# 第５章　関数の基本

## 5-1　関数を作ろう

### ■ キャスト

int 型の変数 a, b が、それぞれ 3, 4 という値をもつとすします。

```
(a + b) / 2          →  7 / 2     →  3      （小数点以下は切り捨て）
(a + b) / 2.0        →  7 / 2.0   →  3.5
double(a + b) / 2    →  7.0 / 2   →  3.5
```

### ■ 三角形の面積を求める

三角形の面積を求めます。底辺・高さは int 型です。

```
/*
    三角形の面積を求める
*/

#include  <iostream.h>

int main(void)
{
    int     height;     /* 高さ */
    int     width;      /* 底辺の長さ */
    double  area;       /* 面積 */

    cout << "三角形の面積を求めます。¥n";

    cout << "底辺："; cin >> width;
    cout << "高さ："; cin >> height;

    area = width * height / 2;       /* 面積＝底辺×高さ／２ */

    cout << "面積は" << area << "です。¥n";

    return (0);
}
```

```
底辺：3
高さ：5
面積は7です。
```

たとえ左辺が double 型でも、int * int / int は所詮 int です。

double 型の面積の中身が整数値となってしまうことに注意しましょう。

## ■ 三角形の面積を求める（解決法１）

```
/*
    三角形の面積を求める（解決法１）
*/

#include  <iostream.h>

int main(void)
{
    int    height;    /* 高さ */
    int    width;     /* 底辺の長さ */
    double  area;     /* 面積 */

    cout << "三角形の面積を求めます。¥n";

    cout << "底辺："; cin >> width;
    cout << "高さ："; cin >> height;

    area = width * height / 2.0;    /* 面積＝底辺×高さ／2.0 */

    cout << "面積は" << area << "です。¥n";

    return (0);
}
```

```
底辺：3
高さ：5
面積は7.5です。
```

右辺の計算は、
int * int / double
ですから、double 型となります。

2.0 で割るとうまくいきます。

## ■ 三角形の面積を求める（解決法２）

```
/*
    三角形の面積を求める（解決法２）
*/

#include  <iostream.h>

int main(void)
{
    /* 途中省略*/

    area = (double)(width * height) / 2;        /* 面積＝底辺×高さ／2.0 */

    cout << "面積は" << area << "です。¥n";

    return (0);
}
```

```
底辺：3
高さ：5
面積は7.5です。
```

右辺の計算は、
(double)(int *int)/int
すなわち double / int
となり、double 型となります。

乗算(底辺 * 高さ)によって得られた値を明示的に double 型にキャスト（型変換）します。

## ■ 実引数と仮引数

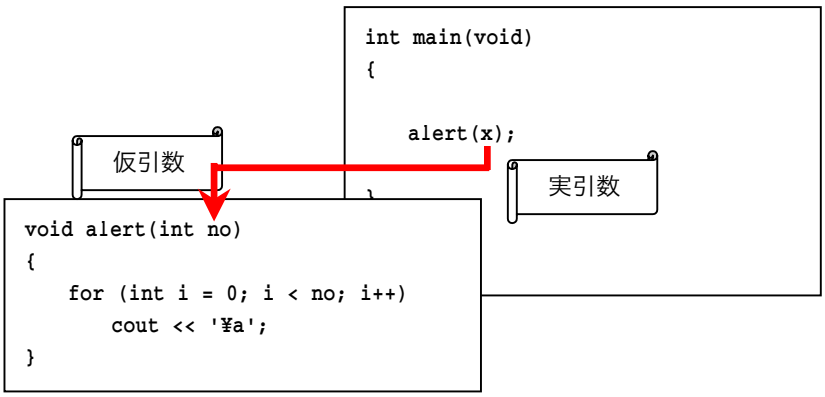
関数間で、何らかの情報をやりとりするために、用いるのが引数である。

呼び出す側が渡す引数が**実引数**（*argument*）、呼び出される側が受け取る引数が**仮引数**（*parameter*）です。

## ■ 指定された回数警報をならす関数

```
/*
    指定された回数警報をならす関数
*/

#include  <iostream.h>

/*--- no回警報をならす ---*/
void alert(int no)
{
    for (int i = 0; i < no; i++)
        cout << '¥a';
}

int main(void)
{
    int  x;

    cout << "何回警報をならしますか：";
    cin >> x;

    alert(x);

    return (0);
}
```

何回警報をならしますか：5



## ■ 値渡し

受け取った仮引数は、実引数のコピーなので、勝手に値を変更しても構いません（本のコピーをとって、そのコピーに鉛筆やペンで書き込みを行っても、もとの本にはまったく無関係であるのと同じです)。

