プログラミング第一同演習

慶應義塾大学 理工学部 情報工学科
講義担当: 河野 健二
演習担当: 杉浦 裕太

本日の内容

- 四則演算の補足
  - `i = i + 1` の意味
  - ちょっと便利な演算子
  - キャスト
- `printf` の補足
  - 桁を指定した出力
- 制御構造
  - `for` 文 (Python の `for` とは違う)
- 配列
  - 配列の基本 (Python のリストとは全く違う)

妙な計算式

sslab system software

- 次のプログラムを実行してみよう

```
#include <stdio.h>

main()
{
        int i;
        i = 3;                          // 変数 i の値を 3 に初期化
        printf("i = %d¥n", i);          // 当然，"i = 3" と表示される

        i = i + 1;                      // この式の意味は？
        printf("i = %d¥n", i);          // 何が表示されるでしょう？

        return 0;
}
```

i = i + 1 の意味

- 思い出そう
  - = は数学の等号ではない
  - 左辺の変数に右辺の式を計算した結果を代入するという意味
- i = i + 1 は次のように実行される
  - まずは右辺の式を計算する
    - i の値は 3 だから，i + 1 は 4 になる
  - 右辺の式を計算した結果を左辺の変数 (この場合は i) に代入
    - i に 4 を代入する

```
i = 3;                      // 変数 i の値を 3 に初期化
printf("i = %d¥n", i);      // 当然，"i = 3" と表示される

i = i + 1;                  // この式の意味は？  → i の値をひとつ増やす
printf("i = %d¥n", i);      // 何が表示されるでしょう？ → 4 が表示される
```

Quiz

- 計算結果を考えてみよう
  - ```
    int i, j;
    i = 3;
    j = 7;
    i = i + 1;
    i = i + 2;
    i = i - 3;
    j = 7;
    j = 7 * j;
    j = 3 + j;
    j = j / 3;
    i = i + j;
    j = i - j;
    ```
    ```

ちょっと便利な代入: 複合代入演算子

- `i = i + j` という代入式はよく出てくる
  - よく出てくる式は簡単に書けるように工夫されている
- `i = i + j` は `i += j` と書いてもよい

演算子	一般形	意味
+=	v += e	v = v + e
−=	v −= e	v = v − e
*=	v *= e	v = v * e
/=	v /= e	v = v / e
%=	v %= e	v = v % e

Quiz

- 計算結果を考えてみよう
  - ```
    int i, j;
    i = 4;
    j = 5;
    i += 2;
    j -= 3;
    i += j;
    j *= j;
    i -= i + j;
    j /= i - j;
    ```
    ```

ちょっと便利な代入: ++ と --

- 1 だけ値を増やす，1 だけ減らすことは多い
  - `i = i + 1` とか `j += 1` とか
  - 1 だけ値を増やすことを インクリメント (increment) という
  - 1 だけ値を減らすことを デクリメント (decrement) という

- インクリメント，デクリメントは簡単に書ける
  - インクリメント: `i++` または `++i`
    - 意味: 整数型の変数 `i` の値を 1 だけ増やす
  - デクリメント: `i--` または `--i`
    - 意味: 整数型の変数 `i` の値を 1 だけ減らす
  - 注意: これらは浮動小数点数 (`float` など) には使用できない

微妙なこと: i++ と ++i

- i++ と ++i は微妙に意味が違う
  - i++ の式の値は ++ を実行する前の値
  - ++i の式の値は ++ を実行したあとの値

```
int i, j, k;
i = 3;
j = ++i;
k = i++;
```

- ++i により i の値は 4 になる
- ++i の式の値は 4
- j の値は 4 になる

- i++ により i の値は 5 になる
- i++ の式の値は 4 (++ を実行する前の値)
- k の値は 4 になる

注意

- 紛らわしい記述は避けること
- `i+++j` はどのように解釈されるのか？
  - `(i++)+j` のようにも，`i+(++j)` のようにも見える
  - 答え：`(i++) + j` と解釈される
- `j = i++ + i++;` はどうなるのだろうか？
  - どうなるのかわからない（処理系依存という）
- C 言語の落とし穴にはまらないように！

