

関数

三池 克明

関数とは `printf()` や `scanf()` のように `()` 内の引数から計算等の処理をしてくれるものです。細かい処理を関数化することで処理の流れを把握しやすくなりますのでぜひマスターして下さい。

—目 次—

1.	関数を使ったプログラム.....	1
1.1.	2 整数の比較をする関数.....	1
1.2.	関数とは	3
2.	関数を記述するには.....	4
2.1.	整数の割り算を計算する関数.....	4
2.2.	プロトタイプ宣言.....	6
2.3.	関数の記述	8
2.4.	引数の変数名を変えてみる.....	10
2.5.	関数の内容を短くする.....	12
2.6.	引数の内容を書き換える.....	13
3.	関数を使ったプログラム.....	16
3.1.	絶対値を求める	16
3.2.	二次方程式の判別式を求める.....	20
3.3.	階乗を求める	25
4.	演習問題.....	29

1. 関数を使ったプログラム

1.1. 2 整数の比較をする関数

まずは以下のソースプログラムを作成し、コンパイル・実行させてみましょう。
初めてみる記述がいくつかありますが、後で説明しますので今は深く考えないで下さい。

func1.cpp

```
1: #include <stdio.h>
2:
3: void comp(int, int);
4:
5: main()
6: {
7:     int    a, b;
8:
9:     printf("a = ");
10:    scanf("%d", &a);
11:    printf("b = ");
12:    scanf("%d", &b);
13:
14:    comp(a, b);
15: }
16:
17: void comp(int a, int b)
18: {
19:     if(a > b){
20:         printf("aの方が大きいです\n");
21:     } else if(a < b){
22:         printf("bの方が大きいです\n");
23:     } else {
24:         printf("aとbは等しいです\n");
25:     }
26: }
```

実行例 1

a = 10
b = 3
aの方が大きいです

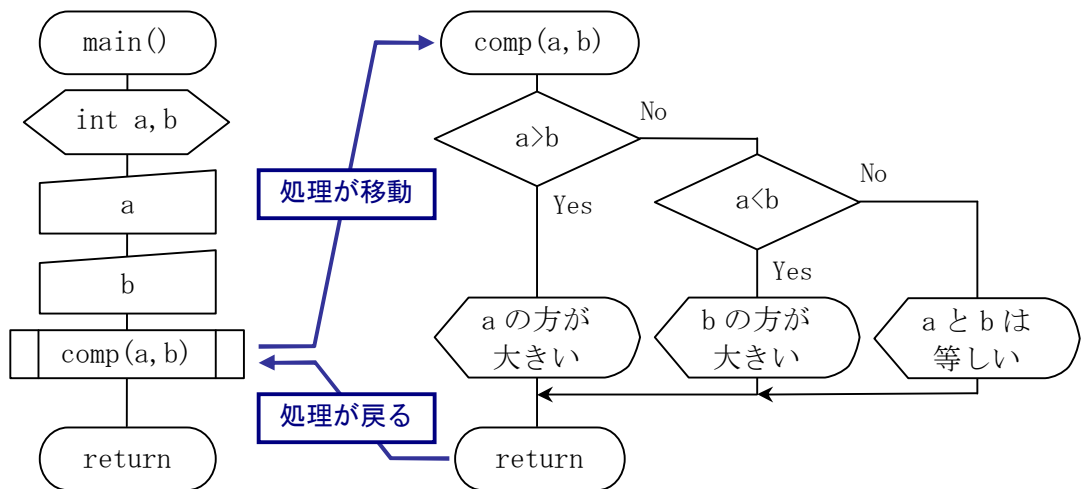
実行例 2

a = -3
b = 1
bの方が大きいです

実行例 3

a = 0
b = 0
aとbは等しいです

func1.cpp をフローチャートで表すとこのようになります。



main()チャートの中にある「comp(a, b)」に処理が来たとき、前ページの図に示すように処理が comp(a, b)チャートに移動します。

その処理が終わると再び「main()」に処理が戻ります。

1.2. 関数とは

C 言語は処理を「関数」という部品で分割することができます。func1. cpp では、

`main()`…処理全体

`comp()`…引数 a, b の大小を比較し結果を表示する。

となります。

このように処理を「関数」という部品にすることで以下の利点を得ることができます。

1. メンテナンス（調整や改良）を容易にする
機能ごとに関数化することで、全体を見渡したり、一部に注目したりすることができる
2. 開発作業を分担させる
関数ごとに担当を分担させるなど、作業を効率よく分散できる。