### ■ 指定された回数警報をならす関数

```
/*
    指定された回数警報をならす関数
*/

#include  <iostream.h>

/*--- no回警報をならす ---*/
void alert(int no)
{
    while (no > 0) {
        cout << '¥a';
        no--;
    }
}

int main(void)
{
    int  x;

    cout << "何回警報をならしますか：";
    cin >> x;

    alert(x);

    cout << x << "回警報をならしました。¥n";

    return (0);
}
```

何回警報をならしますか：5
🔊 🔊 🔊 🔊 🔊
5回警報をならしました。

仮引数noの値は0になってしまう。

実引数xの値は変化していない。

## ■ 関数は部品である

ある意味で、関数はプログラムの部品（パーツ）です。日曜大工でもそうですが、すべてを一から組み立てるよりも、うまく部品を利用したほうが、スムーズに進みます。

長方形を表示するプログラムで比較してみましょう。

### ■ 長方形を表示する（関数を使わない）

```
/*
    長方形を表示する
*/

#include <iostream.h>

int main(void)
{
    int  height;       // 高さ（行数）
    int  width;        // 横幅（列数）

    cout << "長方形を作りましょう。¥n";

    cout << "高さ：";   cin >> height;
    cout << "横幅：";   cin >> width;

    for (int i = 1; i <= height; i++) {
        for (int j = 1; j <= width; j++)
            cout << '*';
        cout << '¥n';
    }

    return (0);
}
```

```
長方形を作りましょう。
高さ：3
横幅：15
***************
***************
***************
```

## ■ 長方形を表示する（関数を利用）

```
/*
    長方形を表示する（関数を利用）
*/

#include <iostream.h>

/*--- no個の'*'を表示 ---*/
void put_star(int no)
{
    for (int i = 0; i < no; i++)
        cout << '*';
}

int main(void)
{
    int  height;       // 高さ（行数）
    int  width;        // 横幅（列数）

    cout << "長方形を作りましょう。¥n";

    cout << "高さ：";   cin >> height;
    cout << "横幅：";   cin >> width;

    for (int i = 1; i <= height; i++) {
        put_star(width);
        cout << '¥n';
    }

    return (0);
}
```

```
長方形を作りましょう。
高さ：3
横幅：15
***************
***************
***************
```

関数 put_star は、no 個だけ文字'*'を表示します。この関数を利用すると、main 関数の繰返し文は、２重ループから、ただのループへとなりました。

なお、関数 put_star は、長方形でなく、三角形を作るプログラムでも利用できます。プログラム例を示します。

### ■ 直角三角形を表示する（関数を利用）

```
/*
    直角三角形を表示する
*/

#include <iostream.h>

/*--- no個の'*'を表示 ---*/
void put_star(int no)
{
    for (int i = 0; i < no; i++)
        cout << '*';
}

int main(void)
{
    int  height;       // 高さ（行数）

    cout << "直角三角形を作りましょう。¥n";

    cout << "高さ：";
    cin >> height;

    for (int i = 1; i <= height; i++) {
        put_star(i);
        cout << '¥n';
    }

    return (0);
}
```

```
直角三角形を作りましょう。
高さ：10
*
**
***
****
*****
******
*******
********
*********
**********
```

長方形を書くための下請けとして作成した関数が、そのまま別の関数に流用できたことが分かりますね。

最初は、関数を定義するのは難しく感じられるかもしれませんが、いったんマスターしてしまえば、『関数のないプログラムなんか作ることはできない』ことに気付くでしょう。

さて、左下側が直角の三角形は、うまく表示できますが、右下側が直角の三角形の表示は簡単ではありません。というのも、'*'ではなく、空白文字' 'を出力する部品がないからです。

似たような部品を別途作るのではなく、まとめた方が、融通がききます。プログラム例を示します。

### ■ 直角三角形を表示する（関数を利用）

```
/*
    直角三角形を表示する
*/

#include <iostream.h>

/*--- no個の文字chを表示 ---*/
void put_nchar(char ch, int no)
{
    for (int i = 0; i < no; i++)
        cout << ch;
}

int main(void)
{
    int  height;       // 高さ（行数）

    cout << "直角三角形を作りましょう。¥n";

    cout << "高さ：";
    cin >> height;

    for (int i = 1; i <= height; i++) {
        put_nchar (' ', height - i);    // height - i個の空白
        put_nchar ('*', i);             // i個の*
        cout << '¥n';
    }

    return (0);
}
```

```
直角三角形を作りましょう。
高さ：10
          *
         **
        ***
       ****
      *****
     ******
    *******
   ********
  *********
 **********
```

関数 put_nchar は、文字 ch を no 個だけ連続表示する関数です。したがって、アスタリスク文字や空白文字だけでなく、あらゆる文字を連続表示することができます。