Quiz

sslab system software

- ```
  int i, j, k;
  i = 2;
  i++;
  j = i;
  ++j;
  --i;
  j--;

  /* ややこしい例 */
  i = j++;
  k = ++i;
  j = --k;
  k = i--;
  ```
  ```

キャスト

- データの型 (type) を変換すること
  - 例: int 型と float 型を相互に変換する
    - (型名)変数 と書くと変数の値を指定された型に変換する
    - 常に可能なわけではない

```
#include <stdio.h>

int main()
{
    int i, j;
    float f;

    scanf("%d %d", &i, &j);
    f =  (float) i / j;
    printf("f = %f¥n", f);

    return 0;
}
```

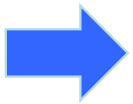
f = i / j; とすると・・・
i, j ともに整数．したがって i / j の結果は切り捨て

切り捨てて欲しくないときは？
float 型にキャストしてから計算する
(float) i により整数 i の値を float 型にする
キャストにより i は float 型とみなされるため，
小数点以下も求めてくれる （切り捨てない）

printf の補足：桁数指定 (1/2)

- printf で表示する数値の桁数を指定できる
  - 何もやらないと左揃えになる

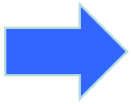
```
printf("[%d]¥n", 1);
printf("[%d]¥n", 12);
printf("[%d]¥n", 123);
printf("[%d]¥n", 1234);
```



```
[1]
[1 2]
[1 2 3]
[1 2 3 4]
```

  - 桁数を指定して右揃えにできる. ここでは 4 桁に設定

```
printf("[%4d]¥n", 1);
printf("[%4d]¥n", 12);
printf("[%4d]¥n", 123);
printf("[%4d]¥n", 1234);
```

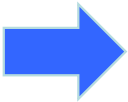


```
[   1]
[  1 2]
[  1 2 3]
[1 2 3 4]
```

printf の補足：桁数指定 (2/2)

- 浮動小数点数の桁数も指定できる
  - 小数点を含めた桁数と小数点以下の桁数を指定する
  - 例：全体で 8 桁，小数点以下 3 桁

```
printf("[%8.3f]¥n", 1.23456);
printf("[%8.3f]¥n", 12.3456);
printf("[%8.3f]¥n", 123.456);
printf("[%8.3f]¥n", 1234.56);
```



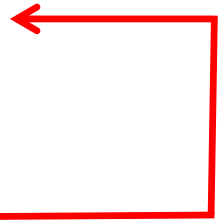
```
[   1.234]
[  12.345]
[ 123.456]
[1234.560]
```

繰り返し

- ある条件が満たされるまで処理を繰り返す
  - 繰り返しのことをループ (loop) という
- 例題:
  - 100 より大きい最小の平方数を求めたい
  - 次のように考える
    - 変数 `i` を `1` から順に大きくしていき,
      `i * i` が `100` を超えたところで, `i * i` の値を表示する
  - 言い換えると,
    - `i * i` が `100` 以下の間, `i` の値をひとつずつ増やす
    - `i * i` が `100` を超えたら, `printf("%d¥n", i * i)` をする

for 文: 繰り返しの形

- 次のように使う
  - ```
    for (i = 0; i < 10; i++) {
        printf("i = %d¥n", i);
    }
    ```
  - まず, i = 0 を実行する
  - 次に i < 10 が成り立つかどうかチェックする
    - もし, i < 10 が成り立っていたら,
      - printf() を実行する
      - i++ を実行する
    - もし, i < 10 が成り立っていなかったら for 文の次に進む



  - 実行結果:
    - 0 から 9 が順に表示される
    ```

for 文: 一般の形

1. 式$_1$ を実行する
2. 式$_2$ が成り立つかどうか調べる
3. 式$_2$ が成り立つなら,
   - 文$_1$～文$_n$ を実行する
   - 式$_3$ を実行する
   - 2. に戻る
4. 式$_2$ が成り立たないなら,
   - for 文の後の文に実行を移す

```
for (式1; 式2; 式3) {
        文1
        文2
        . . .
        文n
}
... // for 文の後の文
```

例題：100 より大きい最小の平方数を求める

```
#include <stdio.h>

int main()
{
    int i;

    for (i = 1; i * i <= 100; i++) {
        /* ここは何もしない */
    }

    printf("answer = %d¥n", i * i);

    return 0;
}
```

i * i <= 100 が成り立つ間，
i++ を繰り返している

- 次のように考える
  - 変数 i を 1 から順に大きくしていく
  - i * i が 100 を超えたところで i * i の値を表示する
- 言い換えると，
  - i * i が 100 以下の間，i の値をひとつずつ増やす
  - i * i が 100 を超えたら printf("%d¥n", i * i) をする

例題: 複利計算

- 銀行の定期預金に預けたときの元利合計を計算しその年数とともに出力せよ.