これは自動車や建築物が部品ごとに製造され組み立てられる概念に通じますね。

2. 関数を記述するには

2.1. 整数の割り算を計算する関数

以下のソースプログラム「func2.cpp」は二つの int 型の値の割り算を計算する（ただし結果は double 型）関数 divide() とその動作を確かめるプログラムです。このプログラムを例に関数の使い方をみてみましょう。

func2.cpp

```
1: #include <stdio.h>
2:
3: double divide(int, int);
4:
5: main()
6: {
7:     int    a, b;
8:
9:     printf("a = ");
10:    scanf("%d", &a);
11:    printf("b = ");
12:    scanf("%d", &b);
13:
14:    printf("divide(%d, %d) = %f¥n", a, b, divide(a, b));
15: }
16:
17: double divide(int a, int b)
18: {
19:     double v;
20:
21:     v = (double)a / b;
22:
23:     return v;
24: }
```

実行例 1

```
a = 10  
b = 3  
divide(10, 3) = 3.333333
```

実行例 2

```
a = 5  
b = 2  
divide(5, 2) = 2.500000
```

2.2. プロトタイプ宣言

関数を記述するには、その前に返値、関数名、引数の型を宣言する必要があります。これを**プロトタイプ宣言**といいます。

このプロトタイプ宣言は `main()` の前に記述しなければなりません。

`func2.cpp` の 3 行目をみてみましょう。

```
2:
3: double divide(int, int);
4:
```

これは関数 `divide()` のプロトタイプ宣言といいます。プロトタイプ宣言は `main()` 関数の手前で記述します。

また、その形式は以下ようになります。

返値の型 関数名 (引数の型…) ;

