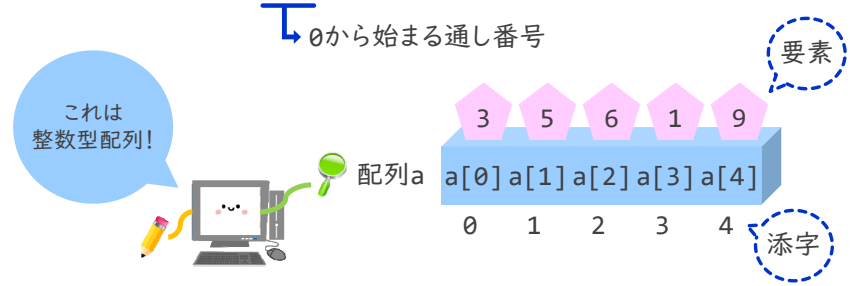
# プログラミング基礎演習A

配列

## 配列

- 同じ型のデータを入れる複数の「箱」の集合
- 入るデータの型によって「〇〇型配列」と呼ぶ
- 配列の要素 ： 各箱に入る値
  他の変数同様に代入や演算に使用可
  配列名[添字]と表記
  └ 0から始まる通し番号






## 配列変数の宣言

型名 配列名[要素数]

- 使用する配列変数の領域を確保する

```
int main()
{
    int a[5];
    double b[3];
    ...
}
```

整数が入る箱を5個,
実数が入る箱を3個
用意するんだね!

要素数
は定数







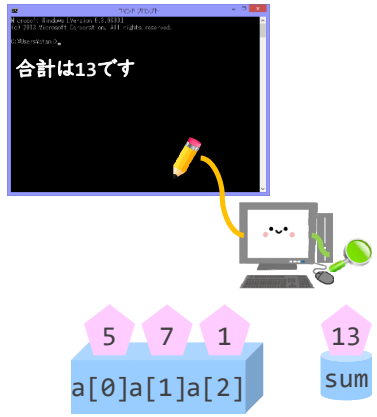
## プログラム例

```
#include <stdio.h>

int main()
{
    int a[3], sum;

    a[0] = 5;
    a[1] = 7;
    a[2] = 1;
    sum = a[0] + a[1] + a[2];

    printf("合計は %d です¥n", sum);
}
```



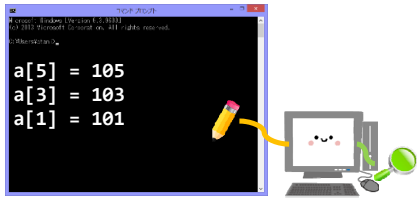
## プログラム例

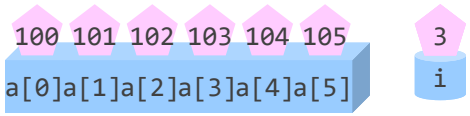
```
#include <stdio.h>

int main()
{
    int a[6], i;

    a[0] = 100;
    a[1] = 101;
    a[2] = 102;
    a[3] = 103;
    a[4] = 104;
    a[5] = 105;

    for(i = 0; i < 3; i++) {
        printf("a[%d] = %d¥n", 5 - i * 2, a[5 - i * 2]);
    }
}
```



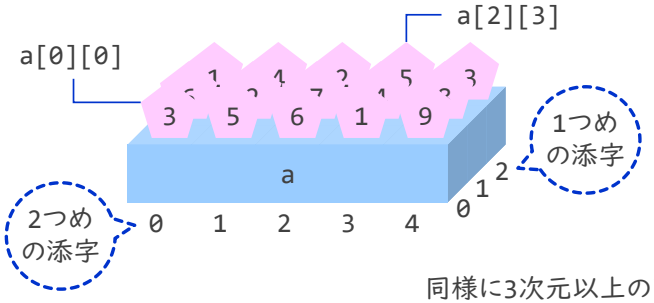

数式も記述可



## 2次元配列

型名 配列名[要素数][要素数]