  預金期間を指定し，それまでの1年ごとの元利合計を示すこと．なお，元金(円)，年利率(%)，預金期間 (年) はキーボードから入力するものとする

```
1000000 5.7 5
year 1, total 1057000.000000 yen
year 2, total 1117249.000000 yen
year 3, total 1180932.193000 yen
year 4, total 1248245.328001 yen
year 5, total 1319395.311697 yen
```

例題：元利合計の計算と表示

```
#include <stdio.h>

int main()
{
        int principal, term, i;
        float rate, total;

        scanf("%d %f %d", &principal, &rate, &term);

        total = principal;

        for (i = 1; i <= term; i++) {
                total *= 1.0 + rate / 100.0;
                printf("year %d, total %f yen¥n", i, total);
        }
        return 0;
}
```

元金： principal
期間（年）： term
年利（%）： rate

scanf で 3 つの変数に読込み

1 年後の預金額は，現在の預金額に（1 + rate / 100） を掛けたもの

for 文で term 回ループする

for 文：無限ループ

- for (式$_1$; 式$_2$; 式$_3$) において 式$_1$, 式$_2$, 式$_3$ は省略可
- 式$_2$ を省略すると無限ループとなる
  - ループを抜ける条件がない
    - for (式$_1$; ; 式$_3$) {
      ...
      }
  - わざと無限ループにしたいときに，よく使うやり方
    - for (;;) {
      ...
      }
- バグで無限ループを作ってしまうこともよくある
  - プログラムを強制終了するには Ctrl-c (controlキーを押しながら 'c') を入力

配列の基本

- 配列の初歩を学ぶ
  - C 言語における配列の意味は，ポインタを学んでから学ぶ
- 配列の宣言，初期化，使い方
  - 1 次元配列
  - 2 次元配列
- Python のリストとは全く違う
  - 配列 (array) とリスト (list) は異なる概念
  - [ ] を使って添字でアクセスするところが似ているだけ

簡単な例題

- 3 人の学生の試験の得点を入力し，平均点を表示した後，3 人の得点も表示するプログラムを作成せよ

```
#include <stdio.h>

int main()
{
        int t1, t2, t3;
        scanf("%d %d %d", &t1, &t2, &t3);

        printf("average = %f¥n", (t1 + t2 + t3) / 3.0);
        printf("scores: %d, %d, %d¥n", t1, t2, t3);

        return 0;
}
```

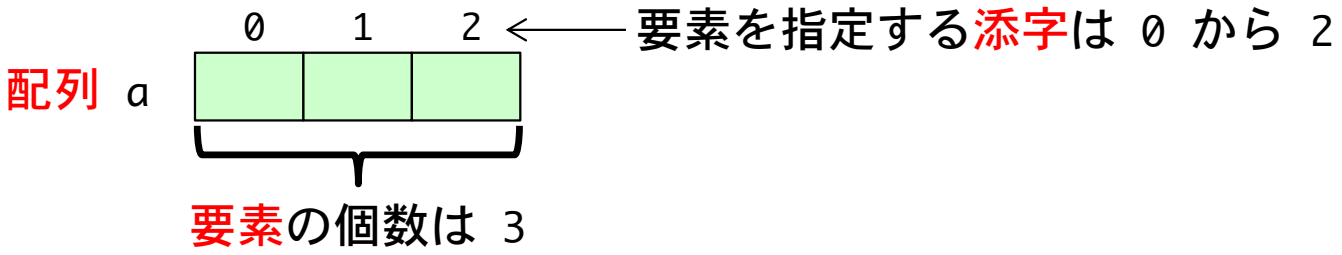
3 人の得点を記録するため，
3 つの int 型の変数を定義

学生が 100 人いたら 100 個の変数を用意する？

とてもやってられない…

配列 (1)

- 同じ型の変数をまとめて定義，利用する仕組み
- 例：int 型の変数を 3 つまとめて宣言する
  - `int a[3];`
    “int 型の配列 a，要素の数は 3 つ” を宣言



