

# プログラミング演習I

## 関数

# 関数とは？

```
#include <stdio.h>

int main(void)
{
    /* ... 中略 ... */

    return 0;
}
```

● **Fig.6-1** おまじないと main 関数

今まで、上記のすべての部分は、いわゆる“おまじない”で、具体的な説明には踏み込まなかった。

まず、赤色で示した部分について説明

main関数＝>関数とは、複数の入力があり、関数を評価した結果が、戻り値として帰る

# 数学での関数

$$f(x, y) = x^2 + y$$

$$f(5, 3) \text{は？}$$

- まず関数を定義
  - 入力は2つ
  - 入力の1つめを2乗して、2つめと加算した結果が $f(5, 3)$ の値となる。

# ライブラリ関数

- 今まで、利用してきた関数
  - printf()
  - scanf()
  - puts()
  - etc....
- これら、C言語が提供する関数のことをライブラリ関数と呼ぶ。

```
/*
 * 二つの整数の大きいほうの値を求める
 */

#include <stdio.h>

/*--- 大きいほうの値を返す ---*/
int max2(int a, int b)
{
    if (a > b)
        return a;
    else
        return b;
}

int main(void)
{
    int n1, n2;

    puts("二つの整数を入力してください。");
    printf("整数1 : ");    scanf("%d", &n1);
    printf("整数2 : ");    scanf("%d", &n2);

    printf("大きいほうの値は%dです。\\n", max2(n1, n2));

    return 0;
}
```

## 実行例 1

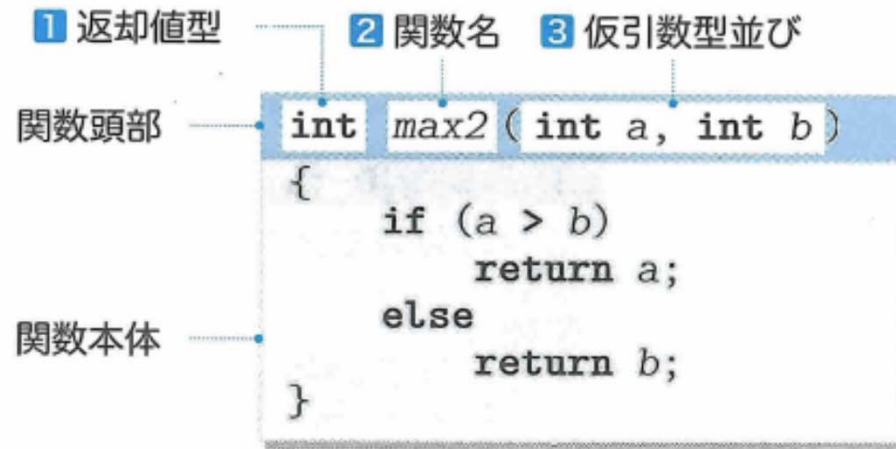
二つの整数を入力してください。  
整数1 : 45   
整数2 : 83   
大きいほうの値は83です。

## 実行例 2

このプログラムは、max2()、main()の2つの関数からなっている。プログラムは、まずmain関数から実行されるので、2つの数値を読み込み、関数max2を呼び出す。関数max2では、大きい値を関数の評価値として返す。

# 関数定義と呼び出し<sup>5</sup>

# max2関数を詳しく見てみると



関数はプログラムの部品 !!

- ・ 関数の名前は max2。
- ・ int 型の仮引数 a, b を受け取る。
- ・ 大きいほうの値を求める。
- ・ 求めた int 型の値を呼出し元に返却する。