- **返値の型**

呼び出した関数に返す値の型です。関数 `divide()` では `double` 型です。

またここを `void` とすると返値が無いことをあらわします。

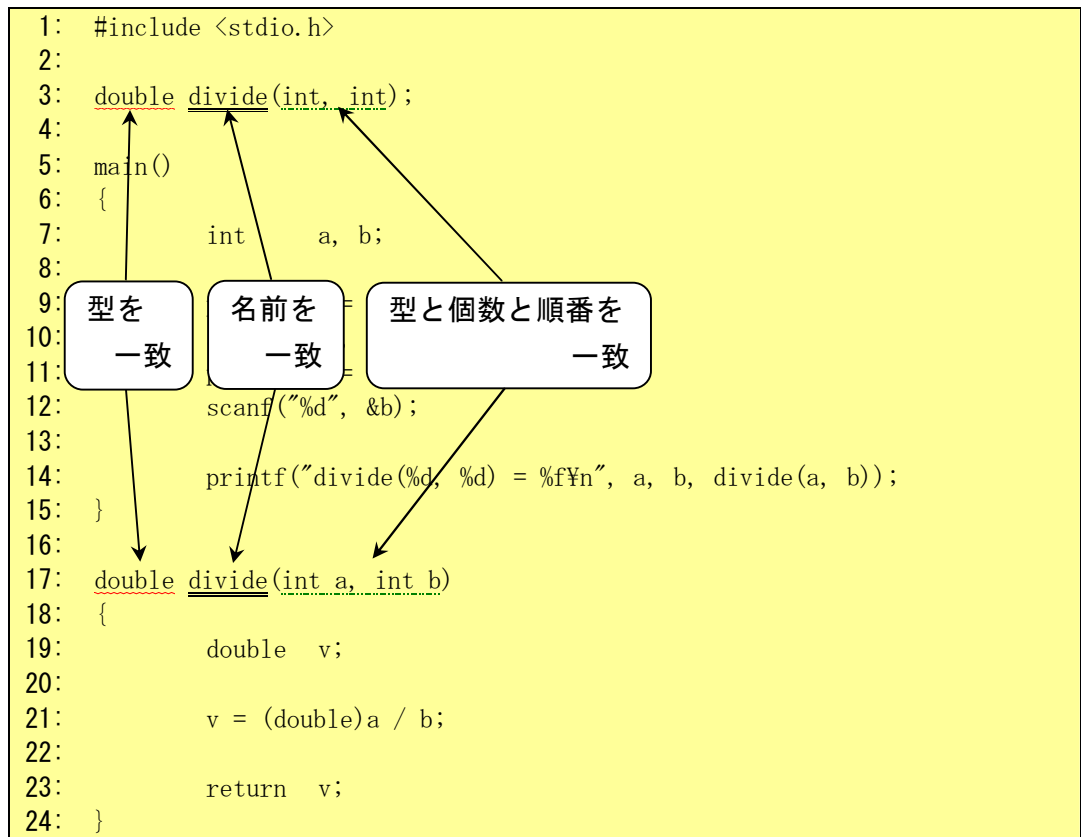
- **関数名**

関数の名前です。 `printf()` 関数であれば「`printf`」の部分になります。

- **引数の型**

関数に渡す値の型です。

またプロトタイプ宣言と実際に記述する関数は返値の型、関数名、引数の型と個数と順番を一致させなければなりません。



2.3. 関数の記述

関数の処理内容は main() 関数の後に記述します。
また処理内容の形式は main() 関数を参考にすると良いでしょう。

func2.cpp の 17 行目以降をみてみましょう。

```
17: double divide(int a, int b)
18: {
19:     double v;
20:
21:     v = (double)a / b;
22:
23:     return v;
24: }
```

この部分で関数 divide() の処理内容を記述しています。関数の内容はこのように main() の後に記述するようにしましょう（これ以外の記述法もありますがここでは割愛します）。

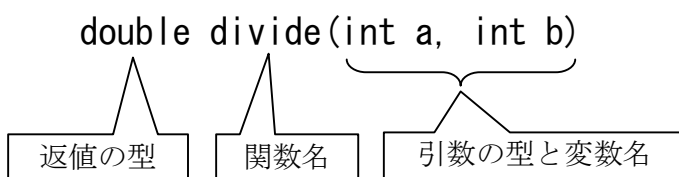
形式は以下のようになります。main() とほぼ同じであることがわかりますね。

```
    返値の型 関数名 (引数の型 変数名…)
    {
        変数宣言
        処理
        return 文
    }
```

- 返値の型 関数名 (引数の型 変数名…)

プロトタイプ宣言とほぼ同じですが、引数の部分には必ず変数名を記述します。なお、行末にセミコロンは必要ありません。

また func2. cpp の 17 行目の場合は



となります。

- **変数宣言と処理**

どちらも `main()` と同様のルールで記述します。if 文や for 文も利用できます。また、他の関数名（と引数）を記述して呼び出すこともできます。

ただし、`main()` 関数など他の関数で宣言した変数は利用できません。

- **return 文**

形式は以下のようになります。

return 式;

この文の式の値が返値になります。

返値が `void` 型の場合、式は必要ありません。また `void` 型は `return` 文そのものを省略することも可能です。

2.4. 引数の変数名を変えてみる

次に func2.cpp を少し改造したプログラム func3.cpp を入力し、コンパイル・実行してみましょう。

func3.cpp

```
1: #include <stdio.h>
2:
3: double divide(int, int);
4:
5: main()
6: {
7:     int    a, b;
8:
9:     printf("a = ");
10:    scanf("%d", &a);
11:    printf("b = ");
12:    scanf("%d", &b);
13:
14:    printf("divide(%d, %d) = %f¥n", a, b, divide(a, b));
15: }
16:
17: double divide(int x, int y)
18: {
19:     double v;
20:
21:     v = (double)x / y;
22:
23:     return v;
24: }
```

引数の変数名を
変えてみる

実行例

```
a = 10
b = 3
divide(10, 3) = 3.333333
```

このように引数の変数名を変更しても特に問題はありません。これは変数そのものではなく引数の値を渡しているからです。

なお、main()内で宣言した変数は divide()内で使用することは出来ませんので注意しましょう。これは逆の場合も同様です。

すなわち、関数内で宣言した変数はその関数内でしか利用できません。

2.5. 関数の内容を短くする

次は関数 `divide()` の内容を短くしたプログラムです。
入力し、コンパイル・実行してみましょう。

func4.cpp

```
1: #include <stdio.h>
2:
3: double divide(int, int);
4:
5: main()
6: {
7:     int    a, b;
8:
9:     printf("a = ");
10:    scanf("%d", &a);
11:    printf("b = ");
12:    scanf("%d", &b);
13:
14:    printf("divide(%d, %d) = %f¥n", a, b, divide(a, b));
15: }
16:
17: double divide(int x, int y)
18: {
19:     return (double)x / y;
20: }
```

return 文に
演算式を記述

実行例

```
a = 10
b = 3
divide(10, 3) = 3.333333
```

このように `return` 文に式を直接記述することでソースコードを短くすることもできます。

2.6. 引数の内容を書き換える

以下のソースプログラムは引数の内容を二倍する関数とその動作を確かめるプログラムです。

twice1.cpp

```
1: #include <stdio.h>
2:
3: void twice(double);
4:
5: main()
6: {
7:     double a;
8:
9:     printf("a = ");
10:    scanf("%lf", &a);
11:
12:    twice(a);
13:
14:    printf("a = %f¥n", a);
15: }
16:
17: void twice(double a)
18: {
19:     a *= 2;
20: }
```

