%.8e %.8e\n", x1, x2);

    return 0;
}
```

$$x_1 = \frac{-b-\sqrt{b^2-4ac}}{2a},$$

$$x_2 = \frac{-2c}{b+\sqrt{b^2-4ac}}$$

解と係数の関係的な式から計算してもok

解答例②

- 方程式の解を求める問題に対しては，ニュートンラフソン法などで精度のよい解を得られる（提出されたものを見てから気づきました）．

問題D

D: 一致探索

N個の整数A_i $(1 \leq i \leq N)$とM個の整数B_j $(1 \leq j \leq M)$が与えられます． $A_1, A_2, \cdots, A_N$のうちB_jに一致するものがいくつあるかをj行目に出力してください．

制約

- $1 \leq N, M \leq 5 \times 10^5$
- $0 \leq A_i, B_j \leq 10^9$
- 入力はすべて整数

入力

入力は以下の形式で標準入力から与えられます．

> N M
> A_1 A_2 $\cdots$ A_N
> B_1 B_2 $\cdots$ B_M

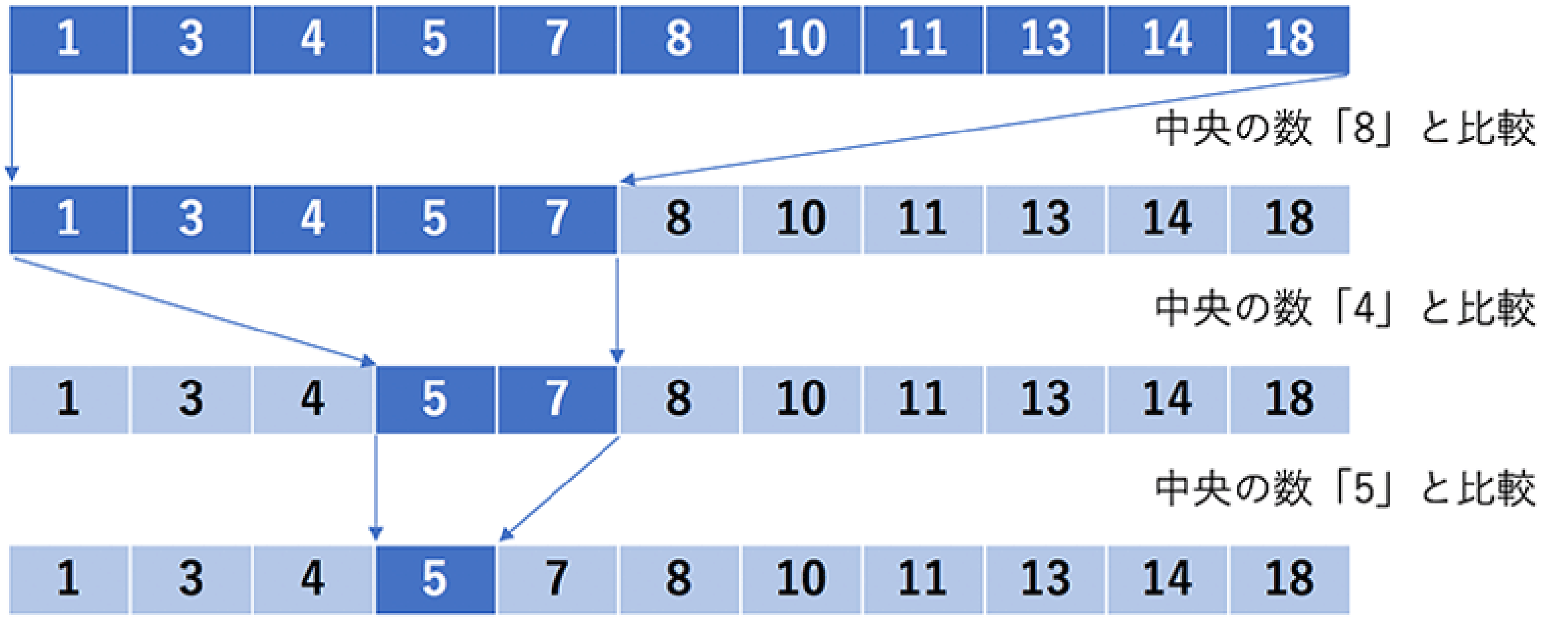
出力

j行目にB_jの個数を出力してください．

二分探索とは

「5」を探したい場合

https://techplay.jp/column/381



見るべき範囲が半分，半分，半分，…
となるので，もしも2^kの長さがあった
としてもたった k 回の探索で終了する．

→データ数 n として，O(log(n))である．

全部の要素を見るとO(n)，二分探索だとO(log(n))．

→n=1000000とすると，log(n)=6*log(10)となる．
約100000倍速く計算が終了する．

C++でソート/二分探索をやるには

- ランダムアクセスイテレータ型のSTLコンテナを使う.
  - こだわりが無ければvectorが使い勝手が良い.

- std::sortを使うとコンテナをソートできる.
  - std::sortは謎のソートアルゴリズムで実装されていて，最悪計算量はクイックソートよりも小さいらしい．最悪計算量は$O(N\log N)$.
  - 余談：vector<pair<int, int>>などでも，大小比較さえ定義すればソートしてくれる.