- 同じ型のデータを入れる複数の箱の集合
- 箱は2次元に並んでいる
- 各箱は2つの添字で管理する




同様に3次元以上の配列も使用可

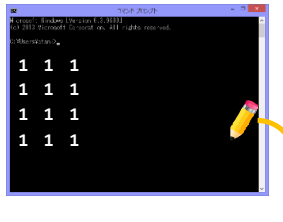


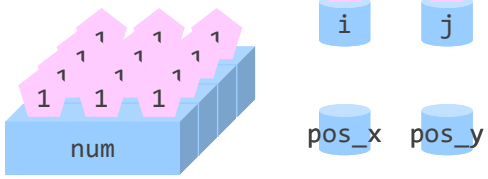
## プログラム例(1)

```
#include <stdio.h>

int main()
{
    int num[4][3];
    int i, j, pos_x, pos_y;

    for(i = 0; i < 4; i++) {
        for(j = 0; j < 3; j++) {
            num[i][j] = 1;
        }
    }
    for(i = 0; i < 4; i++) {
        for(j = 0; j < 3; j++) {
            printf("%d", num[i][j]);
        }
        printf("¥n");
    }
```



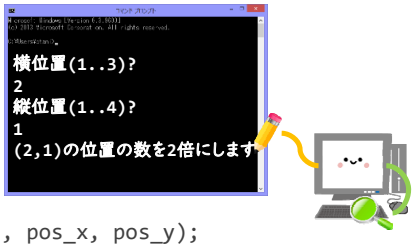



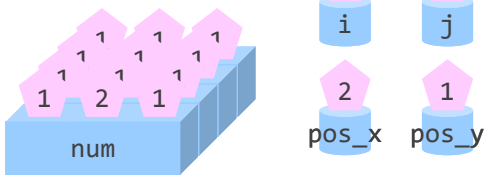



## プログラム例(1)

```
    printf("横位置(1..3)?¥n");
    scanf("%d", &pos_x);
    printf("縦位置(1..4)?¥n");
    scanf("%d", &pos_y);

    printf("(%d,%d)の位置の数を2倍にします¥n", pos_x, pos_y);
    num[pos_y - 1][pos_x - 1] = num[pos_y - 1][pos_x - 1] * 2;
}
```



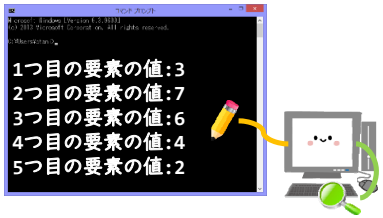



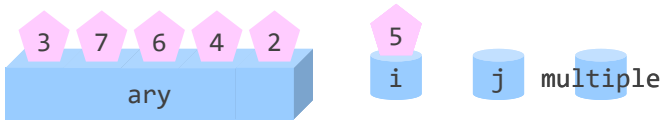
## プログラム例(2)

```
#include <stdio.h>

int main()
{
    int ary[5];
    int i, j;
    int multiple;

    for(i = 0; i < 5; i++) {
        printf("%dつ目の要素の値:", i + 1);
        scanf("%d", &ary[i]);
    }
```





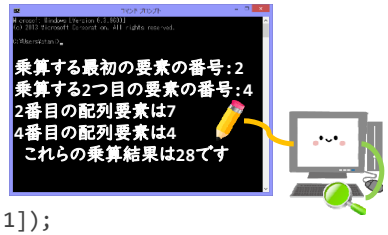

## プログラム例(2)

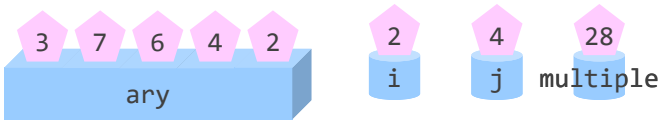
```
    printf("乗算する最初の要素の番号:";
    scanf("%d", &i);

    printf("乗算する2つ目の要素の番号:";
    scanf("%d", &j);

    multiple = ary[i - 1] * ary[j - 1];

    printf("%d番目の配列要素は%d¥n", i, ary[i - 1]);
    printf("%d番目の配列要素は%d¥n", j, ary[j - 1]);
    printf(" これらの乗算結果は%dです¥n", multiple);
}
```