実行例

```
a = 0.8
a = 0.800000
```

おや？ 二倍になりませんね。
その理由を考えてみましょう。
これは、

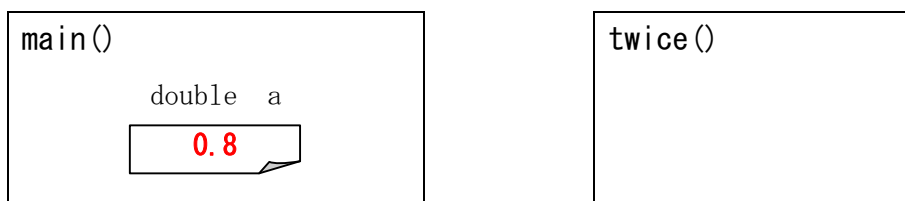
- 変数そのものではなく引数の値を渡している
- 関数内で宣言した変数はその関数内ではしか利用できない

が理由です。

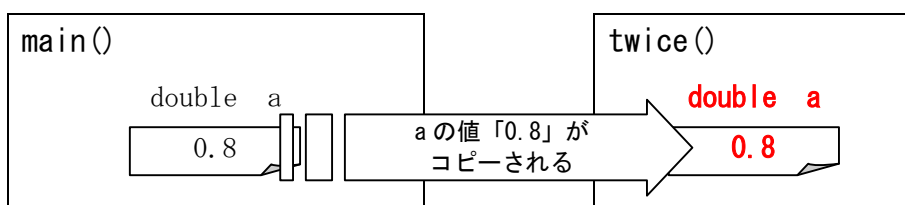
twice1.cpp では main()、twice() 両方に double 型の変数 a を使用していますが、main() の変数 a と twice() の変数 a は違う変数です。同姓同名の別人のようなものです。

下図のようにイメージすればわかると思います。

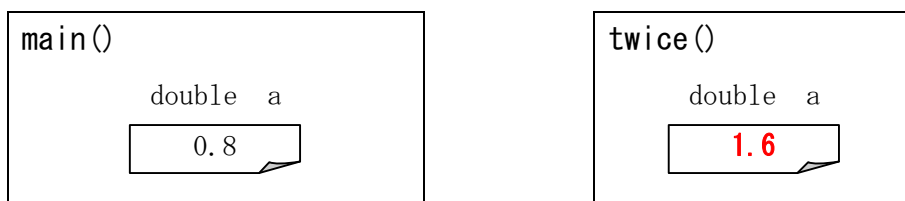
10 行目：scanf() で変数 a に数値を入力



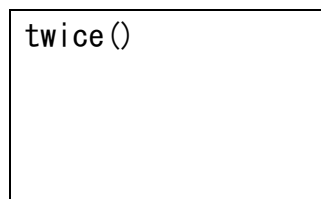
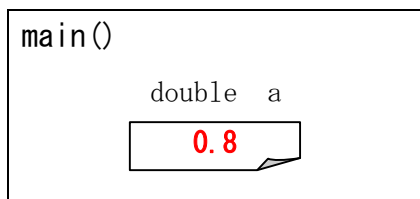
12 行目：twice(a) を実行



17 行目：twice() の変数 a の値が 2 倍される



14 行目 : main() の変数 a の値が表示される



よって二倍されるのは `twice()` の変数 a であって `main()` の変数 a ではありません。
この問題は二倍した値を返値として返せば解決します。
それが以下の `twice2.cpp` です。

`twice2.cpp`

```
1: #include <stdio.h>
2:
3: double twice(double);
4:
5: main()
6: {
7:     double a;
8:
9:     printf("a = ");
10:    scanf("%lf", &a);
11:
12:    a = twice(a);
13:
14:    printf("a = %f\n", a);
15: }
16:
17: double twice(double a)
18: {
19:     return a * 2;
20: }
```