```
// 第二要素が小さい順にソートする
sort(vec.begin(), vec.end(), [](const pair<int, int> a, const pair<int, int> b) {
    if (a.second != b.second)
        return a.second < b.second;
    else
        return a.first < b.first;
});
```

- std::lower_bound，std::upper_boundを使うとそれぞれ”keyの値**以上**が現れる最初の位置”，”keyの値**より大きい値**が現れる最初の位置”のイテレータが返ってくる.
  - 二分探索で実装されているので最悪計算量は$O(\log N)$.

解答例


```
#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;

int b[1010101];

int main() {
    // aは配列ではなくvectorとして用意する
    vector<int> a;
    int n, m;

    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++) {
        int aTemp;
        scanf("%d", &aTemp);
        a.push_back(aTemp);
    }
    for (int i = 0; i < m; i++) {
        scanf("%d", &b[i]);
    }

    // 計算時間はO(nlog(n))
    sort(a.begin(), a.end());

    // 計算時間はO(mlog(n))
    for (int j = 0; j < m; j++) {
        int key = b[j];

        // 計算時間はO(log(n))
        auto lb = lower_bound(a.begin(), a.end(), key);

        // 計算時間はO(log(n))
        auto ub = upper_bound(a.begin(), a.end(), key);

        printf("%d\n", ub - lb);
    }

    return 0;
}
```

問題E

E: 和の探索

N個の整数A_i $(1 \leq i \leq N)$と整数Dが与えられます． $A_i + A_j + A_k + A_l = D$ $(1 \leq i,\ j,\ k,\ l \leq N)$となる$i,\ j,\ k,\ l$の組がいくつ存在するか判定し，その個数を出力してください． 「$i = m$かつ$j = n$かつ$k = o$かつ$l = p$」でない限りは$i,\ j,\ k,\ l$と$m,\ n,\ o,\ p$は別の組であるとみなします．

制約

- $0 \leq A_i \leq 10^8$
- $0 \leq D \leq 10^9$
- 入力は全て整数
- $1 \leq N \leq 30$であるケースに全問正解すると部分点
- $1 \leq N \leq 150$であるケースに全問正解すると部分点
- $1 \leq N \leq 1000$であるケースに全問正解すると満点

入力

入力は以下の形式で標準入力から与えられます．

> N D
> A_1 A_2 $\cdots$ A_N

出力

$i,\ j,\ k,\ l$の組の個数を出力してください．

解答例①　$O(N^4)$の解法


```cpp
#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;

int main() {
    int n, d;
    vector<int> a;
    scanf("%d%d", &n, &d);
    for (int i = 0; i < n; i++) {
        int aTemp;
        scanf("%d", &aTemp);
        a.push_back(aTemp);
    }

    // 4重ループをまわすので計算時間はO(n^4)
    int ret = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++)
                for (int l = 0; l < n; l++)
                    if (a[i] + a[j] + a[k] + a[l] == d)
                        ret++;

    printf("%d\n", ret);

    return 0;
}
```

- forループを4回まわして全探索する.

N	N^4
30	約 10^6
150	約 10^9
1000	10^12

解答例②　$O(N^3\log(N))$の解法