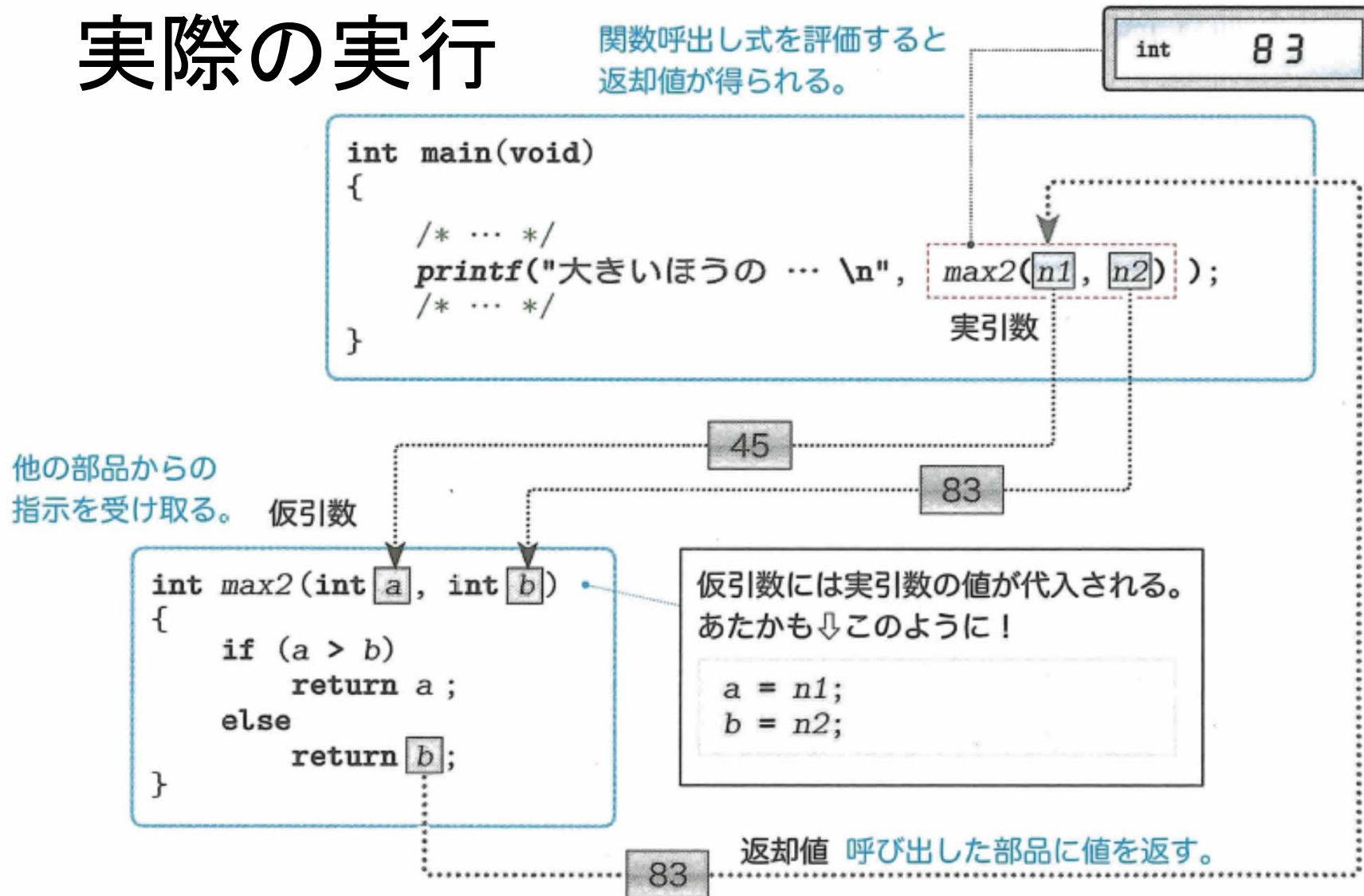
● Fig.6-3 関数定義の構造

関数maxofは2つの入力(仮引数)を持ち、1つめの仮引数の型は整数型、関数内での仮の名前をxとする。2つめの仮引数の型も整数型で、関数内での仮の名前をyとする。また、関数を評価した結果は、整数型となる。

実際に関数を呼ぶ方(今回はmain関数)での引数を実引数と呼ぶ。

# 実際の実行

関数呼出し式を評価すると  
返却値が得られる。



return文を実行することにより、関数が終わり、呼び出し側に制御が戻る。その際return文の引数が関数の戻り値となる。

# max2関数のいろいろな書き方

a

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

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

    return max;
}
```

b

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

    if (b > max)
        max = b;

    return max;
}
```

c

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

条件演算子?:はp.56で学習済み!!

● Fig.6-6 関数 max2 の実現例

プログラムを綺麗、読みやすく書くためには、できる限り開始位置は1つ(これは関数では最初から実行されるので当たり前)、終了位置も1つのほうがよい。=>読みやすい



```
/*
 三つの整数の最大値を求める
*/

#include <stdio.h>

/*--- 三つの整数の最大値を返す ---*/
int max3(int a, int b, int c)
{
    int max = a;

    if (b > max) max = b;
    if (c > max) max = c;
    return max;
}

int main(void)
{
    int a, b, c;

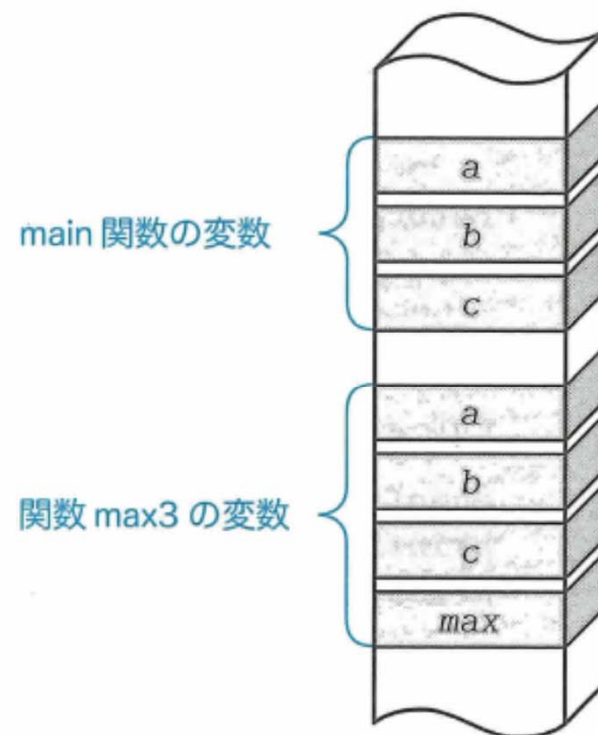
    puts("三つの整数を入力してください。");
    printf("整数a: ");    scanf("%d", &a);
    printf("整数b: ");    scanf("%d", &b);
    printf("整数c: ");    scanf("%d", &c);

    printf("最大値は%dです。 \n", max3(a, b, c));

    return 0;
}
```

## 実行例

三つの整数を入力してください。  
整数a: 5  
整数b: 3  
整数c: 4  
最大値は5です。



● Fig.6-7 二つの関数と変数 9

```
/*
 二つの整数の2乗値の差を求める
*/
```

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

```
/*--- xの2乗値を返す ---*/
```

```
int sqr(int x)
```

```
{
    return x * x;
}
```

```
/*--- xとyの差を返す ---*/
```

```
int diff(int a, int b)
```

```
{
    return (a > b) ? a - b : b - a;    /* 大きいほうから小さいほうを引く */
}
```

```
int main(void)
```

```
{
    int x, y;

    puts("二つの整数を入力してください。");
    printf("整数x: ");    scanf("%d", &x);
    printf("整数y: ");    scanf("%d", &y);
```

```
    printf("xの2乗とyの2乗の差は%dです。 \n", diff(sqr(x), sqr(y)));
```

```
    return 0;
```

```
}
```

### 実行例

二つの整数を入力してください。

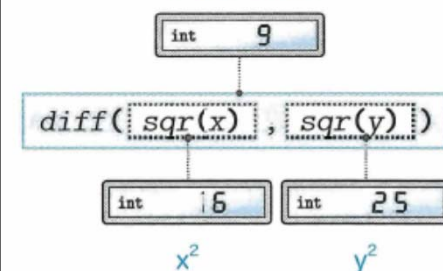
整数x: 4

整数y: 5

xの2乗とyの2乗の差は9です。

差の求め方は List 3-15 (p.57) で学習済み !!