## ■ 演習問題（1）

◆ 以下に示すのは、二つの引数の大きい方の値を返却する関数の実体を定義する (1) であり、(a)部の (2) と(b)部の (3) とから構成される。

なお、(a)の冒頭におかれた int は、この関数の (3) 型が int 型であることを示すものである。

(a)
(b)

```
int maxof(int x, int y)
{
    if (x > y)
        (5) ;
    else
        (5) ;
}
```

◆ 関数呼び出しの際に、呼び出す側が渡す引数を (7) 、呼び出される側が受け取る引数を (8) と呼び、 (8) は (7) によって初期化される。

◆ 下に示す sum は、二つの int 型引数の値の和を返却する関数、右に示す diff は、二つの int 型引数の値の差を返却する関数である。

```
int sum(int x, int y)
{
    return ( (9) );
}

int diff(int x, int y)
{
    if ( (10) )
        return ( (11) );
    return ( (12) );
}
```

◆ 以下に示すのは、いずれも引数として受け取った三つの整数の平均値を返却する関数である。

```
double aveof(int x, int y, int z)
{
    return ( (13) (x + y + z) / 3);
}

double aveof(int x, int y, int z)
{
    return ((x + y + z) / ( (14) )
}
```

上に示した関数で利用したような、ある型の式の値を、他の型での値へと変換する式を (15) 式と呼び、そのための演算子()を (16) 演算子と呼ぶ。

| | |
|---|---|
| (1) | |
| (2) | |
| (3) | |
| (4) | |
| (5) | |
| (6) | |
| (7) | |
| (8) | |
| (9) | |
| (10) | |
| (11) | |
| (12) | |
| (13) | |
| (14) | |
| (15) | |
| (16) | |

## ■ プログラム作成演習（１）

**(5-1a)**　受け取った浮動小数点引数の２乗値を返却する関数を作成せよ。

```
#include  <iostream.h>

double sqr(double x)
{
    return (      );
}

int main(void)
{
    double  vx;

    cout     "実数を入力してください：";
    cin      vx;

    cout << "その数の2乗は" << sqr(vx) << "です。¥n";

    return (0);
}
```

```
実数を入力してください：3.5
その数の2乗は12.25です。
```

**(5-1b)**　同上

```
#include  <iostream.h>

double sqr(double x)
{
    double  y = x;
    y    x;
    return (y);
}
```

（以下同じ）

**(5-1c)**　同上

```
#include  <iostream.h>

double sqr(double x)
{
    double  y = 1.0;
    y    x;
    y    x;
    return (y);
}
```

（以下同じ）

**(5-2a)**　受け取った二つの引数の値の差を返却する関数を作成せよ。

```
[          ] <[          ]>

int diff(int x, int y)
{
    if (x > y)
        return([      ]);
    else
        return([      ]);
}

int [      ]([      ])
{
    int  a, b;

    cout [  ] "整数(１)を入力してください：";    cin [  ] a;
    cout [  ] "整数(２)を入力してください：";    cin [  ] b;

    cout [  ] "それらの差は" << [      ](a, b) << "です。¥n";

    return (0);
}
```

```
整数(１)を入力してください：12
整数(１)を入力してください：35
それらの和は47です。
```

**(5-2b)**　同上（関数 *diff* 以外は省略）

```
int diff(int x, int y)
{
    int  sa;
    if (x - y  >= 0)
        sa = [      ];
    else
        sa = [      ];
    return(sa);
}
```

**(5-2c)**　同上（関数 diff 以外は省略）

```
int diff(int x, int y)
{
    int  sa = x - [  ];
    if (sa >= 0)
        return(sa);
    else
        return(sa * [  ]);
}
```

**(5-3a)**　受け取った二つの整数引数の平均値を実数で返却する関数を作成せよ。

```
#include  <iostream.h>

double aveof(int x, int y)
{
    return (double(      ) / 2);
}

int main(void)
{
    int  a, b;

    cout << "整数を入力してください：";   cin >> a;
    cout << "整数を入力してください：";   cin >> b;

    cout << "平均値は" << aveof(a   b) << "です。¥n";

    return (0);
}
```

```
整数を入力してください：3
整数を入力してください：4
それらの平均は3.5です。
```

**(5-3b)**　同上（関数 aveof 以外は省略）

```
double aveof(int x, int y)
{
          s = x + y;
    return (s / 2);
}
```

**(5-3c)**　同上（関数 aveof 以外は省略）

```
double aveof(int x, int y)
{
    return ((x + y) /    );
}
```

**(5-3d)**　同上（関数 aveof 以外は省略）

```
double aveof(int x, int y)
{
    int  s = x + y;
    return (       / 2);
}
```

**(5-4a)**　整数値を引数 *num* に受け取って、1 + 2 + …+ *num* の値を返却する関数を作成せよ。

```
#include <iostream.h>

int sumup(int num)
{
    int  sum = [ ];
    for (int i = 1; i <= num; i++)
        sum += [ ];
    return (sum);
}

int main(void)
{
    int  a;

    cout << "整数を入力してください：";   cin >> a;
    cout << sumup([ ]) << "¥n";

    return (0);
}
```

```
整数を入力してください：3↵
6
```

