プログラミング実習

第2回: 配列と関数

清水 哲也（ shimizu@info.shonan-it.ac.jp ）

□□□□□□□

- □□□□□□□□□□□□□□□
- □□
- □□
  - □□□□
  - □□□□□
  - □□□□□□□□□□
- □□

前回の授業課題と宿題の解答例

ざっくりと説明します

[illegible]

[illegible]

[illegible]1

5[illegible]

□□□□1□□□□□□□□□□□□□□□

```c
// 5□□□□□□□□□□□□□□□□□□□□□□□□□□□□
#include <stdio.h>

int main(void)
{
    int point1;  // 1□□□□
    int point2;  // 2□□□□
    int point3;  // 3□□□□
    int point4;  // 4□□□□
    int point5;  // 5□□□□
    int sum = 0; // □□□

    printf("5□□□□□□□□□□□□\n");
    printf("1□□"); scanf("%d", &point1); sum += point1;
    printf("2□□"); scanf("%d", &point2); sum += point2;
    printf("3□□"); scanf("%d", &point3); sum += point3;
    printf("4□□"); scanf("%d", &point4); sum += point4;
    printf("5□□"); scanf("%d", &point5); sum += point5;

    printf("□□□□%5d\n", sum);
    printf("□□□□%5.1f\n", (double)sum / 5);

    return 0;
}
```

配列：例1の補足

前のページのプログラムで補足です．

```
    printf("1番: "); scanf("%d", &point1); sum += point1;
```

この部分は，普通は以下のように書きます．

```
    printf("1番: ");
    scanf("%d", &point1);
    sum += point1;
```

スペース的な問題や慣れてくると上のような書き方をしてもいいと思います．

配列の宣言（配列を使うための準備）

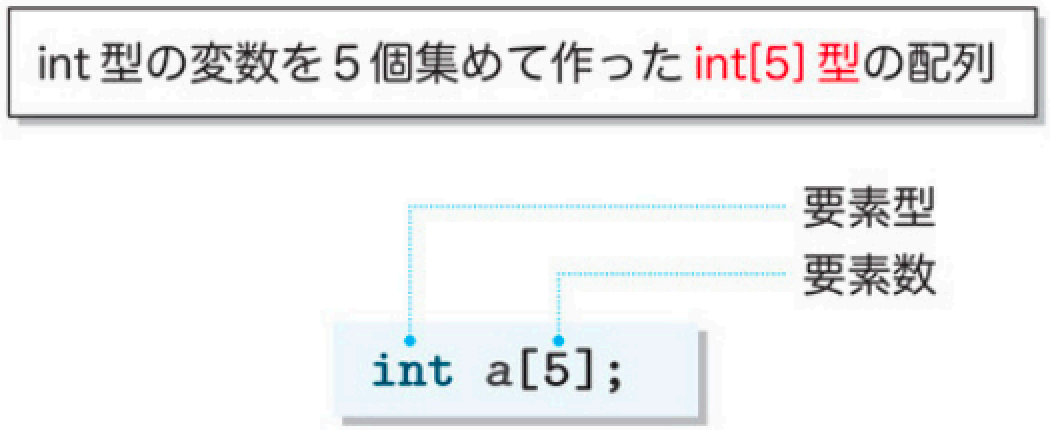
例1のプログラムでは，5人のデータなので書いてもそこまで苦労はしないです．

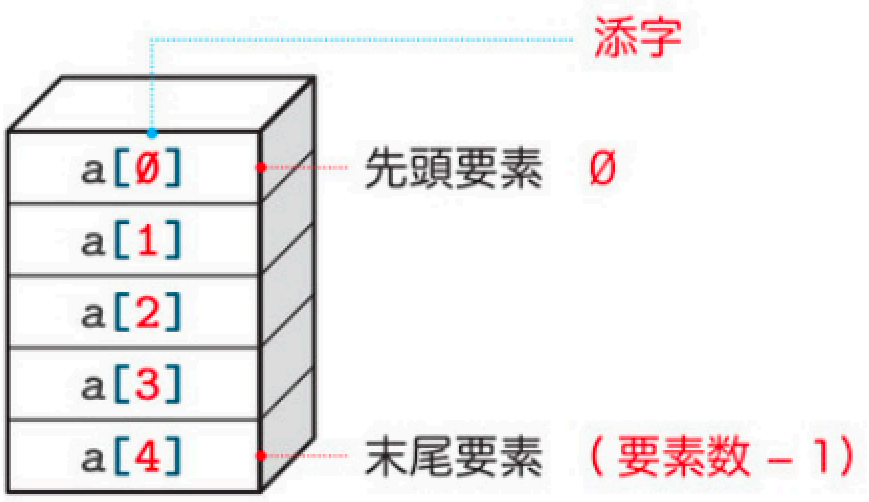
これがこのクラス全員となる約70人分になるので変数を70個用意する必要ことになります．

ここで配列を用います．

配列は，要素型，配列名，要素数で宣言します．

```
要素型 配列名 [ 要素数 ] ;
```





[illegible]

[illegible]

[illegible]`a[b]` [illegible]`a`[illegible]`b`[illegible]

[illegible]`[illegible]-1=[illegible]`[illegible]

[illegible]`int a[5]`[illegible]

[illegible]`a[0], a[1], a[2], a[3], a[4]`[illegible]`a[5]`[illegible]

□□□□□(1)

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```c
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□
#include <stdio.h>
int main(void)
{
    int a[5]; // int[5]□□□□
    a[0] = 1;
    a[1] = 2;
    a[2] = 3;
    a[3] = 4;
    a[4] = 5;

    printf("a[0] = %d\n", a[0]);
    printf("a[1] = %d\n", a[1]);
    printf("a[2] = %d\n", a[2]);
    printf("a[3] = %d\n", a[3]);
    printf("a[4] = %d\n", a[4]);
    return 0;
}
```

配列の走査(2)

前のページのプログラムでもいいのですが，これは非効率的です．繰り返しの部分を `for` 文を使って書き換えます．