```cpp
#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;

int main() {
    int n, d;
    vector<int> a;
    scanf("%d%d", &n, &d);
    for (int i = 0; i < n; i++) {
        int aTemp;
        scanf("%d", &aTemp);
        a.push_back(aTemp);
    }

    // 計算時間はO(n*log(n))
    sort(a.begin(), a.end());

    // 4つめのループはa[l]=d-(a[i]+a[j]+a[k])を満たすlが存在するかを探索することに置き換えられる
    // O(log(n))の計算を3重ループで回すので計算時間はO(n^3*log(n))
    long long ret = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                int key = d - (a[i] + a[j] + a[k]);

                auto lb = lower_bound(a.begin(), a.end(), key);
                auto ub = upper_bound(a.begin(), a.end(), key);

                ret += ub - lb;
            }
        }
    }

    printf("%lld\n", ret);

    return 0;
}
```

- 全てのi, j, k, lに対して$a_i + a_j + a_k + a_l = d$となるものはいくつあるか．

$$\Updownarrow$$

- 全てのi, j, kに対して $a_l = d - a_i - a_j - a_k$になる$a_l$はいくつあるか．

- a_lについては全探索する必要はなく，二分探索すれば高速化する

N	N^4	$N^3\log(N)$
30	約 10^6	約 10^5
150	約 10^9	約 10^7
1000	10^12	約 10^10

```cpp
int main() {
    int n, d;
    vector<int> a;
    scanf("%d%d", &n, &d);
    for (int i = 0; i < n; i++) {
        int aTemp;
        scanf("%d", &aTemp);
        a.push_back(aTemp);
    }

    // まずはa[i]+a[j]の計算結果を網羅したvectorであるa2を作る
    // 計算時間はO(n^2)
    int n2 = n * n;
    vector<int> a2;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            a2.push_back(a[i] + a[j]);
        }
    }

    // 計算時間はO(n^2*log(n))
    sort(a2.begin(), a2.end());

    // a2[i]+a2[j]を計算すればa[i]+a[j]+a[k]+a[l]を網羅できる
    // a2[i]+a2[j]の計算は300点の場合と同様に高速化できる
    // O(log(n))の計算をn^2回のループで回すので計算時間はO(n^2*log(n))
    long long ret = 0;
    for (int i = 0; i < n2; i++) {
        int key = d - a2[i];

        auto lb = lower_bound(a2.begin(), a2.end(), key);
        auto ub = upper_bound(a2.begin(), a2.end(), key);

        ret += ub - lb;
    }

    printf("%lld\n", ret);

    return 0;
}
```

解答例③　$\mathrm{O}(N^2\log(N))$の解法

- 全てのi, jに対して$a_i + a_j$の取り得る数値は$a2_1, a2_2, a2_3, \cdots, a2_{N^2}$だとする（$a2$による メモ化）．
- 全てのpに対して$a2_q = d - a2_p$となる$a2_q$がいくつあるのか数えればよい．

- 配列$a2$のソートで$\mathrm{O}(N^2\log(N))$かかる．
- 配列$a2$の二分探索で$\mathrm{O}(\log(N))$かかり，それをN^2回行う．
- 全体で$\mathrm{O}(N^2\log(N))$のアルゴリズムになる．

N	N^4	$N^3\log(N)$	$N^2\log(N)$
30	約 10^6	約 10^5	約 10^3
150	約 10^9	約 10^7	約 10^5
1000	10^12	約 10^10	約 10^7

問題F

F: クラス

次の操作と終了条件を考えます．

操作

1. 次のいずれかの操作を行う
   - 対象の数字が奇数なら，3を掛けて1を加える
   - 対象の数字が偶数なら，2で割る
2. 現在の数字を標準出力する

終了条件

- 対象の数字が1なら終了する
- 対象の数字がすでに標準出力されたものであれば終了する

N個の整数X_i $(1 \leq i \leq N)$が与えられます． まずX_1に対して，X_1を出力し，**終了条件**を満たすまで**操作**を行ってください． 次に$X_2, \cdots, X_N$に対してもそれぞれ順番に同様の処理を行ってください． **ただし，後述のプログラムの指定した部分だけを編集し，回答してください．** それ以外の部分を編集することは(ジャッジは通りますが)認められません．

制約

- $1 \leq N \leq 10^4$
- $0 \leq X_i \leq 10^4$
- 入力は全て整数
- 上記制約のもとで操作対象の数字は50000000を超えないことが保証される

基となるプログラム