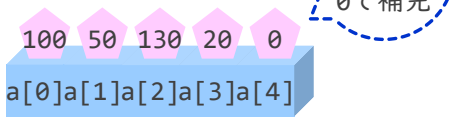



## 配列の初期化

型名 配列名[要素数] = {要素0, 要素1, …}

- 変数の宣言と同時に全要素の初期値を設定する

```
int main()
{
    int a[5] = {100, 50, 130, 20};
    ...
}
```




足りないときは0で補完


- 複数の要素を同時に設定できるのは初期化のみで，代入はできない

```
int a[5];
a[5] = {100, 50, 130, 20};
```



## 多次元配列の初期化

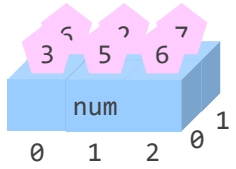
型名 配列名[要素数][要素数] = {{要素0, 要素1,…}, {要素0, 要素1,…},…}

- 変数の宣言と同時に全要素の初期値を設定する

```
int main()
{
    int num[2][3] = {{3, 5, 6}, {6, 2, 7}};
    ...
}
```

1つめの添字が0

1つめの添字が1



## 演習

ベクトルa=(2.5, 3.2, 5.1), ベクトルb=(1.2, 6.6, 3.4)とする. 各ベクトルの長さ, 2つのベクトルの内積, および和を表示するプログラムを作成せよ. ただし, 2つのベクトルの和は配列に格納すること. また, 2つのベクトル, 長さ, および内積を格納する変数は, 初期化子を用いて初期化し, 各値の計算時にはfor文を使うこと.
プログラム名はe8とすること.

<実行例>
ベクトルAの長さ:6.519202
ベクトルBの長さ:7.520638
ベクトルAとベクトルBの内積:41.460000
ベクトルAとベクトルBの和:(3.700000, 9.800000, 8.500000)



## 演習 ～手順～

1. 中身が空のメイン関数, 必要なヘッダファイルを書く
2. 必要なヘッダファイルは?
   - → 入力や出力にはstdio.hが必要
   - → 数学関数の使用にはmath.hが必要
   - → #includeの行を書く
3. 必要な変数は?
   - → ベクトルAとベクトルBを表す変数
   - → ベクトルAの長さとベクトルBの長さを格納する変数
   - → ベクトルAとベクトルBの内積を格納する変数
   - → ベクトルAとベクトルBの和を格納する変数
   - → 繰り返しのためのカウンタ変数
   - → 型と変数名を決めて変数を宣言する



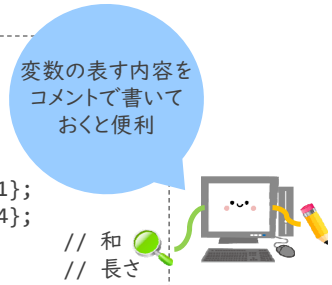
## 演習 ～手順～

- → 型と変数名を決めて変数を宣言する
- → 2つのベクトル, 長さ, および内積を格納する変数は, 初期化子を用いて初期化する

```
#include <stdio.h>
#include <math.h>

int main()
{
    double vecA[3] = {2.5, 3.2, 5.1};
    double vecB[3] = {1.2, 6.6, 3.4};
    double vecC[3];                      // 和
    double lenA = 0.0, lenB = 0.0;       // 長さ
    double ipro = 0.0;                   // 内積
    int i;

}
```






## 演習 ～手順～

4. 処理の順番は?
   - → 以下を3回繰り返す (for文始まり)
     - lenAにvecA[i]の2乗を足す
     - lenBにvecB[i]の2乗を足す
     - iproにvecA[i]とvecB[i]の積を足す
     - vecC[i]にvecA[i]とvecB[i]の和を代入する
   - → lenAの平方根をlenAに代入する
   - → lenBの平方根をlenBに代入する
   - → ベクトルAの長さを表示する (printf)
   - → ベクトルBの長さを表示する (printf)
   - → ベクトルAとベクトルBの内積を表示する (printf)
   - → ベクトルAとベクトルBの和を表示する (printf)