```c
// 配列の各要素に先頭から順に1～5を代入して表示(for文を使う)
#include <stdio.h>

int main(void)
{
    int a[5]; // int[5]型の配列

    for (int i = 0; i < 5; i++) // 要素に値を代入
        a[i] = i + 1;

    for (int i = 0; i < 5; i++) // 要素の値を表示
        printf("a[%d] = %d\n", i, a[i]);

    return 0;
}
```

配列の初期化

配列の各要素へ値を代入するのではなく，初期化で設定する方法があります．

```
要素型 配列名[要素数] = {値1, 値2, ...};
```

このように配列の宣言と同時に各要素に値を入れることができます．

`要素数`を省略することもでできます．

```
int a[] = {1, 2, 3};
```

この場合は，各要素の値の数が自動的に配列の要素数となります．

各に要素にすべて同じ値を入れることもできます．

```
int b[5] = {0};
```

この場合は，`要素数`を指定必要があります．

□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```c
// □□□□□□□□□□□□□□□□□□
#include <stdio.h>

int main(void)
{
    int x[5];

    for (int i = 0; i < 5; i++) // □□□□□□□□□□
    {
        printf("x[%d] : ", i);
        scanf("%d", &x[i]);
    }

    for (int i = 0; i < 5; i++) // □□□□□□□□
        printf("x[%d] = %d\n", i, x[i]);

    return 0;
}
```

□□□□1□□□□□□□□□□□□□

```c
// □□□□□□□□1□□□□□□□□
#include <stdio.h>
#define NUM 5

int main(void)
{
    int point[NUM]; // int[5]□□□□□□□
    int sum = 0;    // □□□

    printf("%d□□□□□□□□□□□□\n", NUM);

    for (int i = 0; i < NUM; i++)
    {
        printf("%d□□", i + 1);
        scanf("%d", &point[i]);
        sum += point[i];
    }

    printf("□□□□%5d\n", sum);
    printf("□□□□%5.1f\n", (double)sum / NUM);

    return 0;
}
```

[illegible]

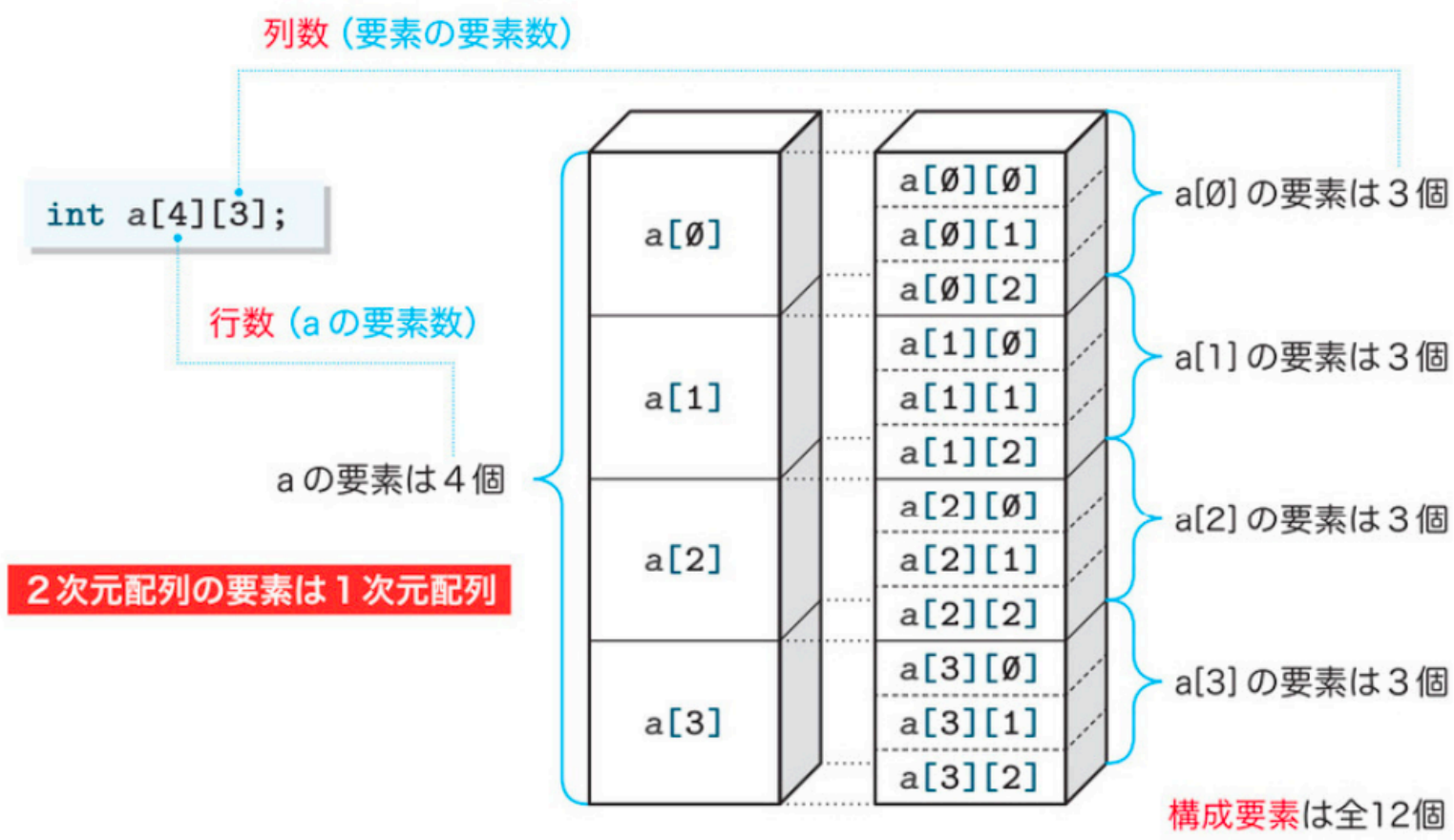
[illegible]

[illegible]**2**[illegible]2[illegible]**3**[illegible]
[illegible]3[illegible]
[illegible]2[illegible]

```
[illegible] [illegible][[illegible]][[illegible]];
```

□□□□□

```
int a[4][3];
```


列数（要素の要素数）
int a[4][3];
行数（a の要素数）
a の要素は４個
２次元配列の要素は１次元配列
a[0]
a[1]
a[2]
a[3]
a[0][0]
a[0][1]
a[0][2]
a[1][0]
a[1][1]
a[1][2]
a[2][0]
a[2][1]
a[2][2]
a[3][0]
a[3][1]
a[3][2]
a[0] の要素は３個
a[1] の要素は３個
a[2] の要素は３個
a[3] の要素は３個
構成要素は全12個

多次元配列

注意

`int a[4][3]`と`int a[3][4]`は全要素数は変わらないですが構造が違うの注意が必要です.