$x^2$  と  $y^2$  の差



```

/*
  左下直角の直角二等辺三角形を表示（関数版）
*/

#include <stdio.h>

/*--- 記号文字'*'をn個連続して表示 ---*/
void put_stars(int n)
{
    while (n-- > 0)
        putchar('*');
}

int main(void)
{
    int i, len;

    printf("左下直角二等辺三角形を作ります。 \n");
    printf("短辺: ");
    scanf("%d", &len);

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

    return 0;
}

```

カウントダウンの制御式は  
前ページの List 6-6 と同じ。

put\_stars()は戻り値のない  
(void)関数。

### 実行例

左下直角二等辺三角形  
を作ります。

短辺: 5

```

*
**
***
****
*****

```

```

/*--- 参考: List 4-18 (p.99) ---*/
for (i = 1; i <= len; i++) {
    for (j = 1; j <= i; j++)
        putchar('*');
    putchar('\n');
}

```

```
/*
 右下直角の直角二等辺三角形を表示 (関数版)
*/

#include <stdio.h>

/*--- 文字chをn個連続して表示 ---*/
void put_chars(int ch, int n)
{
    while (n-- > 0)
        putchar(ch);
}

int main(void)
{
    int i, len;


    printf("右下直角二等辺三角形を作ります。 \n");
    printf("短辺: ");
    scanf("%d", &len);

    for (i = 1; i <= len; i++) {
        put_chars(' ', len - i);
        put_chars('*', i);
        putchar('\n');
    }

    return 0;
}
```

## 実行例

右下直角二等辺三角形  
を作ります。

短辺: 5 

```
 *
 **
 ***
 ****
 *****
```

```
/*--- 参考: List 4-19 (p.99) ---*/
for (i = 1; i <= len; i++) {
    for (j = 1; j <= len - i; j++)
        putchar(' ');
    for (j = 1; j <= i; j++)
        putchar('*');
    putchar('\n');
}
```



```
/*
 * 読み込んだ正の整数値を逆順に表示
 */
```

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

この関数は引数をとらない

```
/*--- 正の整数を読み込んで返す ---*/
```

```
int scan_pint(void)
```

```
{
    int tmp;
    // 引数を受け取らない。
```

```
    do {
```

```
        printf("正の整数を入力してください：");
```

```
        scanf("%d", &tmp);
```

```
        if (tmp <= 0)
```

```
            puts("\a正でない数を入力しないでください。");
```

```
    } while (tmp <= 0);
```

```
    return tmp;
}
```

```
/*--- 非負の整数を反転した値を返す ---*/
```

```
int rev_int(int num)
```

```
{
```

```
    int tmp = 0;
```

```
    if (num > 0) {
```

```
        do {
```

```
            tmp = tmp * 10 + num % 10;
```

```
            num /= 10;
```

```
        } while (num > 0);
```

```
    }
```

```
    return tmp;
}
```

### 実行例

正の整数を入力してください：-5

正でない数を入力しないでください。

正の整数を入力してください：128

反転した値は821です。

```
/*--- 非負の整数を反転した値を返す ---*/
```

```
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("反転した値は%dです。 \n", rev_int(nx));
```

```
    return 0;
```

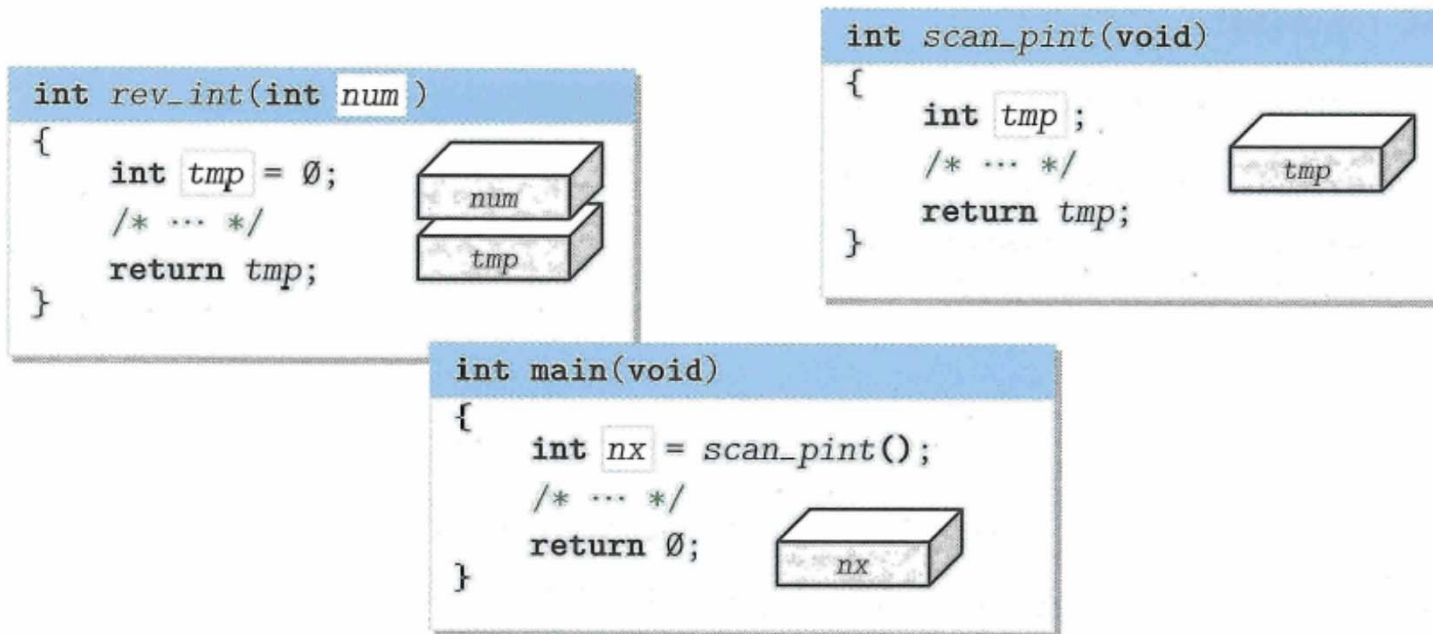
```
}
```

scan\_pint()は引数をとらない関数

実引数を与えない。

# 有効範囲(スコープ)

- `scan_pint()`と`rev_int()`には、同じ識別子(名前)`tmp`がある＝>同じもの？
- 識別子(変数名、関数名などプログラムで使うすべての名前)はどこまで有効か？