**(5-4b)**　同上

```
int sumup(int num)
{
    int  sum = [ ];
    while ([ ]) {
        sum += [ ];
        [ ]--;
    }
    return (sum);
}
```

**(5-4c)**　同上

```
int sumup(int num)
{
    int  sum = [ ];
    for (int i = num; i >= [ ]; i--)
        sum += [ ];
    return (sum);
}
```

# 5-2　参照

## ■ void 関数

関数は、必ずしも値を返却しなくても構いません。値を返却しない関数を **void** 関数と呼びます。

## ■ 値渡しと参照渡し

### ■ 値渡し

◎ 仮引数は、実引数の値で初期化される。

```
void alert(int no)
{
    while (no > 0) {
        cout << '¥a';
        no--;
    }
}
```

**int** 型変数 **x** の値が 5 であるとき、**alert(x)** と呼び出した後も、**x** の値は 5 のままです。

### ■ 参照渡し

※ 仮引数の宣言には&が必ず必要。

◎仮引数は、実引数と実体を共有する。

```
void alert(int &no)
{
    while (no > 0) {
        cout << '¥a';
        no--;
    }
}
```

仮引数の宣言に**&**を付ける
喩え：**no** は《ヤドカリ》です。実体をもちません。実引数に住みます。

**int** 型変数 **x** の値が 5 であるとき、**alert(x)** と呼び出した後は、**x** の値は **0** になります。

次ページのプログラムで確認しましょう(**p.28** のプログラムと比較してみてください)。

■ 指定された回数警報をならす関数（？？？）

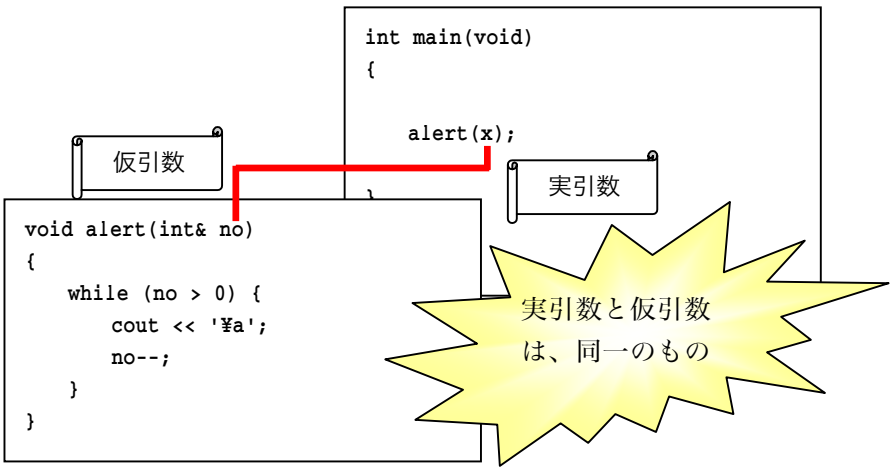
```
/*
    指定された回数警報をならす関数
*/

#include  <iostream.h>

/*--- no回警報をならす ---*/
void alert(int& no)
{
    while (no > 0) {
        cout << '¥a';
        no--;
    }
}

int main(void)
{
    int  x;

    cout << "何回警報をならしますか：";
    cin >> x;

    alert(x);

    // xの値は書き換わってしまう

    cout << x << "回警報をならしました。¥n";

    return (0);
}
```

何回警報をならしますか：5
0回警報をならしました。

x の値は 0 になってしまいます。

仮引数noの値は0になります。

参照渡しなので実引数xの値は変化してしまう。



## ■ 演習問題

■ 関数間の引数の受け渡しは、原則として (1) によって行われるが、仮引数を&付きで宣言することにより、 (2) によって行われることになる。

■ 以下に示すのは、二つの **int** 型引数の値を交換する関数である。

```
void swap( (3) x, (4) y)
{
    (5) temp = x;
    x = (6);
    y = (7);
}
```

■ 以下に示すのは、上に示した関数 swap を利用して二つの引数 a, b を昇順に（a≦b となるように）並べ替える関数である。

```
void sort2( (8) a, (9) b)
{
    if (b < (10))
        swap(a, b);
}
```

■ 以下に示すのは、受け取った int 型の参照引数の値を 2 倍にする関数である。

```
void bai( (11) a)
{
    a *= (12);
}
```

■ 受け取った **int** 型の参照引数を、絶対値にする関数を作成せよ。

```
void cabs( (11) a)
{
    [                ]
    [                ]
    [                ]
}
```

| | |
|---|---|
| (1) | |
| (2) | |
| (3) | |
| (4) | |
| (5) | |
| (6) | |
| (7) | |
| (8) | |
| (9) | |
| (10) | |
| (11) | |
| (12) | |