```
#include <cstdio>
using namespace std;

const int MAX_X = 50005000;

class Collatz {
  private:
    /* ↓ここを編集する． */

    /* ↑ここを編集する． */

  public:
    Collatz(int x);
    int step();
    bool shouldProceed();
    void leaveFootprint();
    int getX();
};

/* ↓ここを編集する． */

/* ↑ここを編集する． */

int main() {
    int n;
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        int x;
        scanf("%d", &x);

        Collatz collatz(x);
        printf("%d\n", collatz.getX());
        while (collatz.shouldProceed()) {
            collatz.leaveFootprint();
            printf("%d\n", collatz.step());
        }
    }

    return 0;
}
```

ヘッダ部分

- Cで書かれているプロジェクトでは，この部分が`.h`というファイルに書かれていることが多い．C++では`.hpp`というファイルに書かれていることが多い．
- 「今までどのような数字を出力してきたか」をstepped_onというスタティック変数に保存しておく．
  - Javaではクラス変数と呼ばれている．
  - あまり使わない機能ではある．この機能が無いプログラミング言語も多い．

```cpp
class Collatz {
  private:
    int x;
    static bool stepped_on[MAX_X];

  public:
    Collatz(int x);
    int step();
    bool shouldProceed();
    void leaveFootprint();
    int getX();
};
```

メンバ関数の実装

- static変数には初期化の記述が必要．static変数に限らずグローバルな場所で変数を初期化すると，bool型ならfalseが初期値として入り，int型なら0が初期値として入る．

```cpp
Collatz::Collatz(int x) {
    this->x = x;
    stepped_on[1] = true;
}

int Collatz::step() {
    if ((x & 1) == 0) {
        x /= 2;
    } else {
        x *= 3;
        x += 1;
    }
    if (x >= MAX_X) {
        printf("Error: x=%d exceeded a limit\n", x);
        return 1;
    }
    return x;
}
```

```cpp
bool Collatz::shouldProceed() {
    return !stepped_on[x];
}

void Collatz::leaveFootprint() {
    stepped_on[x] = true;
}

int Collatz::getX() {
    return x;
}

bool Collatz::stepped_on[MAX_X];
```

問題G

G: テンプレート

決められた文字列を出力してください． **ただし，後述のプログラムの指定した部分だけを編集し，回答してください．** それ以外の部分を編集することは(ジャッジは通りますが)認められません．

入力

入力はありません．

出力

以下の文字列を出力してください．

```
*-----
i:
777
*-----
x1, x2, x3, x4, x5:
1
2
3
4
5
*-----
str, p:
"hello"
(0, 1)
*-----
vec, vec_str, vec_pair:
{0, 1, 2, 3}
{"hello", "world", "!"}
{(0, 1), (2, 3), (4, 5)}
*-----
pair_vecvec:
({0, 1, 2, 3}, {4, 5, 6, 7})
*-----
vec_pair_vecvec:
{({0}, {1, 2}), ({3}, {}), ({4}, {5})}
*-----
vec_vec_vec:
{{{0}, {1, 2}}, {{3}, {4}, {5, 6}}, {{}}}
```

基となるプログラム

```
#include <cstdio>
#include <string>
#include <vector>
using namespace std;

/* ↓ここを編集する． */

/* ↑ここを編集する． */

int main() {
    int i = 777;
    int x1 = 1, x2 = 2, x3 = 3, x4 = 4, x5 = 5;
    string str = "hello";
    pair<int, int> p = {0, 1};
    vector<int> vec = {0, 1, 2, 3};
    vector<string> vec_str = {"hello", "world", "!"};
    vector<pair<int, int>> vec_pair = {{0, 1}, {2, 3}, {4, 5}};
    pair<vector<int>, vector<int>> pair_vecvec = {{0, 1, 2, 3}, {4, 5, 6, 7}};
    vector<pair<vector<int>, vector<int>>> vec_pair_vecvec = {
        {{0}, {1, 2}}, {{3}, {}}, {{4}, {5}}};
    vector<vector<vector<int>>> vec_vec_vec = {{{0}, {1, 2}}, {{3}, {4}, {5, 6}}, {{}}};

    disp(i);
    disp(x1, x2, x3, x4, x5);
    disp(str, p);
    disp(vec, vec_str, vec_pair);
    disp(pair_vecvec);
    disp(vec_pair_vecvec);
    disp(vec_vec_vec);

    return 0;
};
```

C++のマクロとは?


