プログラミング論I
(13) 二次元配列と コンパイラの仕組み

電子情報工学専攻　日浦 慎作

ログインしておいてください

二次元配列とは

- 数値などが2次元的に並んだもの

  用途

  – (数学の)行列を表す，計算する
  – 画像を表す，画像処理する
  – ボードゲームの盤面の状況を表す

  など
- C言語では3次元の配列も可能
  – 次元数は無制限

例1　オセロの盤面の初期化　　ex1.c（前半）

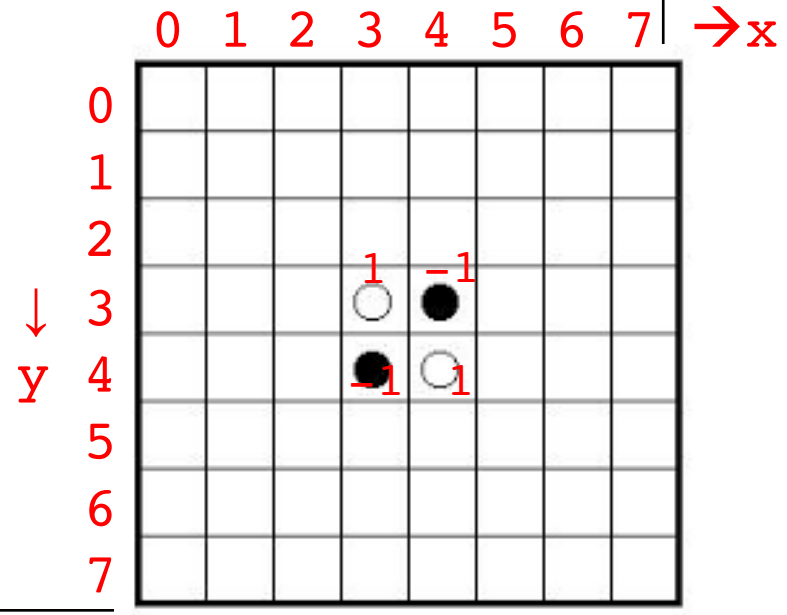
```
#define GRIDSIZE 8

int main(void) {
  int grid[GRIDSIZE][GRIDSIZE];
  int x, y;

  for(y = 0; y < GRIDSIZE; y++) {
    for(x = 0; x < GRIDSIZE; x++) {
      grid[y][x] = 0;
    }
  }
  grid[3][3] =  1;
  grid[3][4] = -1;
  grid[4][3] = -1;
  grid[4][4] =  1;

  . . .
```

2次元配列は
[8][8]のように
[ ]を2回並べる

変数定義

コマのないところは0

白コマは1
黒コマは-1



２次元配列のポイント

- [ ] を2回続ける
  - 変数定義時 `int array[4][3];`
    （4行3列の2次元配列）
  - 変数を使う時の書き方

```
array[y][x] = 1;
printf("data = %d¥n", array[2][2]);
scanf("%d", &array[j][i]);
```

- よくあるミス

  `array[4,3];` のような書き方はできない

例1（続き） オセロの盤面を表示する関数　ex1.c（後半）

```
void showBoard(int g[GRIDSIZE][GRIDSIZE]);

void showBoard(int g[GRIDSIZE][GRIDSIZE]) {
  int x, y;

  for(y = 0; y < GRIDSIZE; y++) {
    for(x = 0; x < GRIDSIZE; x++) {
      if(g[y][x] == 1)
     printf("* ");
      else if(g[y][x] == -1)
     printf("o ");
      else
     printf(". ");
    }
    printf("¥n");
  }
}
```

表示結果

```
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . * o . . .
. . . o * . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

２次元配列を関数に渡す

引数の[ ]の中の数字の省略について

- 良い例

```
void func(int array[8][8]) { ... }
void func(int array[][8]) { ... }
```

- ダメな例

```
void func(int array[][]) { ... }
```

最初の[ ]の中の添字だけ省略可能

2次元配列の初期化

- {} を二重にする
  - 段付け(インデント)をすると見やすい
- 内側の {} は，より後ろ(右)の添字に対応

```
int data[3][2] = {
        { 4, 5 },
        { 2, 3 },
        { 1, 7 }
};
```

	0	1
0	4	5
1	2	3
2	1	7

- 上のような定義の場合「3行2列の行列」と読めば良い.
- カンマの配置，有無に注意

例2　3行2列の行列の初期化と表示　　ex2.c