● Fig.6-10 関数内で宣言されたオブジェクト

- ブロック有効範囲(ブロックスコープ)
  - ブロック内 “{このなか}”で宣言された変数は、ブロック内でのみ通用する。
- ファイル有効範囲(ファイルスコープ)
  - そのソースファイル内で有効な宣言

```
// ファイル・スコープの始まり
void f(int x); // プロトタイプ・スコープ: ) まで
double y;

int main(void)
{ // ブロック・スコープ1の始まり
    int n;
    { // ブロック・スコープ2の始まり
        char c;
        double y;
    } // ブロック・スコープ2の終わり
} // ファイル・スコープとブロック・スコープ1の終わり
```



```

/*
  最高点を求める
*/

#include <stdio.h>

#define NUMBER 5  /* 学生の人数 */

int tensu[NUMBER]; /* 配列の定義 */ 1

int top(void); /* 関数topの関数原型宣言 */ 2

int main(void)
{
    extern int tensu[]; /* 配列の宣言 (省略可) */ 3
    int i;

    printf("%d人の点数を入力してください。 \n", NUMBER);
    for (i = 0; i < NUMBER; i++) {
        printf("%d: ", i + 1);
        scanf("%d", &tensu[i]);
    }
    printf("最高点=%d\n", top());

    return 0;

}

/*--- 配列tensuの最大値を返す関数topの関数定義 ---*/
int top(void)
{
    extern int tensu[]; /* 配列の宣言 (省略可) */ 4
    int i;
    int max = tensu[0];

    for (i = 1; i < NUMBER; i++)

```

配列の実体を作るための宣言 (定義)

別の場所で作られた配列を使うための宣言 (定義ではない)

## 実行例

5人の点数を入力  
してください。

1: 53

2: 49

3: 21

4: 91

5: 77

最高点=91

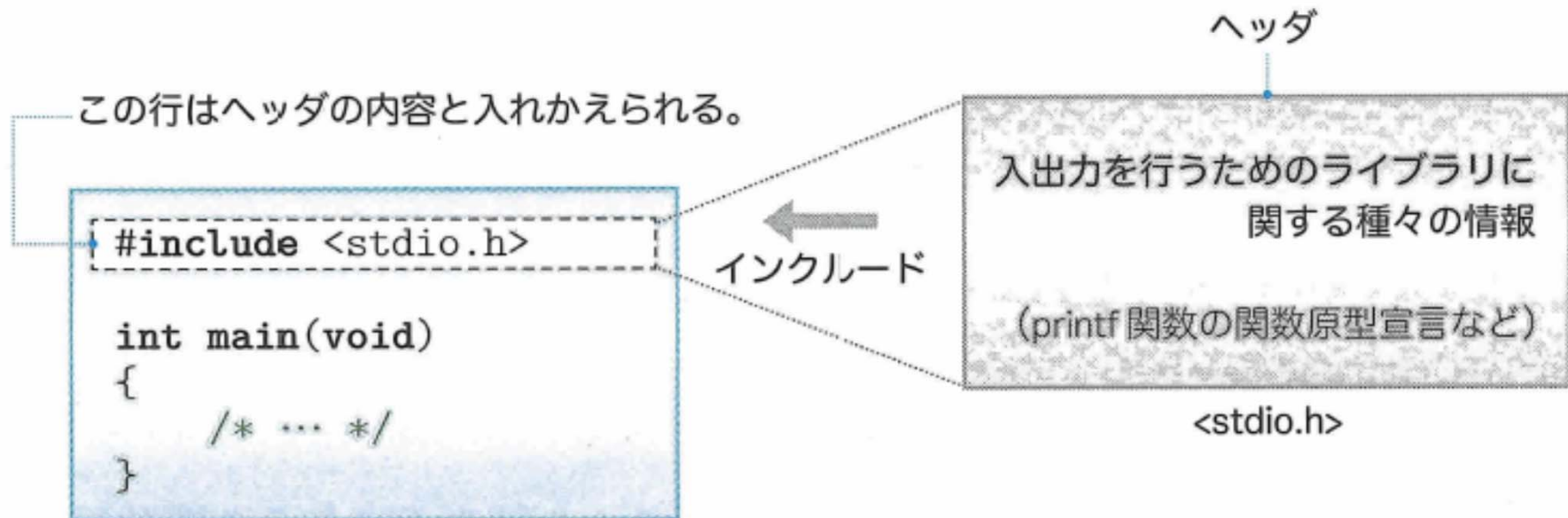
```
/*--- 配列tensuの最大値を返す関数topの関数定義 ---*/  
int top(void)  
{  
    extern int tensu[]; /* 配列の宣言（省略可） */  
    int i;  
    int max = tensu[0];  
  
    for (i = 1; i < NUMBER; i++)  
        if (tensu[i] > max)  
            max = tensu[i];  
    return max;  
}
```

- extern宣言は、どこかで宣言されている変数を使いますという意味。この場合は、ファイルスコープとして宣言されているものを使う  
(ただし、この例の場合は、ファイルスコープとして宣言された配列を使うのでこの定義自体が必要ない)

# 関数原型宣言

- プログラム6－10において、top()がプログラムされる前に、main()で呼んでいる。
- この場合、コンパイラは、top()が、何型の引数を何個取り、かつ戻り値は何型であるかが全くわからないと困る。
- そこで、関数原型宣言(プロトタイプ宣言)を、前もって行う必要がある。
- むろん、実際に関数を使用する前に、すべての関数が定義されているのであれば、プロトタイプ宣言は必要なし。