重要

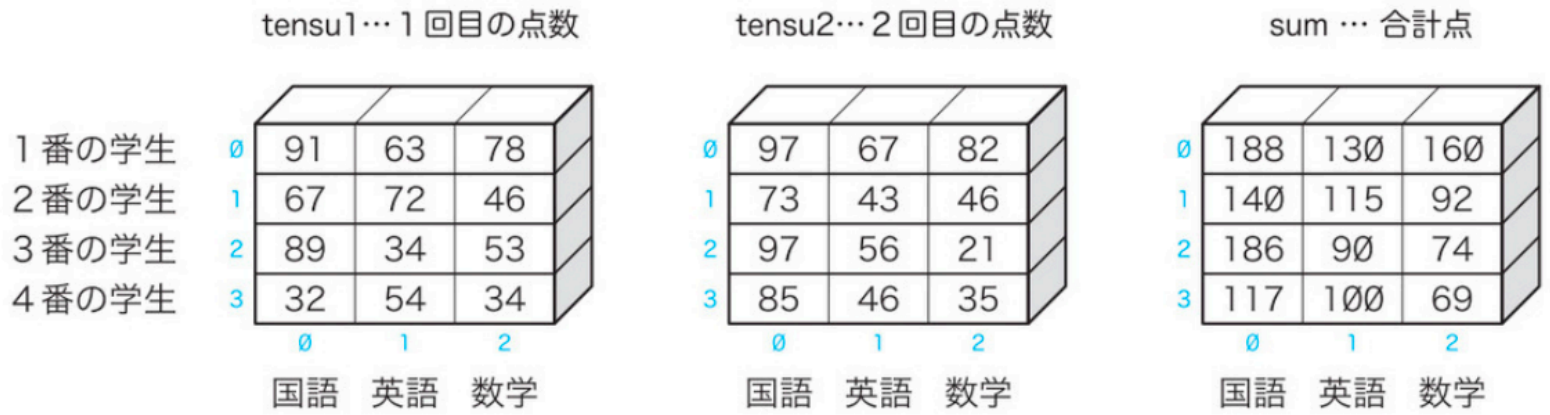
多次元配列の構成要素は，末尾側の添字が優先的に増えていく順に並ぶ.

`a[0][0] a[0][1] a[0][2]` `a[1][0] a[1][1] a[1][2]` ･･･ `a[3][0] a[3][1] a[3][2]`

多次元配列

2次元配列のプログラム例です．

- テストの点数の合計を求めるプログラムです
- 学生は4人です
- 教科は3教科です
- テストの回数は2回です
- 科目別の合計を求めて表示します



tensu1… 1回目の点数

		国語 0	英語 1	数学 2
1番の学生	0	91	63	78
2番の学生	1	67	72	46
3番の学生	2	89	34	53
4番の学生	3	32	54	34

tensu2… 2回目の点数

	国語 0	英語 1	数学 2
0	97	67	82
1	73	43	46
2	97	56	21
3	85	46	35

sum … 合計点

	国語 0	英語 1	数学 2
0	188	130	160
1	140	115	92
2	186	90	74
3	117	100	69

main関数とライブラリ関数

main関数(main function)

プログラムに1個だけ必要であり，プログラムの実行時に本体部が実行されます．

ライブラリ関数(library function)

C言語によって標準で提供されている関数の集まりです．

例: `printf` 関数，`scanf` 関数など

[illegible]

[illegible]
[illegible]

> [illegible]

[illegible]
[illegible]2[illegible]
[illegible]1[illegible]

関数定義

例の関数を作成すると以下のようになります.

```
int max2(int a, int b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

ここでは，関数の構造を理解します．そのために，まずは用語の説明をします.

□□□□

□□□□(function header)

□□□□□□□□ □□ □□□□□□□□□□

```
int max2 (int a, int b)
```

□□□□(function body)

□□□□□□□□□□□□□□□□□□□ □□□ □□□