```
#include <stdio.h>

int main(void) {
  int data[3][2] = {
    { 4, 5 },
    { 2, 3 },
    { 1, 7 }
  };
  int x, y;

  for(y = 0; y < 3; y++) {
    for(x = 0; x < 2; x++) {
      printf("data[%d][%d] = %d¥n", y, x, data[y][x]);
    }
  }
  return 0;
}
```

実行結果

```
data[0][0] = 4
data[0][1] = 5
data[1][0] = 2
data[1][1] = 3
data[2][0] = 1
data[2][1] = 7
```

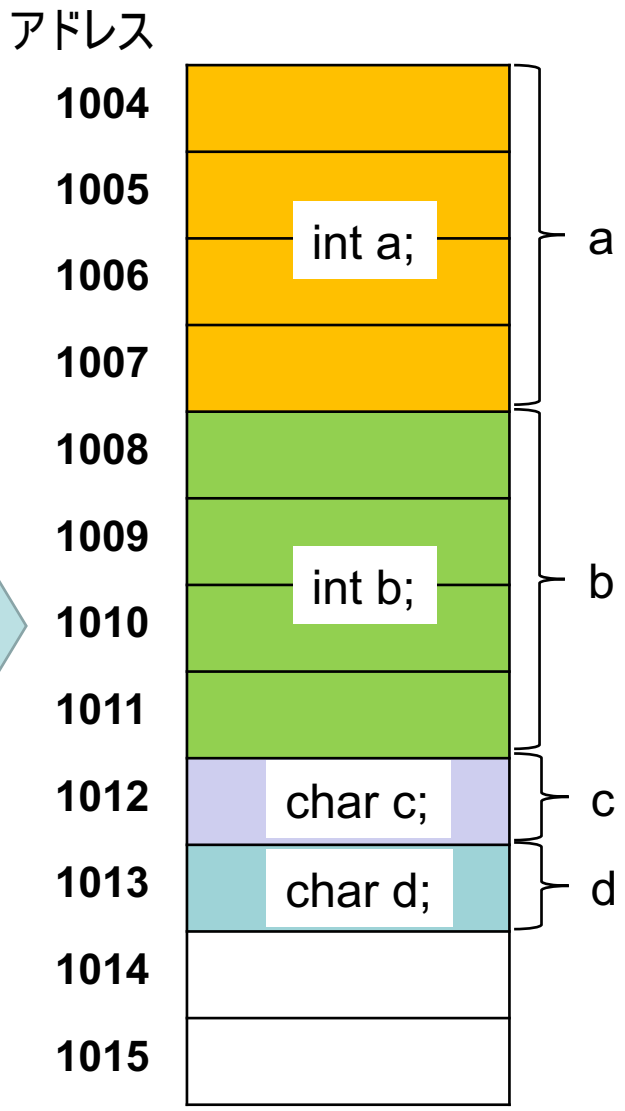
練習：2次元配列 data を4行3列に変更してみよう.（数値は何でも良い）

C言語の変数の仕組みについて(1)

- 変数の型は，使用するメモリ量を決定する
- それぞれの変数には，メモリ番地（アドレス）が割り当てられる