# 最後のおまじない #include



● Fig.6-11 ヘッダのインクルード

#include < > の場合は、あらかじめ決められたディレクトリから読み込む。

#include " " の場合は、現在のディレクトリから読み込む。

## /usr/include/math.hの一部

```
extern double acos __((double ));
extern double asin __((double ));
extern double atan __l((double ));
extern double atan2 __l((double ,double ));
extern double ceil __l((double ));
extern double cos __l((double ));
extern double cosh __((double ));
extern double exp __((double ));
extern double fabs __l((double ));
extern double floor __l((double ));
extern double fmod __((double , double ));
extern double frexp __((double , int *));
extern double ldexp __((double , int ));
extern double log __((double ));
#define M_E      2.7182818284590452354E0 /*Hex 2^0 * 1.5bf0a8b145769 */
#define M_LOG2E  1.4426950408889634074E0 /*Hex 2^0 * 1.71547652B82FE */
#define M_LOG10E 4.3429448190325182765E-1 /*Hex 2^-2 * 1.BCB7B1526E50E */
#define M_LN2    6.9314718055994530942E-1 /*Hex 2^-1 * 1.62E42FEFA39EF */
#define M_LN10   2.3025850929940456840E0 /*Hex 2^1 * 1.26bb1bbb55516 */
#define M_PI     3.1415926535897932385E0 /*Hex 2^1 * 1.921FB54442D18 */
```

# 関数への配列の渡し方

- 通常、関数への引数は、“値渡し(**Call by value**)”である。
- しかし、大量のデータを有する配列をデータごと渡すのは非効率である
- また、関数では、関数を評価した結果の値1つしか返せない。
- そこで、配列が格納されている“場所(アドレス)”を引数として渡す(参照渡し、**Call by reference**)。



```

*/
#include <stdio.h>

#define NUMBER 5      /* 学生の人数 */

/*--- 要素数nの配列vの最大値を返す ---*/
int max_of(int v[], int n)
{
    int i;
    int max = v[0];

    for (i = 1; i < n; i++)
        if (v[i] > max)
            max = v[i];
    return max;
}

int main(void)
{
    int i;
    int eng[NUMBER];      /* 英語の点数 */
    int mat[NUMBER];      /* 数学の点数 */
    int max_e, max_m;      /* 最高点 */

    printf("%d人の点数を入力してください。 \n", NUMBER);
    for (i = 0; i < NUMBER; i++) {
        printf("[%d] 英語: ", i + 1);  scanf("%d", &eng[i]);
        printf("      数学: ");        scanf("%d", &mat[i]);
    }
    max_e = max_of(eng, NUMBER);  /* 英語の最高点 */
    max_m = max_of(mat, NUMBER);  /* 数学の最高点 */

    printf("英語の最高点=%d\n", max_e);
    printf("数学の最高点=%d\n", max_m);
}

```

配列を受け取る仮引数の  
宣言には[]を付ける。

main関数は、配列eng(もしくはmat)の位置  
(アドレス)を渡して、max\_of関数は、その渡  
された配列をvという名前の配列でアクセス  
する。