```
    if (a > b)
        return a;
    else
        return b;
```

関数定義

関数頭部内を詳しくみていきます.

返却値型**(return type)**

関数が戻す値である 返却値**(return value)** の型です．関数頭部の先頭の `int` がそれです.

関数名**(function name)**

関数の名前です．この名前を使って他のプログラムから呼び出します．例の場合 `max2` がそれです.

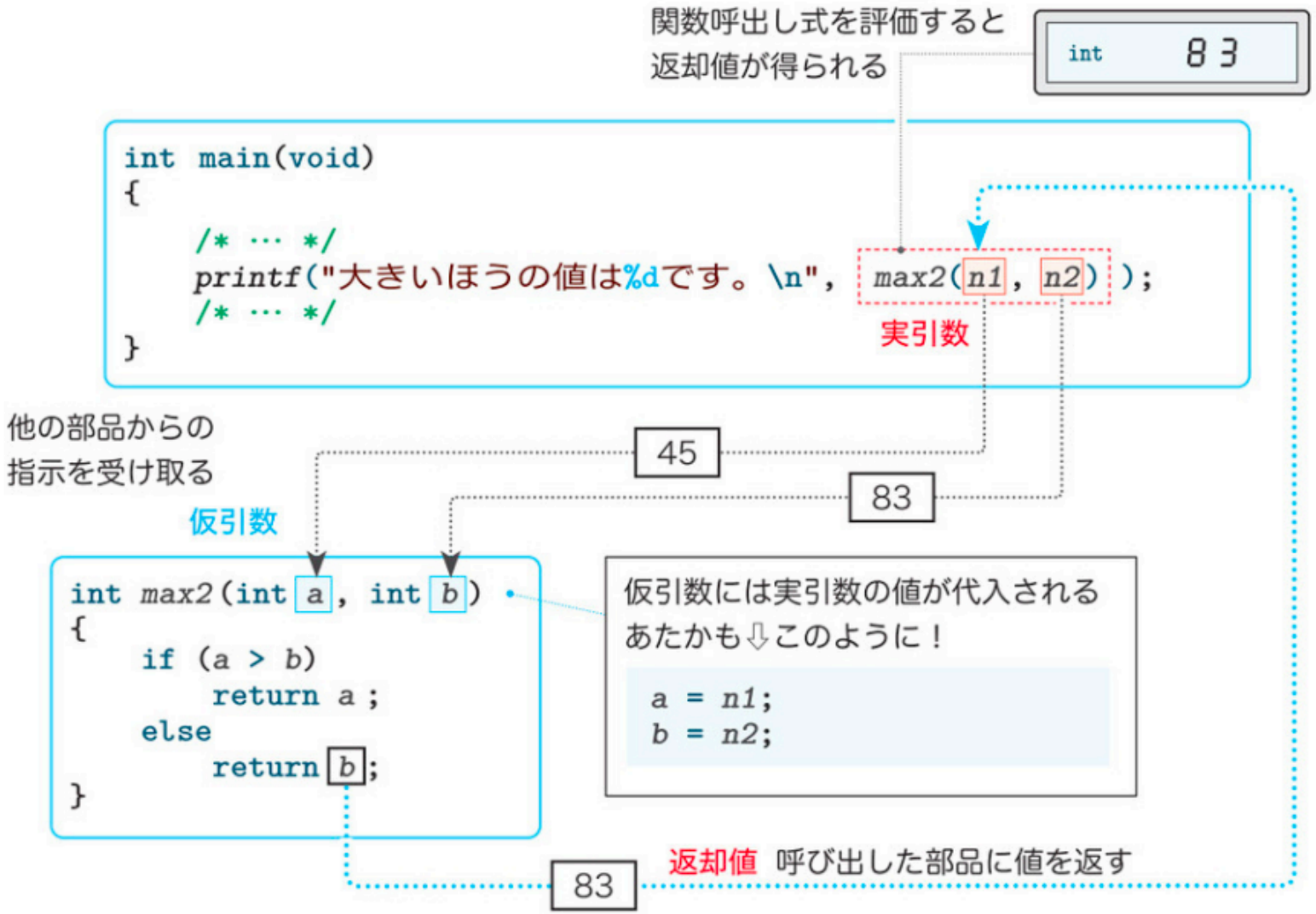
仮引数型並び**(parameter type list)**

カッコの中にあって，値などを受け取るための変数である 仮引数**(parameter)** の宣言をします．例の場合 `int a, int b` がそれです.

□□□□□

```c
#include <stdio.h>

int max2(int a, int b){
    if (a > b) return a;
    else return b;
}

int main(void){
    int n1, n2;

    puts("□□□□□□□□□□□□□□□□□□□□");
    printf("□□1□"); scanf("%d", &n1);
    printf("□□2□"); scanf("%d", &n2);

    printf("□□□□□□□□%d□□□\n", max2(n1, n2));
    return 0;
}
```

□□□□□


関数呼出し式を評価すると
返却値が得られる
int 83
int main(void)
{
/* … */
printf("大きいほうの値は%dです。\n", max2(n1, n2));
/* … */
}
実引数
他の部品からの
指示を受け取る
仮引数
45
83
int max2(int a, int b)
{
if (a > b)
return a;
else
return b;
}
仮引数には実引数の値が代入される
あたかも⇩このように！
a = n1;
b = n2;
83
返却値 呼び出した部品に値を返す

□□□□□

重要 関数呼出しが行われると、プログラムの流れは呼び出された関数に移る。その際、呼出し側が与えた実引数の値が、関数が受け取る仮引数に代入される。

重要 return 文は、関数の実行を終了させて、プログラムの流れを呼出し元に戻すとともに値を返却する。その返却値は、関数呼出し式の評価によって得られる。

return文と返却値

関数が返す（返却）するのは，`return` に続く 式 の値です．

`return` の後ろに置く 式 は0個か1個で，2個以上のの値を返すことができません．

a "chap06/list0601a.c"

```
int max2(int a, int b)
{
    int max;

    if (a > b)
        max = a;
    else
        max = b;