```
#define PI 3.14159265
```

←定数を定義するのに使う人もいる

```
#define rep(i, x) for (int i = 0; i < x; i++)
#define rep1(i, x) for (int i = 1; i <= x; i++)
#define srep(i, s, x) for (int i = s; i < x; i++)
#define rrep(i, x) for (int i = x - 1; i >= 0; i--)
#define rrep1(i, x) for (int i = x; i > 0; i--)
#define rsrep(i, s, x) for (int i = x - 1; i >= s; i--)
```

←for文簡略化のマクロ

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        printf("%d\n", a[i][j]);
    }
}
```

←これが

こう記述できる→

```
rep(i, n) rep(j, m) {
    printf("%d\n", a[i][j]);
}
```

※可読性を著しく下げる，変なクセが身につく，などの理由より，多用は禁物．特に他人が見るようなコードに使うべきではない．

方針

- あらゆる変数を標準出力する最強の関数dispを作りたい．

- 変数名の一覧を出力する関数name_dispを作る．
- 変数の値を出力する関数val_dispを作る．
  - ↓のように変数がいくつあってもそれぞれ出力できるような再帰的な処理を作る．

```
disp(x1, x2, x3, x4, x5);
```

  - vectorの中にvectorがある，というような入れ子の構造になっていても，再帰的に最深部まで出力できるようにする．
  - 変数を文字列化して，最後にその文字列を出力する，という方針を採る．

dispというマクロの周辺

③templateを使うと，任意の変数型に対して同じ操作をできる．そしてこのように記述すれば，いくつの変数が入って来ても最初の変数がFに格納され，それ以外はLに格納される．Lに対してもう一度val_dispをすれば，再帰的に同じ処理をできる．

②の実装

```cpp
void name_disp(string s) {
    printf("%s:\n", s.c_str());
}
void val_disp() {
}
template <typename First, typename... Lest> void val_disp(First F, Lest... L) {
    printf("%s\n", to_string(F).c_str());
    val_disp(L...);
}

#define disp(...)                                   \
    printf("*-----\n");                             \
    name_disp(#__VA_ARGS__);                        \
    val_disp(__VA_ARGS__)
```

あとは任意の変数を文字列化する関数`to_string`を定義すればok

①まずは`*-----¥n`を出力する．

②変数名の一覧を`#__VA_ARGS__`で取得し，そのまま出力する．

任意の変数を文字列化する関数`to_string`

- intなどに対してはもともとto_stringという関数が用意されている

stringに対しては”の記号をつけるだけ

pair用のtemplate．他の関数に呼ばれる可能性のあるtemplateは実装を後回しにするとしても宣言だけしておく必要がある．

```cpp
string to_string(const string &s) {
    return '"' + s + '"';
}
template <typename A, typename B> string to_string(pair<A, B> p);
template <typename A> string to_string(A v) {
    bool first = true;
    string ret = "{";
    for (const auto &x : v) {
        string s = to_string(x);
        if (!first) {
            ret += ", ";
        }
        first = false;
        ret += s;
    }
    ret += "}";
    return ret;
}
template <typename A, typename B> string to_string(pair<A, B> p) {
    return "(" + to_string(p.first) + ", " + to_string(p.second) + ")";
}
```

[重要]vectorなどのコンテナでは，その中身を全て文字列化する．文字列化はto_stringを再帰的に呼ぶことによって，中身がvectorであるvectorなどにも対応できるようになる．

pairには丸括弧をつける

もっとプログラミングやりたい人のために

- E問題みたいなのが好きな人は[AtCoder]で競プロをやりましょう

- 参考文献：通称「蟻本」
- 基本的なアルゴリズムについて学べる.