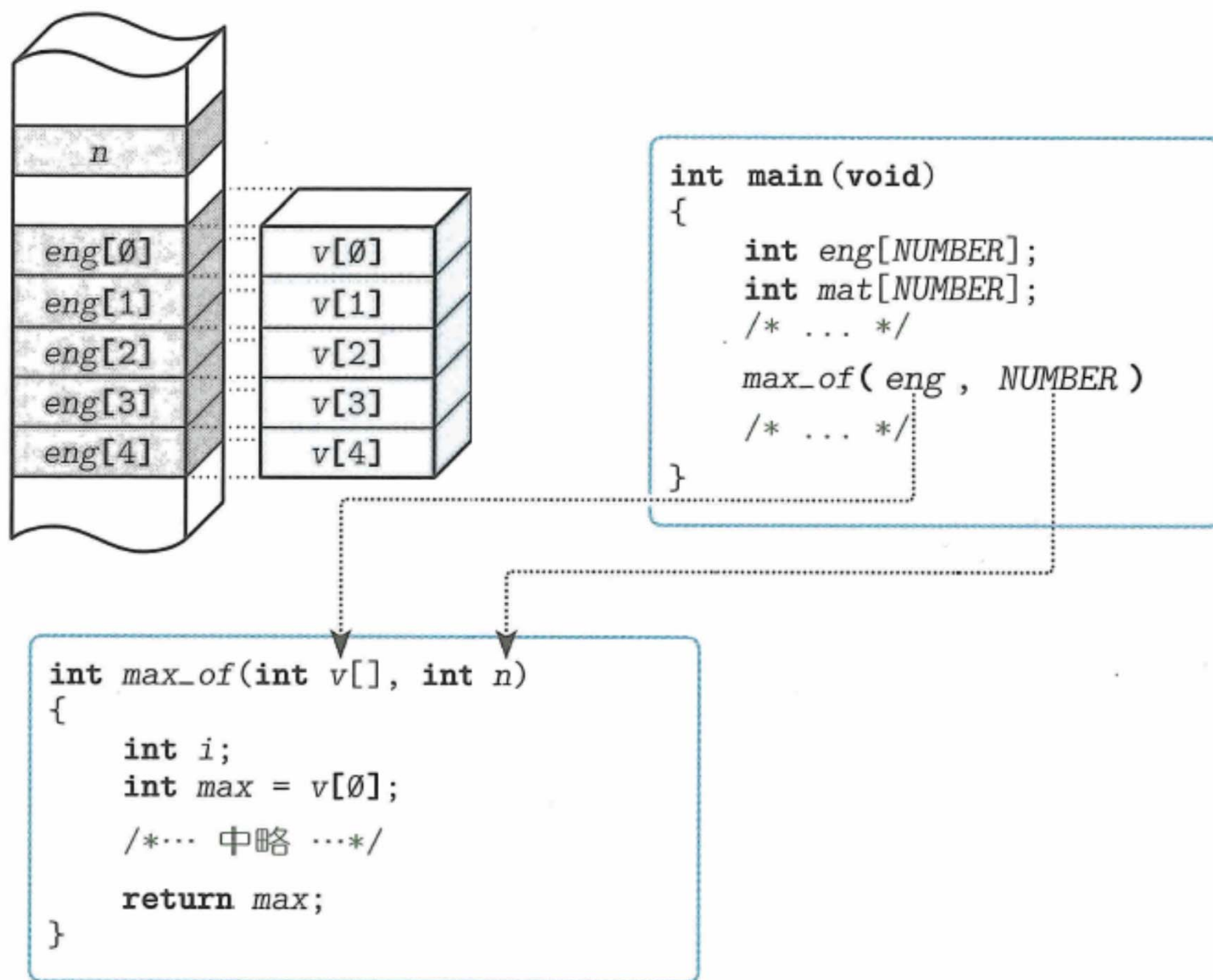
## 実行例

5人の点数を入力してください。

[1] 英語: 53   
 数学: 82   
 [2] 英語: 49   
 数学: 35   
 [3] 英語: 21   
 数学: 72   
 [4] 英語: 91   
 数学: 35   
 [5] 英語: 77   
 数学: 12

LIST6-11

呼出し側では、[]を付けずに  
配列の名前だけを記述する。23



● Fig.6-12 関数呼出しにおける配列の受渡し



# 受け取った配列への書き込み

- 仮引数として受け取った配列（実際には配列への参照（アドレス））は、配列そのもの、従って、関数側で配列を変更した結果は、呼び出し側のプログラムでも有効である。
  - 通常関数は、値渡しなので、関数内の変更は、その関数内でのみ有効

```

/*
  配列の全要素をゼロにする
*/

#include <stdio.h>

/*--- 要素数nの配列vの要素に0を代入 ---*/
void set_zero(int v[], int n)
{
    int i;

    for (i = 0; i < n; i++)
        v[i] = 0;
}

/*--- 要素数nの配列vの全要素を表示して改行 ---*/
void print_array(const int v[], int n)
{
    int i;

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

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

    printf("ary1 = "); print_array(ary1, 5); putchar('\n');
    printf("ary2 = "); print_array(ary2, 3); putchar('\n');

    set_zero(ary1, 5); /* 配列ary1の全要素に0を代入 */
    set_zero(ary2, 3); /* 配列ary2の全要素に0を代入 */

    printf("両配列の全要素に0を代入しました。 \n");
    printf("ary1 = "); print_array(ary1, 5); putchar('\n');
    printf("ary2 = "); print_array(ary2, 3); putchar('\n');

    return 0;
}

```

## 実行結果

```

ary1 = { 1 2 3 4 5 }
ary2 = { 3 2 1 }
両配列の全要素に0を代入しました。
ary1 = { 0 0 0 0 0 }
ary2 = { 0 0 0 }

```

受け取る配列の要素の値を  
変更しないことを宣言する。

# const演算子

- 渡す配列の内容を書き換えられては困るときに使う。READ ONLYになる。

```
/*--- 要素数nの配列vの全要素を表示して改行 ---*/
```

```
void print_array(const int v[], int n)
```

```
{
```

```
    int i;
```

```
    printf("{ ");
```

```
    for (i = 0; i < n; i++)
```

```
        printf("%d ", v[i]);
```

```
    printf("}");
```

```
}
```

受け取る配列の要素の値を  
変更しないことを宣言する。

もし、`v[1]=1`のように代入文があったときは、コンパイル時にエラーが起きる。

# 簡単なアルゴリズム

## その1 逐次探索

- アルゴリズム(算法)とは、いろいろな問題をプログラム化する方法論
- アルゴリズムをちゃんと考えないと、美しいプログラムはできない(美しい=>綺麗、早い、保守性が良い)
- 逐次探索とは
  - 配列の中に、目的とする値(今回は整数であるが、文字列なども同様)が存在するかどうかを見つけること

**a** 49を探索（探索成功）

①	1	2	3	4
83	55	77	49	25
0	①	2	3	4
83	55	77	49	25
0	1	②	3	4
83	55	77	49	25
0	1	2	③	4
83	55	77	49	25

探索成功！

探索すべき値と等しい要素を発見。

**b** 16を探索（探索失敗）

①	1	2	3	4
83	55	77	49	25
0	①	2	3	4
83	55	77	49	25
0	1	②	3	4
83	55	77	49	25
0	1	2	③	4
83	55	77	49	25
0	1	2	3	④
83	55	77	49	25
0	1	2	3	4
83	55	77	49	25

探索失敗！

配列の末端を通り越してしまった。

● Fig.6-13 逐次探索

```

/*
 線形探索 (逐次探索)
*/

#include <stdio.h>

#define NUMBER      5      /* 要素数 */
#define FAILED      -1     /* 探索失敗 */

/*--- 要素数nの配列vからkeyと一致する要素を探索 ---*/
int search(const int v[], int key, int n)
{
    int i = 0;

    while (1) {
        if (i == n)
            return FAILED;      /* 探索失敗 */
        if (v[i] == key)
            return i;           /* 探索成功 */
        i++;
    }
}

```

## 実行例 1

```

vx[0] : 83
vx[1] : 55
vx[2] : 77
vx[3] : 49
vx[4] : 25
探す値 : 49
49は4番目にあります。

```

## 実行例 2

```

vx[0] : 83
vx[1] : 55
vx[2] : 77
vx[3] : 49
vx[4] : 25
探す値 : 16
探索に失敗しました。

```

```
int main(void)
{
    int i, ky, idx;
    int vx[NUMBER];

    for (i = 0; i < NUMBER; i++) {
        printf("vx[%d] : ", i);
        scanf("%d", &vx[i]);
    }
    printf("探す値 : ");
    scanf("%d", &ky);

    idx = search(vx, ky, NUMBER);    /* 要素数NUMBERの配列vxからkyを探索 */

    if (idx == FAILED)
        puts("\a探索に失敗しました。");
    else
        printf("%dは%d番目にあります。\\n", ky, idx + 1);

    return 0;
}
```