```
int a, b;
char c, d;
```

int は32bit(4バイト)整数
char は8bit（1バイト）整数



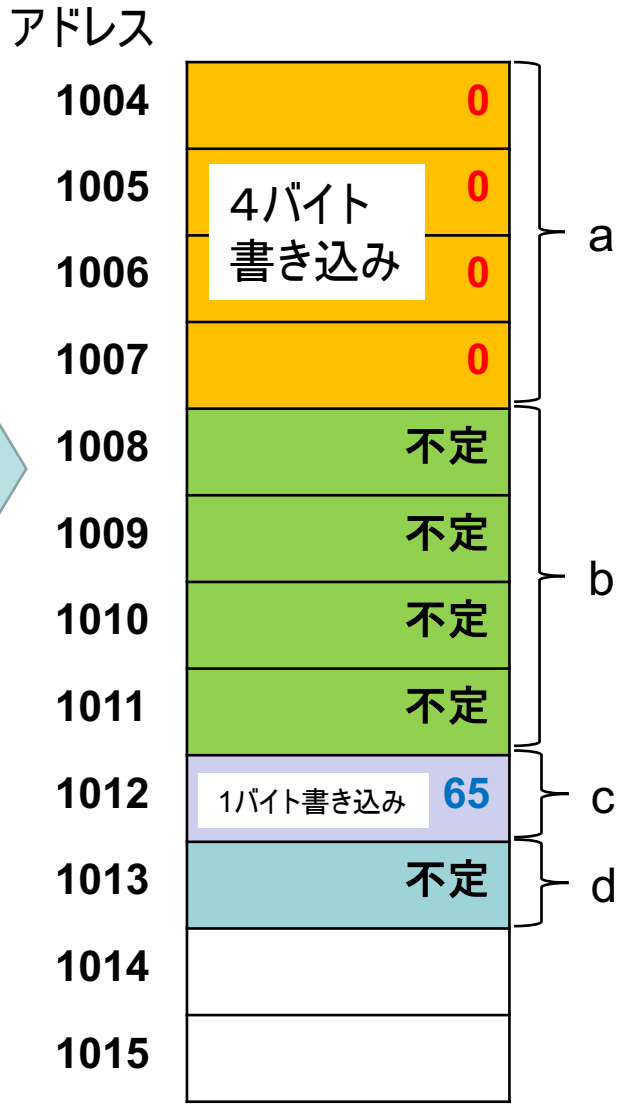
コンパイラが，コンパイル時に，それぞれの変数のメモリアドレスを決める（割り当てる）

C言語の変数の仕組みについて(2)

- 変数に数値を代入すると，
  その変数の型で決まる範囲に
  書き込みされる

```
a = 0;
c = 'A';
```

'A' の文字コードは65



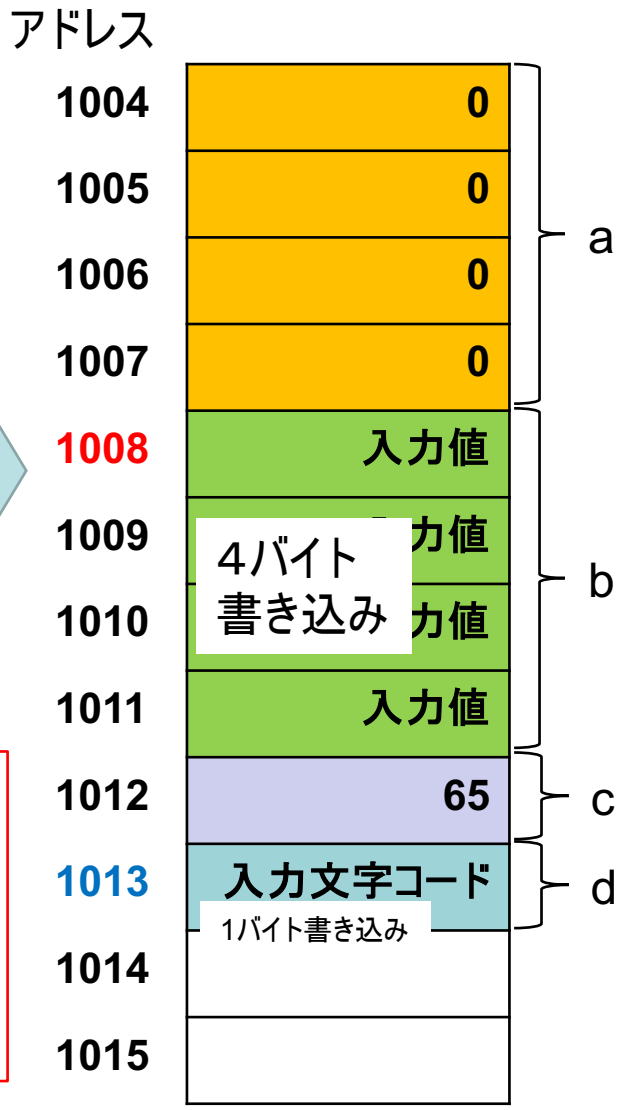
- コンピュータが正常に動作するためには，
  メモリの番地（アドレス）だけでなく，
  書き込む範囲（大きさ）を決める必要がある
- コンパイラは，「型」によって読み書きする
  範囲の大きさを決める

（発展）アドレス渡しでの動作

- scanfは，変数のアドレスを受け取り，%□で決まる範囲に読み込んだ値を書き込む