  - 数学の数列 $\{a_i\}$ $(0 \leqq i \leqq 2)$ にちょっと似ている

配列 (2)

- int a[3];
  - a[0], a[1], a[2] という 3 つの変数を宣言したような感じ
    - a[0] = 1;
      a[1] = 3;
      a[2] = 2;

      ⎫ 配列のそれぞれの要素は int 型の変数となる

    - ```
      for (i = 0; i < 3; i++) {
          printf("a[%d] = %d¥n", i, a[i]);
      }
      ```
      ← for 文ですべての要素を表示
- Python のリストのようなもの？
  - まったく違う！
      ```

例題を配列を使って書き直し

- 変数 t1，t2，t3 の代わりに配列 t[3] を使う

```
#include <stdio.h>
int main()
{
        int t[3];
        int i, total = 0;
        for (i = 0; i < 3; i++) {
                scanf("%d", &t[i]);
                total += t[i];
        }
        printf("average = %f¥n", total / 3.0);
        for (i = 0; i < 3; i++) {
                printf("%d  ", t[i]);
        }
        return 0;
}
```

3 人の得点を記録するため，
要素数 3 の int 型の配列を定義

学生が 100 人いたら，
配列の要素数を 100 にする

あとは青地の 3 をすべて
100 にすればよい

マクロ定義: #define

- #define NUM_STUDENTS 3
  - 「プログラム中に NUM_STUDENTS という文字列があったら，3 という文字列に置き換える」というマクロ定義

- #define を使う理由
  - プログラム中に定数 (e.g., 3) がいきなり現れると，何を意味しているのか分からない
  - このような定数は変更されるかもしれないが，そのときすべてを書き換えるのは面倒 (ミスも起こりやすい)
  - #define を使えば，その定義のみを書き換えればよい

例題を配列を使って“さらに”書き直し

- #define を使ってプログラムの変更を楽にする
  - 数字の 3 というマジックナンバー(意味のわかりくい数字のこと)をなくす

```
#include <stdio.h>
#define NUM_STUDENTS  3
int main()
{
        int t[NUM_STUDENTS];
        int i, total = 0;
        for (i = 0; i < NUM_STUDENTS; i++) {
                scanf("%d", &t[i]);
                total += t[i];
        }
        printf("average = %f¥n", (float)total / NUM_STUDENTS);
        for (i = 0; i < NUM_STUDENTS; i++) {
                printf("%d ", t[i]);
        }
        return 0;
}
```

学生が 100 人いたら,
青字の 3 を 100 にするだけ！

知っていると便利なこと: 配列の初期化

- 配列を宣言するとき，その要素を初期化できる
  - 例1: int 型の配列を宣言し，その要素を順に 3，4，8 で初期化

```
int a[ ] = {3, 4, 8};
```

    - a[0] = 3, a[1] = 4, a[2] = 8 となる
    - 配列の要素数は 3 になる
    - 要素の個数を宣言しなくてよいことに注意

  - 例2: 要素の数を指定することもできる

```
int a[5] = {7, 6, 2};
```

    - 配列の要素数は指定したとおり 5 になる
    - 初期化されていない要素の初期値は 0 になる
      - a[0] = 7, a[1] = 6, a[2] = 2, a[3] = 0, a[4] = 0 となる

Quiz: 間違い探し

- ```
  int a[3];
  a[1] = 1;
  a[2] = 2;
  a[3] = 3;
  ```

- ```
  int i, b[100];
  for (i = 0; i <= 100; i++){
      b[i] = i;
  }
  ```

- ```
  int c[] = {3, 4, 5};
  printf("%d¥n", c[3]);
  ```

- ```
  int d[2] = {6, 7, 8};
  ```

- 答え:
  - 配列 a の要素数は 3, 有効な添字は 0 ～ 2. よって, a[3] = 3; が誤り
  - 配列 b の要素数は 100, 有効な添字は 0 ～ 99.
    for 文の条件が i <= 100 なので, b[100] = 100 という代入が行われる.
    よって, i < 100 が正しい
  - 配列 c の要素数は 3, 有効な添字は 0 ～ 2. よって, c[3] が誤り
  - 配列 d の要素数は 2. なのに 3 つの要素を初期化している. 要素数を 3 に増やすか, 初期化を減らす