実行例

```
a = 0.8
a = 1.600000
```

3. 関数を使ったプログラム

3.1. 絶対値を求める

以下の関数 `intabs()` を作り、動作を確かめてみましょう。

プロトタイプ	<code>int intabs(int value)</code>	
引数	<code>int value</code>	絶対値を求めたい値
返値	<code>int</code>	<code>value</code> の絶対値

【解説】
引数の絶対値を求めます。

【例】

```
a = intabs(-50);
```

-50 の絶対値である 50 が `a` に代入されます。

この仕様より、プロトタイプ宣言は、

```
int intabs(int);
```

とすればよいことがわかります。また関数を記述するときは、

```
int intabs(int value)
{
    (ここに処理を記述)
}
```

とすればよいことがわかります。

なお絶対値とは数値の符号部を取った値のことで $|x|$ の形式で表します。

$$\begin{aligned}|-5.2| &= 5.2 \\|+1.32| &= 1.32 \\|25| &= 25\end{aligned}$$

といっても上記の例からわかるように正の数はそのままで、負の数なら符号を逆転させるだけで十分なことがわかります。そこで `intabs()` では次のように処理をさせることにしましょう。

1. 負の数なら-1倍して符号を正にし、それを返値にする
2. それ以外ならそのまま返値にする

そして最終的な `intabs()` の仕様は以下のようになります。

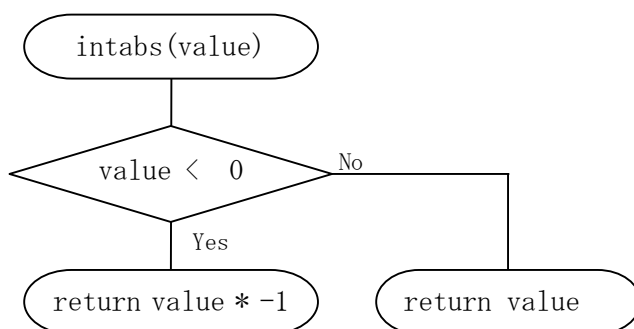
<関数 `intabs()` の仕様>

1. 仕様

プロトタイプ	<code>int intabs(int value)</code>	
引数	<code>int value</code>	絶対値を求めたい値
返値	<code>int</code>	<code>value</code> の絶対値
【解説】 引数の絶対値を求めます。		
【例】 <code>a = intabs(-50);</code> -50 の絶対値である 50 が <code>a</code> に代入されます。		

2. 関数内で使用する変数
なし。

3. フローチャート



4. ソースコード

```
int intabs(int value)
{
    if(value < 0){
        return value * -1;
    }

    return value;
}
```

<以上>

それでは intabs() の動作確認プログラムを作りましょう。
以下のプログラムを入力し、コンパイル・実行します。

```
1: #include <stdio.h>
2:
3: int intabs(int);
4:
5: main()
6: {
7:     int    v;
8:
9:     printf("v = ");
10:    scanf("%d", &v);
11:
12:    printf("|v| = |%d| = %d\n", v, intabs(v));
13: }
14:
15: int intabs(int value)
16: {
17:     if(value < 0){
18:         return  value * -1;
19:     }
20:
21:     return  value;
22: }
```

実行例 1

```
v = -10
|v| = |-10| = 10
```

実行例 2

```
v = 22
|v| = |22| = 22
```

どうやら問題なさそうですね。

3.2. 二次方程式の判別式を求める

二次方程式の判別式を求める関数 `hanbetsu()` を作ってみましょう。

以下の二次方程式

$$ax^2 + bx + c = 0$$

の判別式 D は

$$D = b^2 - 4ac$$

でしたよね。

よって `hanbetsu()` の引数は二次方程式の各項の係数 a 、 b 、 c の3つを、返値は判別式の結果 D にすればよいことがわかります。

以上のことから `hanbetsu()` の仕様は以下のようにしましょう。

<関数 hanbetsu() の仕様>

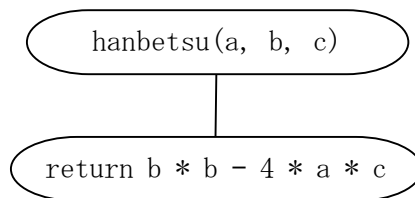
1. 仕様

プロトタイプ	<code>double hanbetsu(double a, double b, double c)</code>
引数	double a double b 二次方程式 $ax^2 + bx + c = 0$ の各項の係数 double c
返値	double 判別式の結果
【解説】 二次方程式の判別式を求めます。	
【例】 <code>D = hanbetsu(2, 0.5, 3);</code> 二次方程式 $2x^2 + 0.5x + 3 = 0$ の判別式の値を D に代入します。	