```
scanf("%d", &b);
scanf("%c", &d);
```

%d は，int（4バイト）の書き込みを行う
%c は，char（1バイト）の書き込みを行う



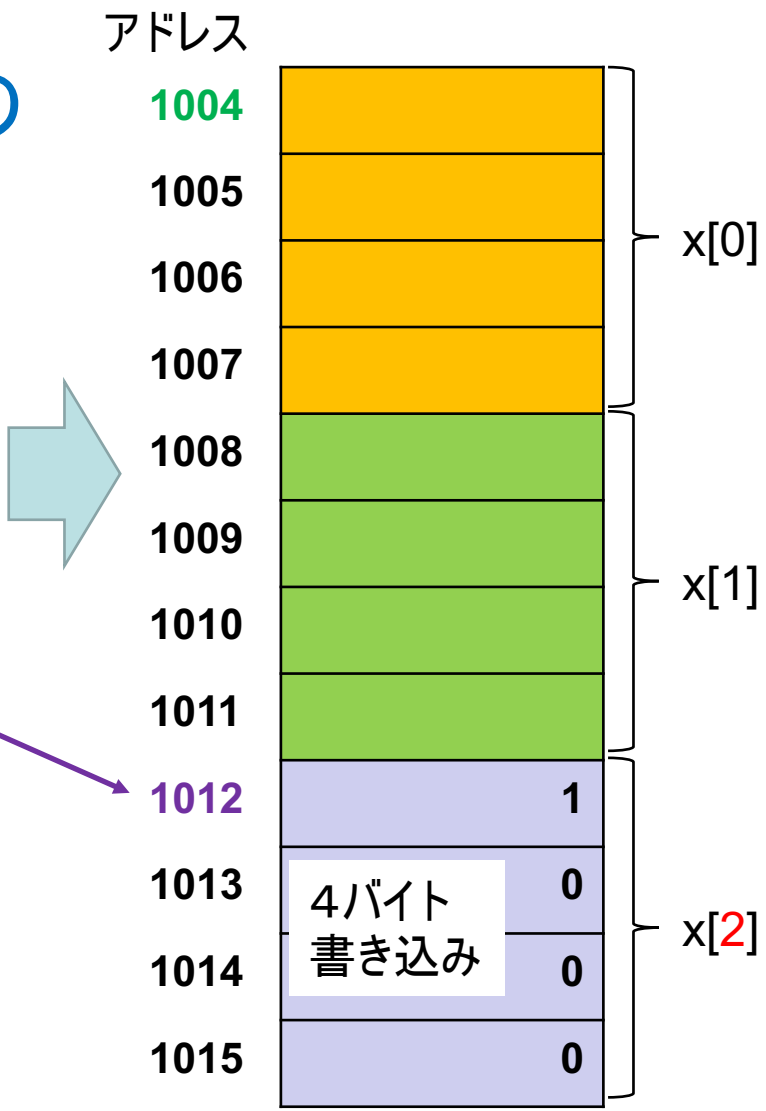
- &演算子（ポインタ演算子）により，変数に割り当てられたアドレスの値が得られる
- scanfは，第1引数のフォーマットにより書き込み先の範囲を知る

配列のアドレス計算

- 配列では，添字の値と，配列の要素の型からアドレス計算する