    return max;
}
```

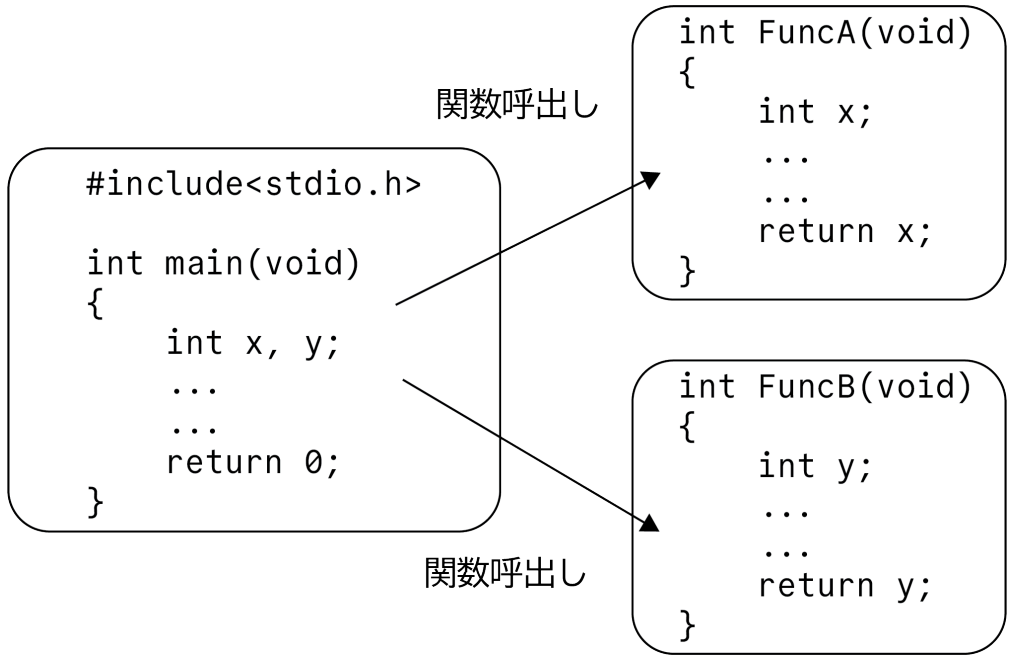
b "chap06/list0601b.c"

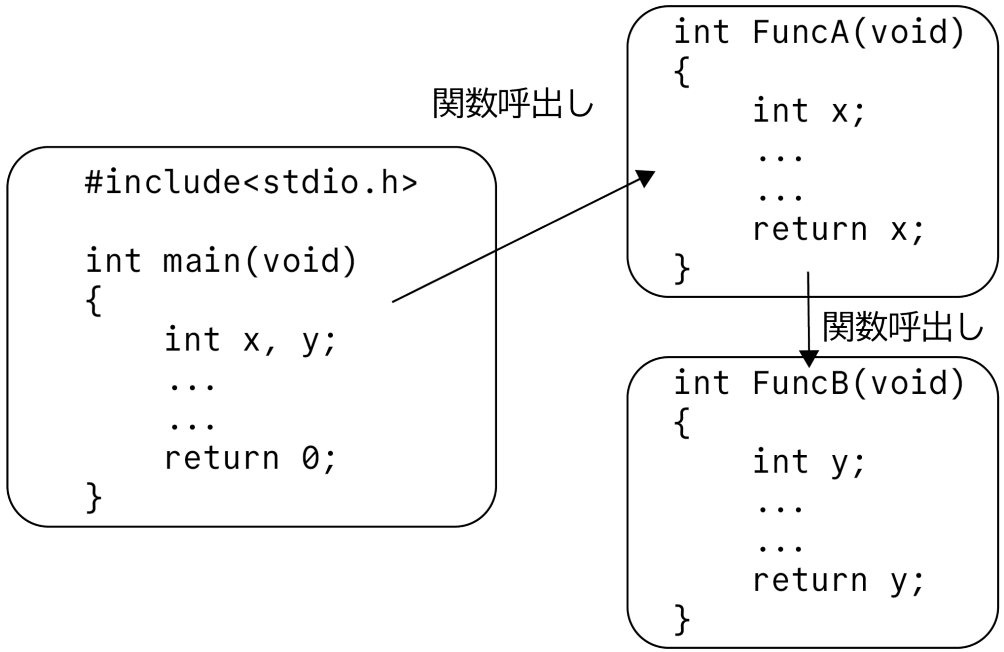
```
int max2(int a, int b)
{
    int max = a;

    if (b > max)
        max = b;

    return max;
}
```

c "chap06/list0601c.c"

```
int max2(int a, int b)
{
    return a > b ? a : b;
}
```

□□□□□□□□

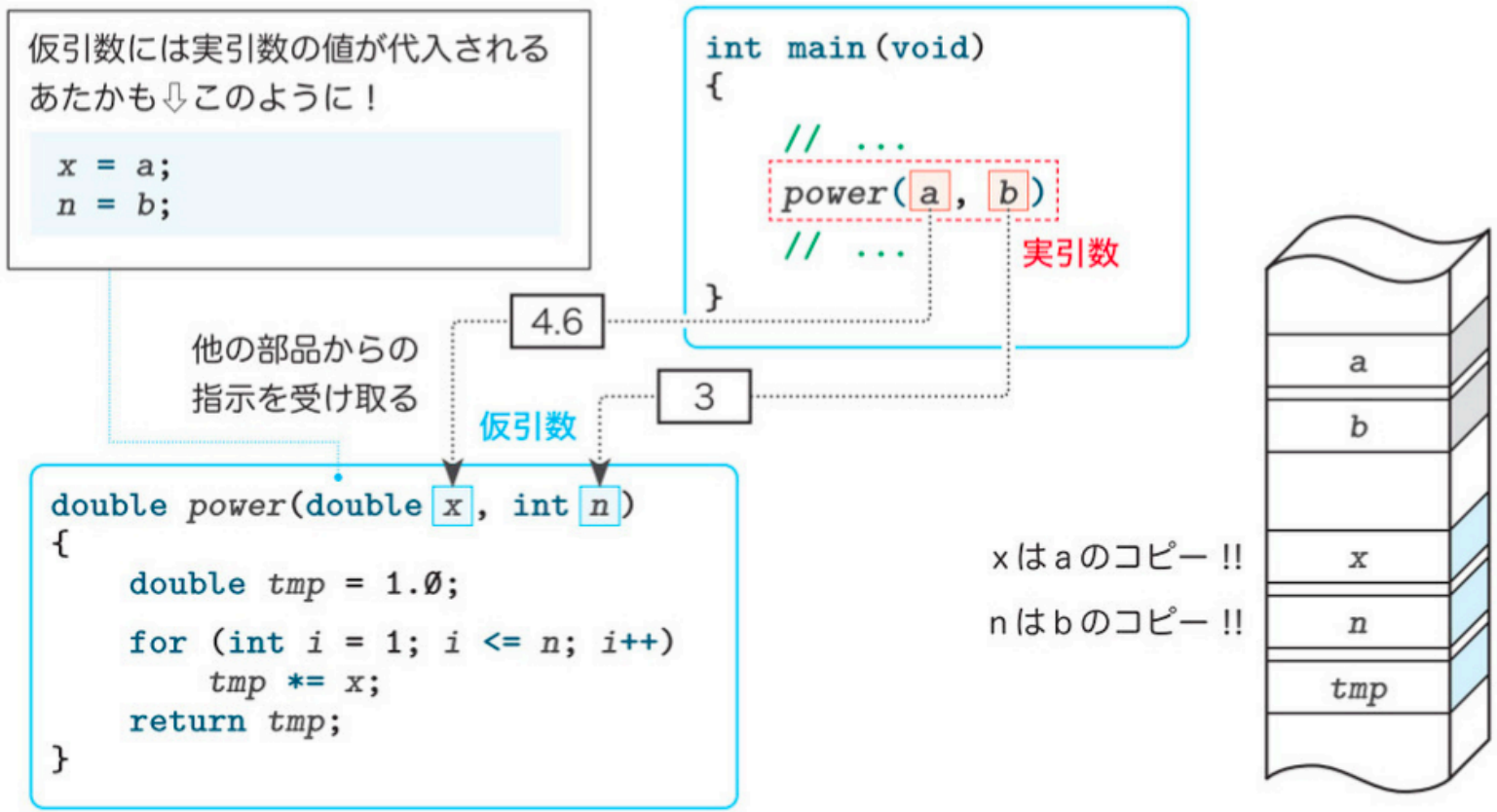

関数呼出し
関数呼出し
#include<stdio.h>
int main(void)
{
int x, y;
...
...
return 0;
}
int FuncA(void)
{
int x;
...
...
return x;
}
int FuncB(void)
{
int y;
...
...
return y;
}


関数呼出し
関数呼出し
#include<stdio.h>
int main(void)
{
int x, y;
...
...
return 0;
}
int FuncA(void)
{
int x;
...
...
return x;
}
int FuncB(void)
{
int y;
...
...
return y;
}

□□□