## 5-3　配列の受け渡し

多くの関数に触れてみましょう。

■ 受け取った配列の全要素の値を表示する関数

```
/*
    受け取った配列の要素の値をすべて表示
*/

#include  <iostream.h>

//--- 要素数noの配列aの各要素の値をすべて表示 ---//
void ary_puts(int a[], int no)
{
    for (int i = 0; i < no; i++)
        cout << "a[" << i << "] = " << a[i] << '¥n';
}

int main(void)
{
    const int max = 10;       // 要素数
    int x[max];               // 要素数がmax個の配列

    for (int i = 0; i < max; i++) {
        cout << "x[" << i << "] : ";
        cin >> x[i];
    }

    ary_puts(x, max);

    return (0);
}
```

```
x[0] : 3
x[1] : 6
x[2] : 8
x[3] : 12
x[4] : 6
x[5] : 7
x[6] : 5
x[7] : 10
x[8] : 2
x[9] : 9
a[0] = 3
a[1] = 6
a[2] = 8
a[3] = 12
a[4] = 6
a[5] = 7
a[6] = 5
a[7] = 10
a[8] = 2
a[9] = 9
```

const を付けると定数となります。

■　実引数と仮引数の名前は同一である必要はありません。

■　要素数を 10 以外の値に変更したければ、

```
const int max = 10;
```

の宣言のみを書き換えればよいです。

■ 受け取った配列の全要素に添字と同じ値を代入する関数
※すなわち、先頭から順に 0, 1, 2, …を代入します。

```
/*
   受け取った配列に添字と同じ値を代入
*/

#include  <iostream.h>

//--- 要素数noの配列aの要素に添字と同じ値を代入 ---//
void fill_idx(int a[], int no)
{
   for (int i = 0; i < no; i++)
      a[i] = i;
}

//--- 要素数noの配列aの各要素の値をすべて表示 ---//
void ary_puts(int a[], int no)
{
   for (int i = 0; i < no; i++)
      cout << "a[" << i << "] = " << a[i] << '¥n';
}

int main(void)
{
   const int max = 10;    // 要素数
   int x[max];            // 要素数がmax個の配列

   fill_idx(x, max);      // 添字と同じ値をセット

   ary_puts(x, max);      // 表示

   return (0);
}
```

```
a[0] = 0
a[1] = 1
a[2] = 2
a[3] = 3
a[4] = 4
a[5] = 5
a[6] = 6
a[7] = 7
a[8] = 8
a[9] = 9
```

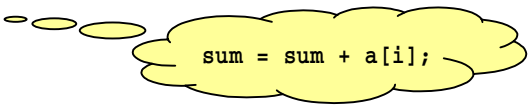
この関数は前ページの
ものをそのまま流用。

■ 受け取った配列の全要素に 0 を代入する関数
上のプログラムの fill_idx を fill_zero におきかえてプログラムを完成させましょう。

```
//--- 要素数noの配列aの要素に0を代入 ---//
void fill_zero(int a[], int no)
{
   for (int i = 0; i < no; i++)
      a[i] = 0;
}
```

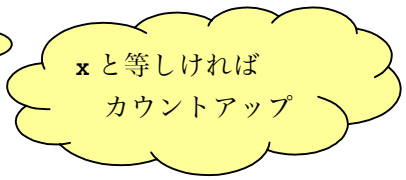
■ 受け取った配列の全要素の合計を求める

```
/*
   受け取った配列の要素の合計値を求める
*/

#include  <iostream.h>

//--- 要素数noの配列aの全要素の合計値を求める ---//
int ary_sum(int a[], int no)
{
   int  sum = 0;          // 合計値

   for (int i = 0; i < no; i++)
      sum += a[i];
   return (sum);
}

int main(void)
{
   const int max = 10;    // 要素数
   int x[max];            // 要素数がmax個の配列

   for (int i = 0; i < max; i++) {
      cout << "x[" << i << "] : ";
      cin >> x[i];
   }

   cout << "合計は" << ary_sum(x, max) << "です。¥n";

   return (0);
}
```



■ 受け取った配列からある値の要素をもつ個数を求める

```
/*
   受け取った配列からある値をもつ要素の個数を求める
*/

#include  <iostream.h>

//--- 要素数noの配列aから値xをもつ要素数を求める ---//
int ary_cnt(int a[], int no, int x)
{
   int  count = 0;        // 要素数

   for (int i = 0; i < no; i++)
      if (a[i] == x)
         count++;
   return (count);
}

int main(void)
{
   const int max = 10;    // 要素数
   int x[max];            // 要素数がmax個の配列
   int v;                 // この値を探索

   for (int i = 0; i < max; i++) {
      cout << "x[" << i << "] : ";
      cin >> x[i];
   }

   cout << "探索する値を入力してください：";
   cin >> v;

   cout << "その値をもつ要素は" << ary_cnt(x, max, v) << "個です。¥n";

   return (0);
}
```