# もう少し綺麗にならないか？

- 繰り返しのたびに、2つの終了条件をチェックするのは無駄。

```
int search(const int v[], int key, int n)
{
    int i = 0;
    while (1) {
        if (i == n)
            return FAILED;          /* 探索失敗 */
        if (v[i] == key)
            return i;               /* 探索成功 */
        i++;
    }
}
```

while文のなかで、(1)目的とする数字を見つけるIF文と、(2)配列の終わりを見つけるIF文、の2つが存在している=>計算時間の無駄



a 49を探索（探索成功）

①	1	2	3	4
83	55	77	49	25
0	①	2	3	4
83	55	77	49	25
0	1	②	3	4
83	55	77	49	25
0	1	2	③	4
83	55	77	49	25

探索成功！  
探索すべき値と等しい要素を発見。

b 16を探索（探索失敗）

①	1	2	3	4
83	55	77	49	25
0	①	2	3	4
83	55	77	49	25
0	1	②	3	4
83	55	77	49	25
0	1	2	③	4
83	55	77	49	25
0	1	2	3	④
83	55	77	49	25
0	1	2	3	4
83	55	77	49	25

探索失敗！  
配列の末端を通り越してしまった。

● Fig.6-13 逐次探索

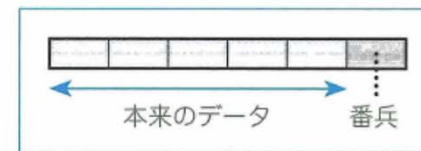
配列の最後に、探索対象の数値を、事前に挿入しておく=>番兵(sentinel)法

a 49を探索（探索成功）

0	1	2	③	4	5
83	55	77	49	25	49

要素数 n より小さい値。

探索する値と等しい要素を発見。



b 16を探索（探索失敗）

0	1	2	3	4	⑤
83	55	77	49	25	16

要素数 n と同じ値。

探索する値と等しい要素を発見。  
※ただし見つけたのは番兵。

● Fig.6-14 逐次探索(番兵法)

```
/*
 逐次探索（番兵法）
*/

#include <stdio.h>

#define NUMBER      5      /* 要素数 */
#define FAILED      -1     /* 探索失敗 */

/*--- 要素数nの配列vからkeyと一致する要素を探索（番兵法） ---*/
int search(int v[], int key, int n)
{
    int i = 0;

    v[n] = key;      /* 番兵を格納 */

    while (1) {
        if (v[i] == key)
            break;    /* 探索成功 */
        i++;
    }
    return (i < n) ? 0 : FAILED;
}
```

## 実行例 1

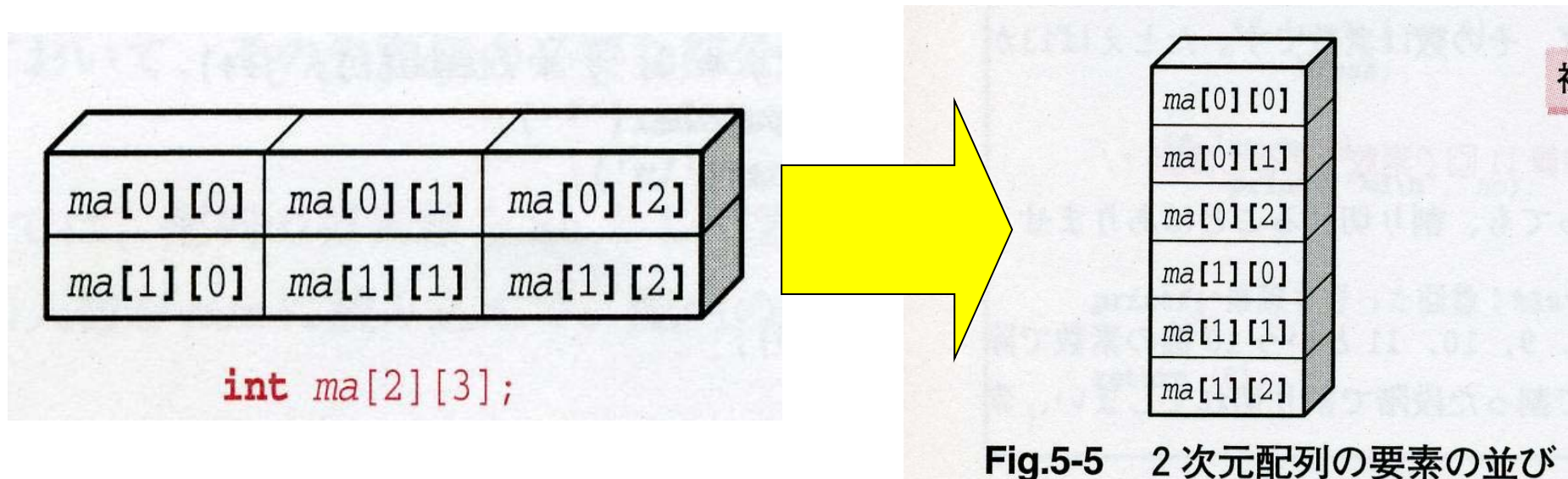
vx[0] : 83 ☐  
vx[1] : 55 ☐  
vx[2] : 77 ☐  
vx[3] : 49 ☐  
vx[4] : 25 ☐  
探す値 : 49 ☐  
49は4番目にあります。

## 実行例 2

Loop内のIF文は1つになった。  
v(^0^)/

# 多次元配列の引き渡し

- 計算機のアドレスは、1次元であり、基本的に多次元配列も1次元空間で表される。
- 従って、多次元配列を引数とするときも、開始番地を教えるだけでOK
- ただし、何次元配列で、それぞれの要素数は？という情報は必要である。



```

/*
  4人の学生の3科目のテスト2回分の合計を求めて表示（関数版）
*/

#include <stdio.h>

/*--- 4行3列の行列aとbの和をcに格納する ---*/
void mat_add(const int a[4][3], const int b[4][3], int c[4][3])
{
    int i, j;
    for (i = 0; i < 4; i++)
        for (j = 0; j < 3; j++)
            c[i][j] = a[i][j] + b[i][j];
}

/*--- 4行3列の行列mを表示 ---*/
void mat_print(const int m[4][3])
{
    int i, j;
    for (i = 0; i < 4; i++) {
        for (j = 0; j < 3; j++)
            printf("%4d", m[i][j]);
        putchar('\n');
    }
}