例題: 金種計算

- 商品の値段と支払額を読み込んで，釣銭とそれに必要な硬貨の最小枚数を求めなさい．
  硬貨は 500円，100円，50円，10円，5円，1円の 6種類とする

- 例:
  - 645 円の商品を買うのに 1,000 円札を出した．
  - おつりは 1,000 − 645 = 355 円．
  - 355 円のおつりを払うのに必要な硬貨の最小枚数
    - 100 円玉 3 枚，50 円玉 1 枚，5 円玉 1 枚の合計 5 枚
    - なるべく金額の大きい硬貨から順に，使えるだけ使っておつりを払うようにしていけばよい．

例題: 実行例

```
1284 2000
  500 yen coins: 1
  100 yen coins: 2
   10 yen coins: 1
    5 yen coins: 1
    1 yen coins: 1

1000 1000
exact payment

5050 5030
payment: 20 yen shortage
```

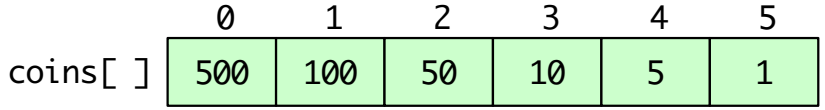
例題: 硬貨の枚数計算の考え方

sslab system software

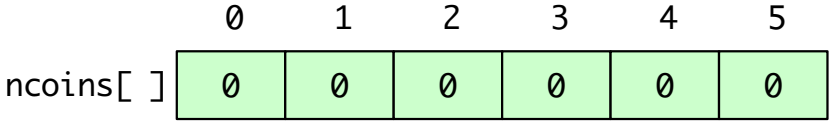
- 釣銭 x 円に必要な a 円硬貨の枚数 $n = x / a$
  - 整数同士の割り算なので，切り捨てになる
- 残りの釣銭 = $x - n * a = x \% a$
  - x を a で割った余りが残りの釣り銭
- 以上の操作を500円硬貨からはじめて釣銭が0になるまで繰り返す

例題: 硬貨の枚数計算

- 各硬貨の金額を覚えておく配列を用意する
  - `int coins[ ]`:
    500円玉が 0 番, 100 円玉が 1 番, ･･･ 1 円玉が 5 番という感じ



- それぞれの硬貨を使う枚数を記録する配列を用意する
  - `int ncoins[ ]`:
    0 番目の要素に 500 円玉の枚数を記録,
    1 番目の要素に 100 円玉の枚数を記録, ･･･ という感じ
    - 500円玉が 0 番, 100 円玉が 1 番, ･･･ 1 円玉が 5 番という関係は `coins[ ]` と同じ
    - 最初に `0` 初期化しておく



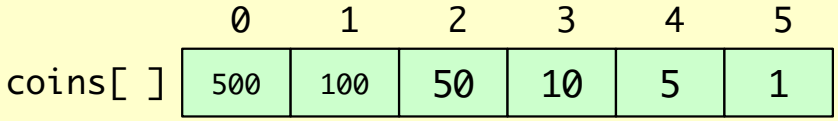
例題: 配列の初期化

```
#include <stdio.h>

int coins[ ] = {500, 100, 50, 10, 5, 1};

#define ASIZE 6

   // continue to next page
```



- coins[ ]への初期値の設定
- coins[0]～coins[5]の領域が確保され 500～1 で初期化される

ASIZE は配列 coins[] の要素数として使う
添字は 0 ～ 5, 要素数は 6 となることに注意

例題: main()の変数宣言

```
#include <stdio.h>

int coins[ ] = {500, 100, 50, 10, 5, 1};
#define ASIZE 6

int main()
{
      int ncoins[ASIZE];
      int price, payment, change, i;

      for (i = 0; i < ASIZE; i++) {
            ncoins[i] = 0;
      }
      // continue to next page
```

- ncoins[ ] は釣銭の枚数を保持する配列
  - ncoins[0]は500円硬貨の枚数
  - ncoins[1]は100円硬貨の枚数，など
- price は値段，payment は支払額，change は釣銭
- i はループを制御する変数