```c
#include <stdio.h>

double power(double x, int n)
{
    double tmp = 1.0;

    for (int i = 1; i <= n; i++)
    {
        tmp *= x; // tmp□x□□□□
    }
    return tmp;
}

int main(void)
{
    double a;
    int b;

    printf("a□b□□□□□□.\n");
    printf("□□a□"); scanf("%lf", &a);
    printf("□□b□"); scanf("%d", &b);

    printf("%.2f□%d□□%.2f□□□\n", a, b, power(a, b));

    return 0;
}
```

□□□


仮引数には実引数の値が代入される
あたかも⇩このように！
x = a;
n = b;
int main(void)
{
// ...
power(a, b)
// ...
}
実引数
4.6
3
他の部品からの
指示を受け取る
仮引数
double power(double x, int n)
{
double tmp = 1.0;
for (int i = 1; i <= n; i++)
tmp *= x;
return tmp;
}
xはaのコピー !!
nはbのコピー !!
a
b
x
n
tmp

値を返さない関数，関数の汎用性

関数は値を返さない関数もある．

例えば，作成した関数で表示して返すものがない場合

その場合は，返却値型は，`void` となります．

`void` は，「空の」という意味があります．

重要：値を返却しない関数の返却値型は **`void`** とする．

関数は **1**つの目的だけ をおこなうのではなく，さまざまなデータ型やデータ構造を処理したり，さまざまな入力条件で実行することで 汎用性の高い（さまざまな目的で使用できる） 関数となります．

重要：関数はなるべく 汎用性 の高いものにする

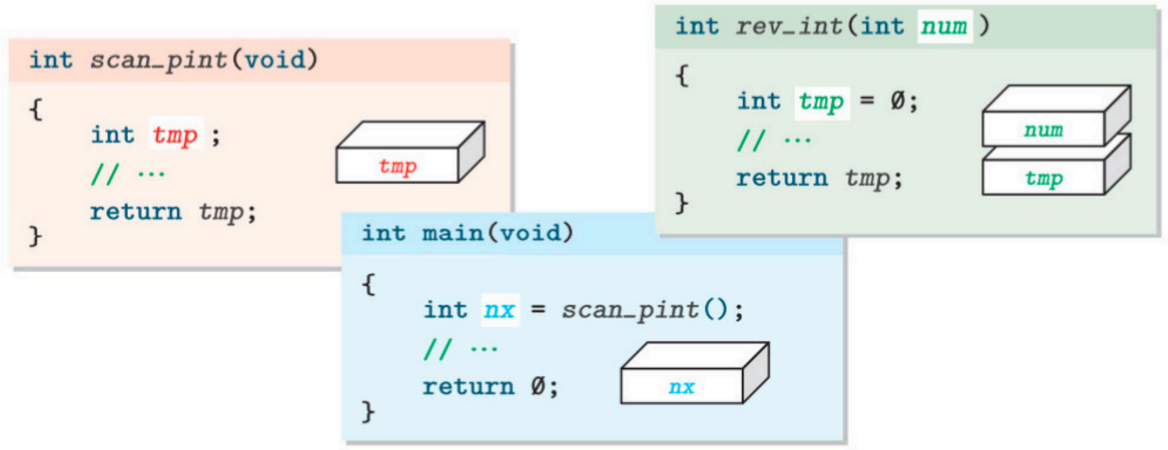
□□□□□□□□□□□□□□□□□□□□

```c
// □□□□□□□□□□□□□□□□□□□□
#include <stdio.h>

//--- □□□□'*'□n□□□□□□□ ---//
void put_starts(int n)
{
    while ((n-- > 0))
        putchar('*');
}

int main(void)
{
    int len;

    printf("□□□□□□□□□□□□□□□□□□□□\n");
    printf("□□□"); scanf("%d", &len);

    for (int i = 1; i <= len; i++)
    {
        put_starts(i);
        putchar('\n');
    }

    return 0;
}
```

引数を受け取らない関数，プロック有効範囲

関数には，引数を受け取らす呼出されたら関数を実行してその結果を返すだけのものがある．その場合，仮引数がないため `()` の中が `void` と宣言されている．

重要：引数を受け取らない関数は，仮引数型を **`void`** と宣言する



重要：ブロックの中で宣言された変数の名前には，宣言された場所から，その宣言を囲むブロック終端の `}` まで通用する ブロック有効範囲 が与えられる

□□□□□□□□□□□□□□□□□□□□□□□□□□