2. 関数内で使用する変数

なし

3. フローチャート



4. ソースコード

```
double hanbetsu(double a, double b, double c)
{
    return b * b - 4 * a * c;
}
```

<以上>

それでは hanbetsu() の動作確認プログラムを作りましょう。
以下のプログラムを入力し、コンパイル・実行します。

hanbetsu.cpp

```
1: #include <stdio.h>
2:
3: double hanbetsu(double, double, double);
4:
5: main()
6: {
7:     double a, b, c, D;
8:
9:     printf("x の二次方程式 ax^2+bx+c=0 の a, b, c を入力してください。¥n");
10:
11:     printf("a = ");
12:     scanf("%lf", &a);
13:     printf("b = ");
14:     scanf("%lf", &b);
15:     printf("c = ");
16:     scanf("%lf", &c);
17:
18:     printf("方程式は %gx^2+%gx+g=0 ですね。¥n", a, b, c);
19:
20:     D = hanbetsu(a, b, c);
21:     printf("判別式 D=%g より、", D);
22:     if(D < 0.0) {
23:         printf("異なる 2 つの虚数解です。¥n");
24:     } else if(D > 0.0) {
25:         printf("異なる 2 つの実数解です。¥n");
```

```

26:         } else {
27:             printf("重解です。¥n");
28:         }
29:     }
30:
31: double hanbetsu(double a, double b, double c)
32: {
33:     return  b * b - 4 * a * c;
34: }

```

実行例 1

x の二次方程式 $ax^2+bx+c=0$ の a、b、c を入力してください。

a = 1.5

b = 2.2

c = 4

方程式は $1.5x^2+2.2x+4=0$ ですね。

判別式 $D=-19.16$ より、異なる 2 つの虚数解です。

実行例 2

x の二次方程式 $ax^2+bx+c=0$ の a、b、c を入力してください。

a = 1

b = 2

c = 1

方程式は $1x^2+2x+1=0$ ですね。

判別式 $D=0$ より、重解です。

実行例 3

x の二次方程式 $ax^2+bx+c=0$ の a、b、c を入力してください。

a = .3

b = 8.9

c = -2

方程式は $0.3x^2+8.9x-2=0$ ですね。

判別式 $D=81.61$ より、異なる 2 つの実数解です。

どうやら問題なさそうですね。

—%g について—

hanbetsu.cpp の関数 printf() にて %g という書式指定子を使っています。

これは %f 同様 double 型などの実数値を表示させますが %f と異なり 余計な 0 は表示しません。

例) 0.230000 → 0.23

3.3. 階乗を求める

階乗とは以下のような計算のことで！であらわします。

階乗の定義)

$$n! = 1 \times 2 \times 3 \cdots \times n \quad (\text{ただし } n \text{ は自然数})$$

例)