■ 受け取った配列の要素の並びを逆転する

```
/*
    受け取った配列の並びを逆転する
*/

#include  <iostream.h>

//--- x, yの値を交換する ---//
void swap(int &x, int & y)
{
    int temp = x;
    x = y;
    y = temp;
}

//--- 要素数noの配列aの並びを逆転する ---//
void ary_rvs(int a[], int no)
{
    for (int i = 0; i < no / 2; i++)
        swap(a[i], a[no - i - 1]);
}

//--- 要素数noの配列aの各要素の値をすべて表示 ---//
void ary_puts(int a[], int no)
{
    for (int i = 0; i < no; i++)
        cout << "a[" << i << "] = " << a[i] << '¥n';
}

int main(void)
{
    const int max = 10;    // 要素数
    int x[max];            // 要素数がmax個の配列

    for (int i = 0; i < max; i++) {
        cout << "x[" << i << "] : ";
        cin >> x[i];
    }

    ary_rvs(x, max);
    cout << "配列の並びを逆転しました。¥n";

    ary_puts(x, max);

    return (0);
}
```

List5-5 と同じ。

この関数は p.41 のものをそのまま流用。

■ 身長と体重から BMI（body mass index）を求める

List5-7 を書きかえて作りましょう。

```
/*
   10人のBMIを求める
*/

#include  <iostream.h>
#include  <iomanip.h>

//--- 身長体重からBMIを求める ---//
void get_bmi(int h[], int w[], double b[], int n)
{
   for (int i = 0; i < n; i++)
      b[i] = w[i] / (h[i] / 100.0) / (h[i] / 100.0);
}

int main(void)
{
   const int ninzu = 10;
   int    height[ninzu];      // 身長
   int    weight[ninzu];      // 体重
   double bmi[ninzu];      // BMI

   cout << ninzu << "人の身長と体重を入力せよ：¥n";
   for (int i = 0; i < ninzu; i++) {
      cout << setw(2) << i+1 << "番目の身長：";
      cin >> height[i];
      cout << setw(2) << i+1 << "番目の体重：";
      cin >> weight[i];
   }
   get_bmi(height, weight, bmi, ninzu);

   for (i = 0; i < ninzu; i++)
      cout << "No." << i
           << setw(5) << height[i] << "cm"
           << setw(5) << weight[i] << "kg"
           << setw(8) << bmi[i] << '¥n';

   return (0);
}
```

BMI は、体重／身長$^2$です。単位は kg と cm です。ちなみに 22〜23 前後が適正です（統計的に病気になりにくいことが分かっています）。

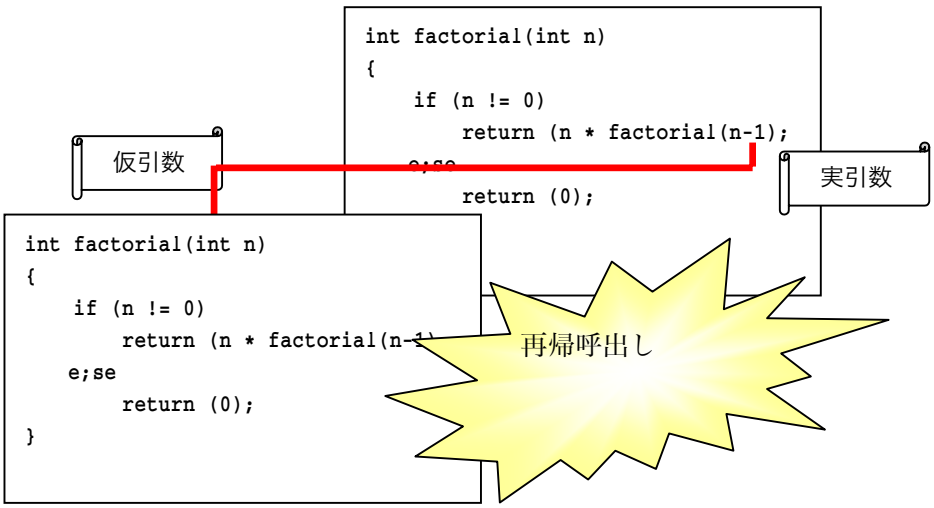
■ 数当てゲーム（入力履歴と正解との差を表示）

```
/*
   数当てゲーム（その５）
*/

#include  <time.h>
#include  <stdlib.h>
#include  <iomanip.h>
#include  <iostream.h>

//--- 入力履歴を表示 ---//
void put_history(int buff[], int x, int cnt)
{
   cout << "--------------¥n";
   cout << " No 入力   差 ¥n";
   cout << "--------------¥n";
   for (int i = 0; i < cnt; i++)
      cout << setw(3) << i << setw(5) << buff[i]
          << setw(5) << buff[i] - x << '¥n';
}

int main(void)
{
   const int max = 12;    /* 入力制限回数 */
   int  x;                /* 読み込んだ値 */
   int  no;            /* この数を当てさせる  */
   int x_buff[max];       /* 入力された数値をすべて格納 */
   int count = 0;         /* 何回目の入力か */

   srand(time(NULL));     /* 乱数の種を初期化 */
   no = rand() % 100;     /* 0〜999の乱数を発生 */

   do {
      cout << "残り" << max - count << "回です。¥n";
      cout << "整数を入力せよ：";
      cin >> x;

      x_buff[count] = x;
      count++;

      if (x > no)
         cout << "¥a大きいです。¥n";
      else if (x < no)
         cout << "¥a小さいです。¥n";
   } while (x != no && count < max);

   if (x == no) {
```

```
        cout << "正解です。¥n";
        cout << count << "回で当たりましたね。¥n";
    }

    put_history(x_buff, no, count);

    return (0);
}
```