```
int x[3];
x[2] = 1;
```

アドレス計算は
1004 + 2 * 4 = 1012



アドレス計算に必要な情報
- 配列 x の先頭アドレス　1004
- 配列 x の型が int　4つ飛ばし
- x[2] の添字が 2　2をかける

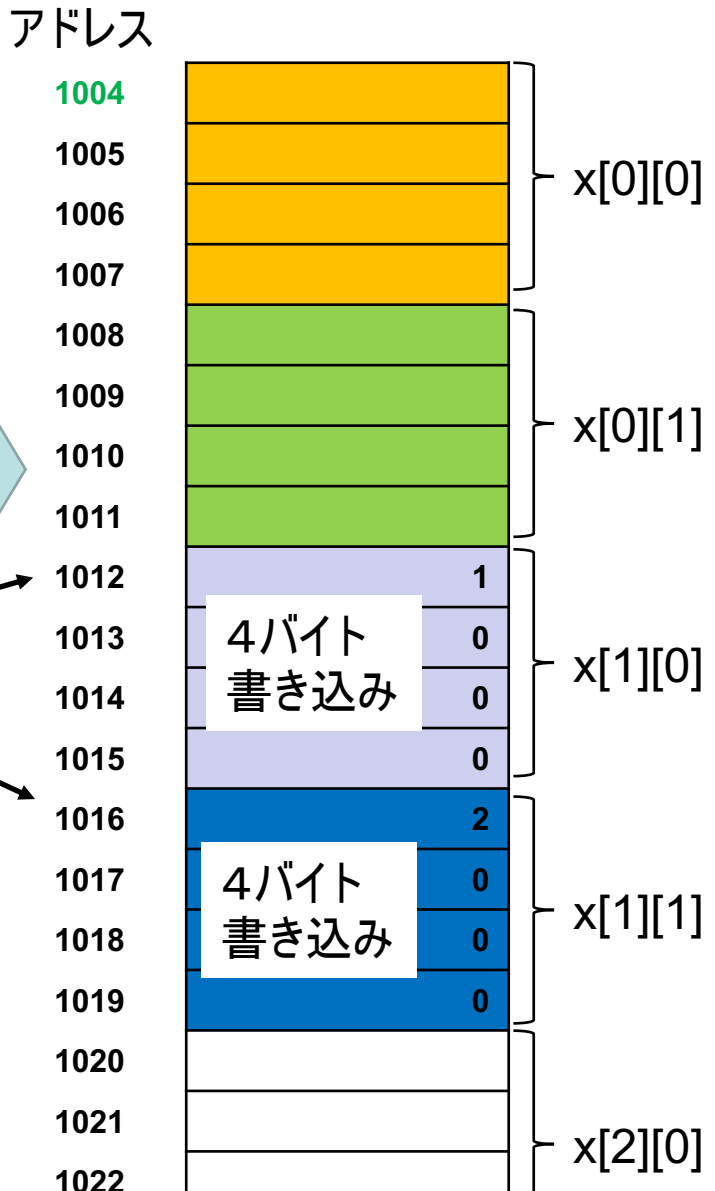
2次元配列のアドレス計算

- 2次元配列では，2つ目の要素数がアドレス計算に必要

```
int x[3][2];
x[1][0] = 1;
x[1][1] = 2;
```

アドレス計算は

```
1004 + (1 * 2 + 0) * 4 = 1012
1004 + (1 * 2 + 1) * 4 = 1016
```



- 2つ目の要素数の値が，1つ目（左）の要素数に掛けられてアドレス計算される
- 1つ目の要素数（左の 3）は，アドレス計算には用いられない

再掲スライド

２次元配列を関数に渡す

引数の[ ]の中の数字の省略について

- 良い例

```
void func(int array[8][8]) { ... }
void func(int array[][8]) { ... }
```

- ダメな例

```
void func(int array[][]) { ... }
```

最初の[ ]の中の添字だけ省略可能

Q:　なぜ最初の [ ] の中の添字だけ省略できるか？
A:　アドレス計算に不要だから

変数の大きさを調べる

- `sizeof()`
  - カッコ内の変数や型の大きさを調べることができる
  - 関数みたいな形だけど，関数ではない（演算子です）
- 使い方１　型の大きさを調べる

  `sizeof(int)` や `sizeof(float)` のように
- 使い方２　変数の大きさを調べる

  ```
  int b; char str[10];
  ```
  `sizeof(b)` や `sizeof(str)` のように

例3　3行2列の行列の初期化と表示　　ex3.c

```
#include <stdio.h>

int main(void) {
  int a, b[4];
  char str[10] = "Hello!";

  printf("size of int is %ld¥n", sizeof(int));
  printf("size of a is %ld¥n", sizeof(a));
  printf("size of int[4] is %ld¥n", sizeof(int [4]));
  printf("size of b is %ld¥n", sizeof(b));
  printf("size of str is %ld¥n", sizeof(str));

  return 0;
}
```

実行結果

```
size of int is 4
size of a is 4
size of int[4] is 16
size of b is 16
size of str is 10
```

練習：2次元配列の大きさを調べてみよう

型の変換(キャスト)

- 型を強制的に変換する

```
float a, b = 1.5;
a = (int)b;
```

とすると

– bの値(1.5)が一旦，強制的にint型に変換される

– そのときに切り捨てが起こるので，aの値は1.0になる

- キャストとは？

cast ： 鋳造する，成形する の意味がある．

( )の中に型の名前を入れて，変数や値の前に付ける．

例4　型の変換（キャスト）と，切り捨ての発生　ex4.c