```c
#include <stdio.h>

int scan_pint(void){
    int tmp;
    do {
        printf("□□□□□□□□□□□□□□□□");
        scanf("%d", &tmp);
        if (tmp <= 0)
            puts("\a□□□□□□□□□□□□□□□□□□□□");
    } while (tmp <= 0);
    return tmp;
}

int rev_int(int num) {
    int tmp = 0;
    if (num > 0) {
        do {
            tmp = tmp * 10 + num % 10;
            num /= 10;
        } while (num > 0);
    } return tmp;
}

int main(void){
    int nx = scan_pint();
    printf("□□□□□□%d□□□\n", rev_int(nx));
    return 0;
}
```

ファイル有効範囲，関数原型宣言

ファイル有効範囲

有効範囲には，色々な種類があります．先程説明したブロック有効範囲の他に ファイル有効範囲 などがあります．ファイル有効範囲は，ソースコードが書かれているファイル内で有効になります．

関数原型宣言

`main` 関数より後（下）に関数を書くときに，その関数があることを示すために宣言するものです．ですがこの書き方は推奨しません．変更をするときに煩雑になるためです．

□□□□□□□□□□□□□□□□□□

```c
#include <stdio.h>

#define NUMBER 5 // □□□□□

int point[NUMBER]; // □□□□□

int top(void); // □□top□□□□□□□

int main(void){
    extern int point[]; // □□□□□□□□□□□

    printf("%d□□□□□□□□□□□□□□□□□\n", NUMBER);
    for (int i = 0; i < NUMBER; i++) {
        printf("%d□", i + 1); scanf("%d", &point[i]);
    }
    printf("□□□□%d\n", top());
    return 0;
}

int top(void)
{
    extern int point[]; // □□□□□□□□□□□
    int max = point[0];

    for (int i = 0; i < NUMBER; i++)
        if (point[i] > max)
            max = point[i];
    return max;
}
```

配列の受渡し

関数を作るときに汎用性が重要であり，汎用性を高めることはその関数が独立していることになります．この独立しているとは，関数の外で定義された変数などの情報に依存しない という意味です．先程の例では，点数の配列をファイル内で使える配列として宣言しましたが，関数 `top` からすると得体の知らない配列を扱うことになります．なので，これはあまりおすすめしません．でも配列を関数間で使用したので配列の受け渡しをします．

□□□□□□

```c
#include <stdio.h>

#define NUMBER 5 // □□□□□

int max_of(int v[], int n){
    int max = v[0];

    for (int i = 0; i < n; i++)
        if (v[i] > max)
            max = v[i];
    return max;
}

int main(void){
    int eng[NUMBER]; // □□□□□
    int mat[NUMBER]; // □□□□□

    printf("%d□□□□□□□□□□□□□□□□□\n", NUMBER);
    for (int i = 0; i < NUMBER; i++) {
        printf("[%d] □□□", i + 1); scanf("%d", &eng[i]);
        printf("     □□□");        scanf("%d", &mat[i]);
    }
    int max_e = max_of(eng, NUMBER); // □□□□□□
    int max_m = max_of(mat, NUMBER); // □□□□□□

    printf("□□□□□□□%d\n", max_e);
    printf("□□□□□□□%d\n", max_m);

    return 0;
}
```

配列の受渡しと `const` 型修飾子

重要：呼び出された側の関数で仮引数として受け取った配列は，呼び出した側で与えた 実引数の配列そのもの です．

なので，受け取った配列に変更を加えると元の配列にも影響がでます．

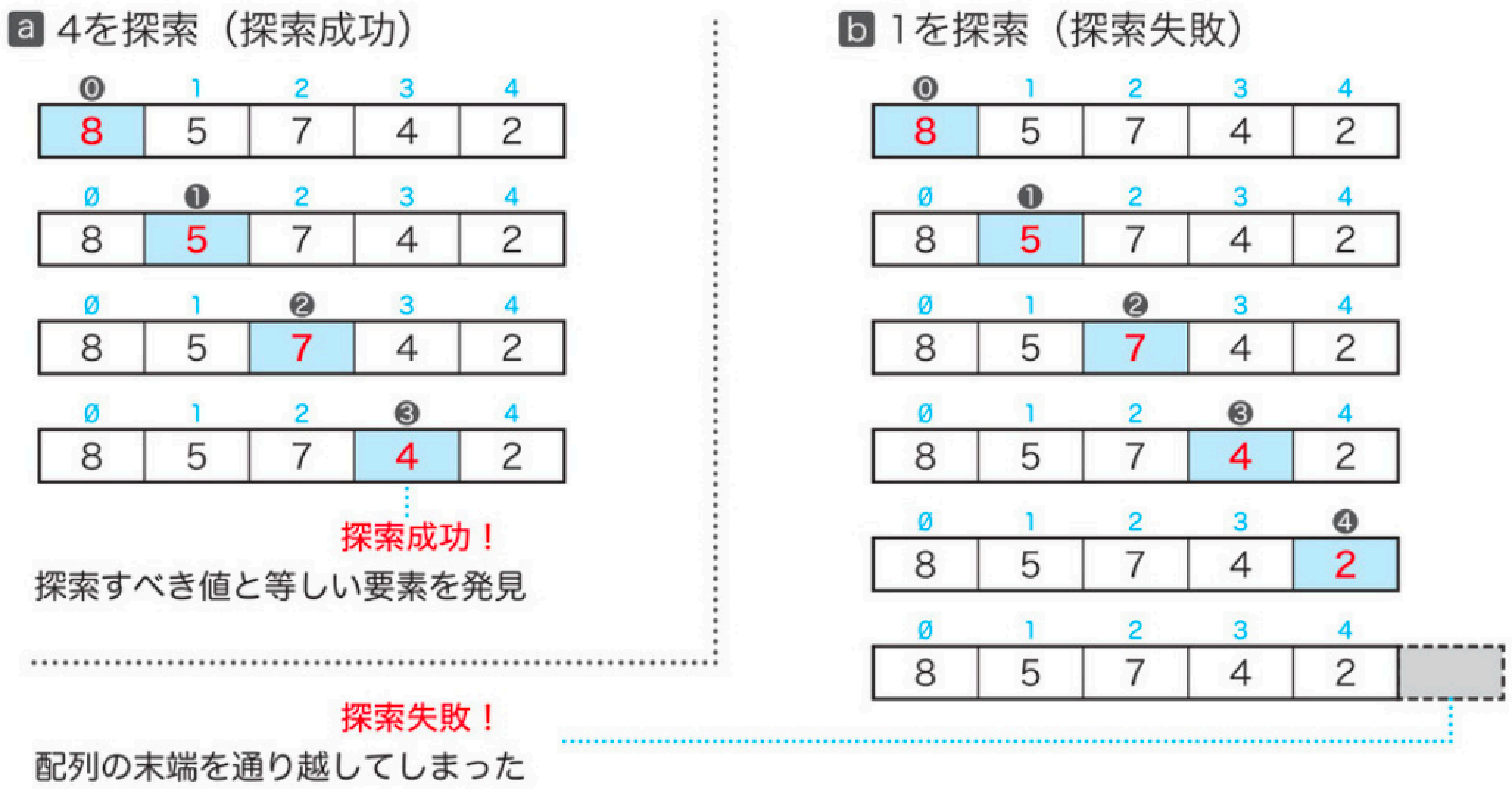
影響を与え内容にするのが，`const` 型修飾子です．

重要：仮引数に受け取った配列の要素の値を読み取るだけで書き込まないのであれば，その仮引数は **`const`** 付きで宣言します．

□□□□□□□ const □□□