```

## 実行結果

1 回目の点数

91 63 78

67 72 46

89 34 53

73 43 46

97 56 21

85 46 35

合計点

188 130 160

140 115 92

186 90 74

117 100 69

void mat\_add(const int a[][3], const int b[][3] ..  
のように最初の要素数だけは、省略可能



# 有効範囲(スコープ)と記憶域期間

List 6-17

chap06/list0617.c

```
/*
 識別子の有効範囲を確認する
*/

#include <stdio.h>

int x = 75; /* A : ファイル有効範囲 */

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

int main(void)
{
    int i;
    int x = 999; /* B : ブロック有効範囲 */

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

    for (i = 0; i < 5; i++) {
        int x = i * 100; /* C : ブロック有効範囲 */
        printf("x = %d\n", x);
    }

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

    return 0;
}
```

## 実行結果

```
x = 75
x = 999
x = 0
x = 100
x = 200
x = 300
x = 400
x = 999
```

# 記憶域期間

- 自動記憶域期間(Automatic storage duration)
  - プログラムの流れが、宣言を通過する際に、生成される。宣言を含むブロックの終点(すなわち“}”)を通過する際に、その実態は消滅する。
- 静的記憶域期間(Static storage duration)
  - プログラムの開始時に、生成され、プログラムの終了時に消滅する。
  - static を付けて宣言する。

```

/*
 自動記憶域期間と静的記憶域期間
*/

#include <stdio.h>

int fx = 0; /* 静的記憶域期間+ファイル有効範囲 */

void func(void)
{
    static int sx = 0; /* 静的記憶域期間+ブロック有効範囲 */
    int ax = 0; /* 自動記憶域期間+ブロック有効範囲 */

    printf("%3d%3d%3d\n", ax++, sx++, fx++);
}

int main(void)
{
    int i;

    puts(" ax sx fx");
    puts("-----");
    for (i = 0; i < 10; i++)
        func();
    puts("-----");

    return 0;
}

```

static属性のfx,sx 生成

### 実行結果

ax	sx	fx
0	0	0
0	1	1
0	2	2
0	3	3
0	4	4
0	5	5
0	6	6
0	7	7
0	8	8
0	9	9

```

/*
 自動記憶域期間と静的記憶域期間
*/

#include <stdio.h>

int fx = 0;          /* 静的記憶域期間+ファイル有効範囲 */

void func(void)
{
    static int sx = 0; /* 静的記憶域期間+ブロック有効範囲 */
    int      ax = 0;   /* 自動記憶域期間+ブロック有効範囲 */

    printf("%3d%3d%3d\n", ax++, sx++, fx++);
}

int main(void)
{
    int i;

    puts(" ax sx fx");
    puts("-----");
    for (i = 0; i < 10; i++)
        func();
    puts("-----");

    return 0;
}

```

auto属性のi 生成(初期値は不定)

### 実行結果

ax	sx	fx
0	0	0
0	1	1
0	2	2
0	3	3
0	4	4
0	5	5
0	6	6
0	7	7
0	8	8
0	9	9



```

/*
 自動記憶域期間と静的記憶域期間
*/

#include <stdio.h>

int fx = 0; /* 静的記憶域期間＋ファイル有効範囲 */

void func(void)
{
    static int sx = 0; /* 静的記憶域期間＋ブロック有効範囲 */
    int ax = 0; /* 自動記憶域期間＋ブロック有効範囲 */

    printf("%3d%3d%3d\n", ax++, sx++, fx++);
}

int main(void)
{
    int i;

    puts(" ax sx fx");
    puts("-----");
    for (i = 0; i < 10; i++)
        func();
    puts("-----");

    return 0;
}

```

auto属性のax生成＋初期値0

0 0 0と表示

関数の終了とともにaxは消滅

### 実行結果

ax	sx	fx
0	0	0
0	1	1
0	2	2
0	3	3
0	4	4
0	5	5
0	6	6
0	7	7
0	8	8
0	9	9

```

/*
 自動記憶域期間と静的記憶域期間
*/

#include <stdio.h>

int fx = 0;          /* 静的記憶域期間＋ファイル有効範囲 */

void func(void)
{
    static int sx = 0; /* 静的記憶域期間＋ブロック有効範囲 */
    int ax = 0;        /* 自動記憶域期間＋ブロック有効範囲 */

    printf("%3d%3d%3d\n", ax++, sx++, fx++);
}

int main(void)
{
    int i;

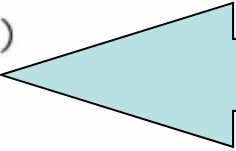
    puts(" ax sx fx");
    puts("-----");
    for (i = 0; i < 10; i++)
        func();
    puts("-----");

    return 0;
}

```

## 実行結果

ax	sx	fx
0	0	0
0	1	1
0	2	2
0	3	3
0	4	4
0	5	5
0	6	6
0	7	7
0	8	8
0	9	9


 iを1増加させてfunc()を呼ぶ

```

/*
 自動記憶域期間と静的記憶域期間
*/

#include <stdio.h>

int fx = 0; /* 静的記憶域期間+ファイル有効範囲 */

void func(void)
{
    static int sx = 0; /* 静的記憶域期間+ブロック有効範囲 */
    int ax = 0; /* 自動記憶域期間+ブロック有効範囲 */

    printf("%3d%3d%3d\n", ax++, sx++, fx++);
}

int main(void)
{
    int i;

    puts(" ax sx fx");
    puts("-----");
    for (i = 0; i < 10; i++)
        func();
    puts("-----");

    return 0;
}

```

auto属性のax 生成

0 1 1と表示

### 実行結果

ax	sx	fx
0	0	0
0	1	1
0	2	2
0	3	3
0	4	4
0	5	5
0	6	6
0	7	7
0	8	8
0	9	9