```
#include <stdio.h>

int main(void) {
  float a, b = 1.5;
  int c = 3;

  a = b;
  printf("%f¥n", a);
  a = (int)b;
  printf("%f¥n", a);

  a = 1/c;
  printf("%f¥n", a);
  a = 1/3.0;
  printf("%f¥n", a);
  a = 1/(float)c;
  printf("%f¥n", a);

  return 0;
}
```

切り捨て発生

切り捨て発生

切り捨て防止

実行結果

```
1.500000
1.000000
0.000000
0.333333
0.333333
```

クリスマスプレゼント

<u>オセロのプログラム</u>

ここまでの講義の知識でほぼ読めるはずです

まだ話していない内容は以下の演算子のみ

- ３項演算子　(項1)？(項2) : (項3)
  - 項1が成立したら（0でなければ）項2の値を，
    そうでなければ項3の値となる演算子．
  - 次回講義（年明け）に説明します

- コンピュータはランダムに手を指すだけです
  - コンピュータ同士の対戦もできます
  - 盤面の大きさを簡単に変えられます

ちょっと解説(1)

- 38行目
  - 乱数で，先手がコンピュータか人かを決めている
    - 変数 turn が，現在のプレーヤーを表す（+1か-1）
- 41行目
  - ゲーム全体のループがここから始まる
- 42-48行目
  - 現在が何手目か，
    現在の盤面，
    プレーヤがコンピューターか，人か　を表示する

ちょっと解説(2)

- 51-61行目
  - プレーヤーに指させる前に，どこかにコマを置けるかどうかを調べている．
  - コマを置ける場所がなければ，強制的にパスさせる
  - パスが2回続いたときは，どちらも指せる場所がないので，終了とする（53行目）
- 64-70行目
  - 人かコンピュータに指させる
  - 関数 human と compRandomを書き換えると，人対人・コンピュータ対コンピュータに変更できる

ちょっと解説(3)

- 83行目～ initBoard()
  - 盤面を初期化する（コマを全部取り除き，中央に4つのコマを配置する）
- 99行目～ showBoard()
  - 盤面を表示する.
    - 盤面の外側に，行数・列数を表示している.
      C言語らしく0始まりにしている.
      %-2dは，２桁（２文字）で，左寄せで表示を意味する
    - コンピュータと人の記号は #define している

ちょっと解説(4)

- 125行目～ 配列 dir[]
  - コマを置いた位置から8方向に進み，敵のコマひっくり返すプログラムを単純化するため，それぞれの方向（0-7）に対応する変化量を配列に格納している
- 137行目～ isPlaceOK()
  - 指定されたマス目に，role で指定されたコマが置けるかどうか調べる
    - 斜めに進んでいき，ボードの外に出たり，敵の駒以外が現れたら終了
    - 自分の駒が現れたときに，それまでに敵の駒があれば，ひっくり返すコマがある＝コマを置くことができる

ちょっと解説(5)

- 166行目～ place()
  - コマを実際に置いて，敵のコマをひっくり返す
  - isPlaceOK()とほぼ同じアルゴリズム
- 201行目～ shouldPass()
  - 盤面上のすべての座標について isPlaceOK()を呼び出し，１箇所でも置けるところがあるかどうか調べる
- 215行目～　human()
  - 座標を人に入力させる
  - ２年生で習う「ポインタ渡し」を使っている（前回講義の最後の方のスライド参照）

ちょっと解説(6)

- 223行目～ compRandom()
  - ランダムに座標を選び，置けるならコマを置く
  - いつかは置けるだろう，というアルゴリズム.
- 239行目～ countNum()
  - ゲーム終了時に，コマ数を数える

趣旨
- 人の書いた，長いプログラムを読んで理解する練習
- 関数を駆使して，プログラムの見通しを良くする方法の例を学ぶ

そして
- 3ヶ月，プログラムを学んだらこんな事ができる！という実感

改造してみよう

- ランダムに打つのではなく・・
  - ４つの角を先に調べ，置けるなら優先的に置く
  - 角のすぐとなりには，できるだけ置かないようにする
- この２つだけでちょっと強くなります．
- さらに（かなり高度だけど，力試し）
  - 自分のコマが最も増えるように置く
  - 相手が指せるコマ数が最小になるように置く

  など．盤面の配列をもう1つ用意して，コピーした上でplace() や isPlaceOK()を呼び出せば，作れる．