```c
#include <stdio.h>

void set_zero(int v[], int n){
    for (int i = 0; i < n; i++)
        v[i] = 0;
}

void print_array(const int v[], int n) {
    printf("{");
    for (int i = 0; i < n; i++)
        printf("%d ", v[i]);
    printf("}\n");
}

int main(void){
    int ary1[] = {1, 2, 3, 4, 5};
    int ary2[] = {3, 2, 1};

    printf("ary1 = "); print_array(ary1, 5);
    printf("ary2 = "); print_array(ary2, 3);

    set_zero(ary1, 5);
    set_zero(ary2, 3);

    printf("□□□□□□□□0□□□□□□□□\n");
    printf("ary1 = "); print_array(ary1, 5);
    printf("ary2 = "); print_array(ary2, 3);
    return 0;
}
```

線形探索（逐次探索）


a 4を探索（探索成功）
0 1 2 3 4
8 5 7 4 2
0 ❶ 2 3 4
8 5 7 4 2
0 1 ❷ 3 4
8 5 7 4 2
0 1 2 ❸ 4
8 5 7 4 2
探索成功！
探索すべき値と等しい要素を発見
b 1を探索（探索失敗）
0 1 2 3 4
8 5 7 4 2
0 ❶ 2 3 4
8 5 7 4 2
0 1 ❷ 3 4
8 5 7 4 2
0 1 2 ❸ 4
8 5 7 4 2
0 1 2 3 ❹
8 5 7 4 2
0 1 2 3 4
8 5 7 4 2
探索失敗！
配列の末端を通り越してしまった

□□□□□□□□□□□

```c
#include <stdio.h>
#define NUMBER 5  // □□□
#define FAILED -1 // □□□□

int search(const int v[], int key, int n) {
    int i = 0;
    while (1) {
        if (i == n)      return FAILED; // □□□□
        if (v[i] == key) return i; // □□□□
        i++;
    }
}

int main(void) {
    int ky, idx;
    int x[NUMBER];

    for (int i = 0; i < NUMBER; i++) {
        printf("x[%d]□", i); scanf("%d", &x[i]);
    }
    printf("□□□□"); scanf("%d", &ky);

    idx = search(x, ky, NUMBER); // □□□NUMBER□□□x□□ky□□□

    if (idx == FAILED)
        puts("\a□□□□□□□□□□□□");
    else
        printf("%d□%d□□□□□□□□□\n", ky, idx + 1);
    return 0;
}
```

有効範囲と識別子の可視性

重要：ファイル有効範囲とブロック有効範囲をもつ同じ名前の変数が存在する場合は，ブロック有効範囲の名前が見えて，ファイル有効範囲の名前は隠される．

重要：ブロック有効範囲をもつ同じ名前の変数が存在する場合，より内側のものが見えて，より外側のものが隠される．

□□□□□□□□□□□□□□

```c
// □□□□□□□□□□□□□□□□
#include <stdio.h>

int x = 75; // □□□□□□□□□

void print_x(void)
{
    printf("x = %d\n", x);
}

int main(void)
{
    int x = 999; // □□□□□□□□□

    print_x();

    printf("x = %d\n", x);

    for (int i = 0; i < 5; i++)
    {
        int x = i * 100;
        printf("x = %d\n", x);
    }
    printf("x = %d\n", x);
}
```

課題について

第2回授業の時には「授業課題」と「宿題」の2種類の課題を提示しました.

今回からは，「課題」のみとします.

「課題」の扱いについて

- 提出期限は翌週月曜日**20**時までとします
- 基本的には授業時間内に取り組んでください
- 11:30以降は自由解散とします

課題

- Moodleに課題ファイル（Wordファイル）があります
- 課題ファイルに書かれている問題に回答するプログラムを作成してください
- 作成したプログラムを実行して問題なく動作しているかを確認してください
- 動作確認が終わったら，プログラム（ソースコード）をWordファイルに貼り付けてください
- すべての問題ができたらWordファイルをMoodleに提出してください

提出期限は **10月7日(月) 20:00** まで