- ncoins[ ] を 0 で初期化しておく

例題: 釣り銭の金額で場合分け

```
        // continued from previous page

        scanf("%d %d", &price, &payment);    // 商品の金額と支払額を入力

        change = payment - price;            // change は釣り銭の金額
        if (change > 0) {  // 釣り銭の値で場合分け

                // ここが肝心なところ．後述

         } else if (change < 0) {                    // 支払額が足らない
                printf("payment: %d yen shortage¥n", -change);
         else {
                printf("exact payment¥n");          // change == 0 の場合．釣り銭はなし
         }
        return 0;
}
```

例題：肝心なところ

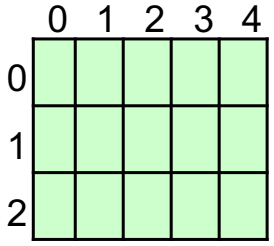
- 硬貨の額の大きい方から順に，できるだけ多く釣り銭に使う
  - i == 0 からスタート
    - coins[0]，すなわち 500 円玉を使って払えるだけ釣り銭を払う
    - ncoins[0] に 500 円玉で使って支払える枚数を代入する
    - change を残りの釣り銭に更新する
  - これを釣り銭 (change) が 0 になるまで繰り返す

```
// 前のスライドの肝心なところ
for (i = 0; change > 0; i++) {
    ncoins[i] = change / coins[i];
    change %= coins[i];
}
```

```
// 結果の表示
for (i = 0; i < ASIZE; i++) {
  if (ncoins[i] > 0) {
     printf("%4d yen coins: %d¥n", coins[i], ncoins[i]);
  }
}
```

2 次元配列

- ここまで説明してきた配列は 1 次元配列
  - a[i] というように，要素を指定するための添字はひとつ
- 2 次元配列も利用できる
  - a[i][j] というように，2 つの添字で要素を指定する
  - 宣言の仕方：
    - int a[3][5];
      右図のように 3 行 5 列の行列のような感じ



2 次元配列の初期化

- 次のように初期化する
  - `int a[ 2 ][ 3 ] = {{72, 67, 84}, {67, 92, 71}};`
  - 次のように初期化される

	0	1	2
0	72	67	84
1	67	92	71

- 要素数の指定は，最初だけ省略できる
  - `int a[][ 3 ] = {{72, 67, 84}, {67, 92, 71}};`

例題：行列の和

- 3 行 3 列の正方行列 $A = (a_{ij})$ と $B = (b_{ij})$ の和を求めるプログラムを作成せよ. 行列 A の各要素は $a_{ij} = i + j$ とし，行列 B は単位行列とする

- 解法：
  - 行列 A, B を 2 次元配列で表す
    - ループを使って行列 A, B の各要素を初期化する
  - 計算結果を入れる 2 次元配列 c を用意する
    - 配列 c の各要素は行列 A, B の対応する要素の和を入れればよい
  - 結果の入った配列 c の中身を表示する

例題: 行列の和

```
#include <stdio.h>
int main()
{
    int a[3][3], b[3][3], c[3][3];
    int i, j;

    for (i = 0; i < 3; i++) {
      for (j = 0; j < 3; j++) {
        a[i][j] = i + j;
        if (i == j) {
          b[i][j] = 1;
        } else {
          b[i][j] = 0;
        }
      }
    }
   …
```

- 行列 A, B に対応する二次元配列 a, b を宣言.
- 答えを入れるための二次元配列 c も宣言

- 行列 A の各要素 a_{ij} の値は $i+j$

- 行列 B を初期化して単位行列にする
- すなわち i と j の値が等しいときは 1,
  i と j の値が等しくないときは 0 にすればよい

例題: 行列の和

```
        for (i = 0; i < 3; i++) {
            for (j = 0; j < 3; j++) {
                c[i][j] = a[i][j] + b[i][j];
            }
        }

        for (i = 0; i < 3; i++) {
            for (j = 0; j < 3; j++) {
                printf("%4d ", c[i][j]);
            }
            printf("¥n");
        }

        return 0;
}
```

- 行列 A, B の対応する要素同士を足し，配列 c の各要素に代入する
- 計算結果を表示
- 一行表示したところで改行する

まとめ

- 四則演算の補足
  - 複合代入演算子，++, --, キャスト
- printf の補足
  - 表示する桁数の指定
- C 言語の for 文
  - Python とは勝手が違うので注意
- C 言語の配列
  - Python のリストと混同しないこと