## 4-4　再帰

関数の中から、自分自身と同じ関数を呼び出すことを再帰関数呼出しといいます。




これは、自分自身の関数を呼び出しているのではなく、自分自身と同じ関数を別途呼び出していることに注意しましょう。

本当に自分自身を呼び出すのだったら、えんえんと自分自身を呼び出すことになってしまいます。

## 演習問題

**(01)** n1 と n2 間の全整数を昇順に 1 行に 10 個ずつ 6 桁で表示する関数 void putint (int n1, int n2)を作成せよ。たとえば n1 が 5 で n2 が 24 のときは（逆に n1 が 24 で n2 が 5 のときも）以下のように表示する。

```
     5     6     7     8     9    10    11    12    13    14
    15    16    17    18    19    20    21    22    23    24
```

**(02)** n1 と n2 間の全整数を降順に 1 行に 10 個ずつ 6 桁で表示する関数 void putintr (int n1, int n2)を作成せよ。

**(03)** 記号文字'+'を用いて、no 段の左下側が直角の三角形を表示する関数 void triangle1(int no)を作成せよ。

```
+
++
+++
```

**(04)** 記号文字'*'を用いて、no 段の左上側が直角の三角形を表示する関数 void triangle2(int no)を作成せよ。

**(05)** 記号文字'-'を用いて、no 段の右下側が直角の三角形を表示する関数 void triangle3(int no)を作成せよ。

```
  -
 --
---
```

**(06)** 記号文字'='を用いて、no 段の右上側が直角の三角形を表示する関数 void triangle4(int no)を作成せよ。

**(07)** no 段の数字ピラミッドを表示する関数 void pm(int no)を作成せよ。表示する数字は(段数 % 10)とする。すなわち no が 11 のときは、右図のように表示する

```
          1
        222
       33333
      4444444
     555555555
    66666666666
   7777777777777
  888888888888888
 99999999999999999
 0000000000000000000
11111111111111111111
```

**(08)** n1 行 n2 列の数字長方形を表示する関数 void rct (int n1, int n2)を作成せよ。n1 が 11 で n2 が 12 ののときは、左図のように表示する。

```
123456789012
234567890123
345678901234
456789012345
567890123456
678901234567
789012345678
890123456789
901234567890
012345678901
123456789012
```

**(09)**　no が 0 であれば 1 を、そうでなければ 0 を返す関数 int is_zero(int no)を作成せよ。

**(10)**　no が正であれば 1 を、そうでなければ 0 を返す関数 int is_plus(int no)を作成せよ。

**(11)**　no が負であれば 1 を、そうでなければ 0 を返す関数 int is_minus(int no)を作成せよ。

**(12)**　no が正であれば 1 を、0 であれば 0 を、負であれば-1 を返す関数 int sign_of(int no)を作成せよ。

**(13)**　n1 と n2 が等しければ 1 を、等しくなければ 0 を返す関数 int is_equal (int n1, int n2)を作成せよ。

**(14)**　n1 が n2 より大きければ 1 を、等しければ 0 を、小さければ-1 を返す関数 int int_cmp (int n1, int n2)を作成せよ。

**(15)**　要素数が no である配列 a 中から正の値をもつ要素をすべて合計した値を返す関数 int ary_sump(int a[], int no)を作成せよ。

**(16)**　要素数が no である配列 a から負の値をもつ要素をすべて合計した値を返す関数 int ary_summ(int a[], int no)を作成せよ。

**(17)**　要素数が no である配列 a 中から x 以上の値をもつ要素の個数を返す関数 int ary_cntge(int a[], int no, int x)を作成せよ。

**(18)**　要素数が no である配列 a 中から x 以下の値をもつ要素の個数を返す関数 int ary_cntge(int a[], int no, int x)を作成せよ。

**(19)**　要素数が no である配列 a の全要素に 0 を代入する関数 void fill_zero (int a[], int no)を作成せよ。

**(20)**　要素数が no である配列 a の要素に添字と同じ値（すなわち先頭から順に 0, 1, 2, …）を代入する関数 void fill_idx (int a[], int no)を作成せよ。

**(21)**　要素数が no である配列 a の要素に no から添字を引いた（すなわち no が 5 であれば先頭から順に 5, 4, 3, 2, 1）を代入する関数 `void fill_idxr (int a[], int no)` を作成せよ。

**(22)**　要素数が no である配列 a の全要素の平均値を double 型で返す関数 `double ave_of (int a[], int no)` を作成せよ。

**(23)**　要素数が no である配列 a の要素の値の範囲（すなわち最大値と最小値の差）を返す関数 `int range_of (int a[], int no)` を作成せよ。

**(24)**　要素数が no である配列 a の要素の値を、最小値との差に変更する関数 `void offset_of1(int a[], int no)` を作成せよ。たとえば、要素数が 5 であり、その要素の値が順に 5, 3, 7, 2, 6 であれば最小値は 2 であるので、各要素の値を 3, 1, 5, 0, 4 にする。

**(25)**　要素数が no である配列 a の要素の値を、最大値との差に変更する関数 `void offset_of2(int a[], int no)` を作成せよ。

## 演習問題

**(01)** 底辺 w、高さ h の三角形の面積を double 型の実数値で返す関数 double ts(int w, int h)を作成せよ。

**(02)** 上底 w1、下底 w2、高さ h の台形の面積を double 型の実数値で返す関数 double tt(int w1, int w2, int h)を作成せよ。

**(03)** 半径 r の円の面積を double 型の実数値で返す関数 double tu(int r)を作成せよ。ただし円周率は 3.14 とする。

**(04)** n1 と n2 間の全整数の合計値を返す int s1(int n1, int n2)を作成せよ。たとえば n1 が 5 で n2 が 7 のときは（逆に n1 が 7 で n2 が 5 のときも）18 を返す。

**(05)** n1 以下の全ての正の整数の合計値を返す int s2(int n1)を作成せよ。たとえば n1 が 5 のときは 1 + 2 + 3 + 4 + 5 すなわち 15 を返す。ただし、n1 が負のときは-1 を返すこと。

**(06)** n1 以下の全ての正の偶数の合計値を返す int s3(int n1)を作成せよ。たとえば n1 が 5 のときは 2 + 4 すなわち 6 を返す。ただし、n1 が負のときは-1 を返すこと。

**(07)** n1 以下の全ての３の倍数の合計値を返す int s4(int n1)を作成せよ。たとえば n1 が 10 のときは 3 + 6 + 9 すなわち 18 を返す。ただし、n1 が負のときは-1 を返すこと。

**(08)** n1 の下から 2 桁目の値を返す関数 int d1(int n1)を作成せよ。たとえば n1 の値が 123 であれば 2 を返す。ただし、n1 が 10 未満のときは-1 を返すこと。

**(09)** n1 の各桁の合計値を返す関数 int d2(int n1)を作成せよ。たとえば n1 の値が 325 であれば 3 + 2 + 5 すなわち 10 を返す。ただし、負のときは-1 を返すこと。

**(10)** n1 の桁数を返す関数 int d3(int n1)を作成せよ。たとえば n1 の値が 125 であれば 3 を、75 であれば 2 を返す。ただし、負のときは-1 を返すこと。

**(11)** 要素数が no である配列 a 中から偶数である要素をすべて合計した値を返す関数 int ary_sum1(int a[], int no)を作成せよ。

**(12)** 要素数が no である配列 a から奇数である要素をすべて合計した値を返す関数 int ary_sum2(int a[], int no)を作成せよ。

**(13)** 要素数が no である配列 a から偶数である要素の個数を返す関数 int ary_cnt1(int a[], int no)を作成せよ。

**(14)** 要素数が no である配列 a 中から奇数である要素の個数を返す関数 int ary_cnt2 (int a[], int no)を作成せよ。

**(15)** 要素数が no である配列 a の全要素の符号を逆転する関数 void ary_rsign(int a[], int no)を作成せよ。たとえば、要素数が 5 であり、その要素の値が順に 5, 3, 7, 2, 6 であれば、各要素の値を-5, -3, -7, -2, -6 にする。

**(16)** 要素数が no である配列 a の全要素を、正であれば 1、0 であれば 0、負であれば-1 とする関数 void ary_sign(int a[], int no)を作成せよ。たとえば、要素数が 5 であり、その要素の値が順に 5, 0, -7, -5, 6 であれば、各要素の値を 1, 0, -1, -1, 1 にする。