$$4! = 1 \times 2 \times 3 \times 4 = 24$$

$$6! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 = 720$$

この階乗を求める関数 `kaijo()` を作りましょう。

引数は $n!$ の n 、返値はその結果にすればよいことがわかります。

以上のことから `kaijo()` の仕様は以下のようにしましょう。

<関数 `kaijo()` の仕様>

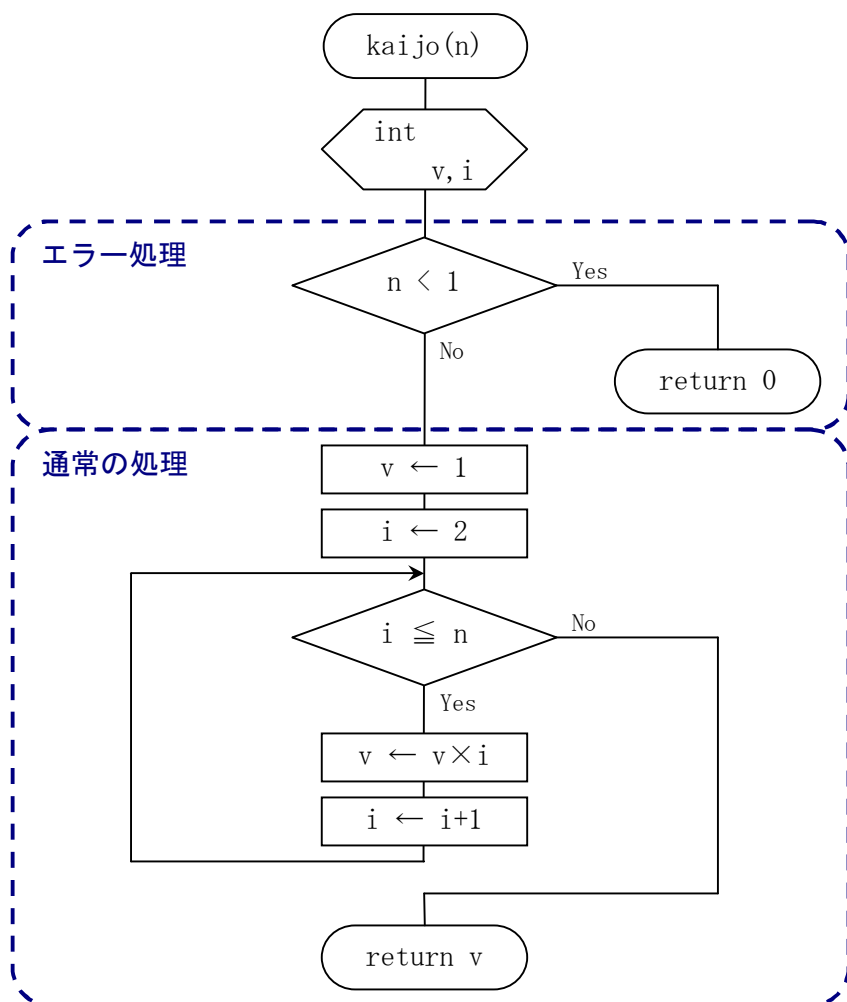
1. 仕様

プロトタイプ	<code>int kaijo(int n)</code>	
引数	<code>int n</code>	階乗の式 $n!$ の n
返値	<code>int</code>	$n!$ ただし $n < 1$ のときは 0 を返す
【解説】 階乗を求めます。		
【例】 <code>v = kaijo(13);</code> 13! を求めに、 <code>v</code> に代入します。		

2. 関数内で使用する変数

変数名	型	概要
v	int	n!の値
i	int	カウンタ

3. フローチャート



n!のnは自然数であるので引数はint型にしていますが、kaijo(-5)のようにnが負の数になってしまう可能性があります。

そこで kaijo() の最初の処理に n が自然数かどうかを if 文で判定し、自然数でなかったら (n が 1 より小さい : $n < 1$) であつたら 0 を返すようにします。

4. ソースコード

```
int kaijo(int n)
{
    int          v, i;

    if(n < 1){
        return 0;
    }

    v = 1;
    for(i = 2; i <= n; i++){
        v *= i;
    }

    return v;
}
```

<以上>

それでは intsaidai() の動作確認プログラムを作りましょう。
以下のプログラムを入力し、コンパイル・実行します。

kaijo.cpp

```
1: #include <stdio.h>
2:
3: int kaijo(int);
4:
5: main()
6: {
7:     int    a;
8:
```

```

9:         printf("a = ");
10:        scanf("%d", &a);
11:
12:        printf("a! = %d! = %d¥n", a, kaijo(a));
13:    }
14:
15:    int kaijo(int n)
16:    {
17:        int          v, i;
18:
19:        if(n < 1){
20:            return 0;
21:        }
22:
23:        v = 1;
24:        for(i = 2; i <= n; i++){
25:            v *= i;
26:        }
27:
28:        return v;
29:    }

```

実行例 1

```

a = 4
a! = 4! = 24

```

実行例 2

```

a = 6
a! = 6! = 720

```

どうやら問題なさそうですね。

4. 演習問題

10-1. `twice2.cpp` を参考に以下の仕様を満たす関数 `half()` を完成させ、その動作を確認するプログラム「`ensyu10-1.cpp`」を作りなさい。

プロトタイプ	<code>double half(double x)</code>	
引数	<code>double x</code>	半分にしたい値
返値	<code>double</code>	<code>x</code> を半分にした値
【解説】 引数を半分にします。		
【例】 <code>a = half(v);</code> <code>v</code> の値を半分にし、それを <code>a</code> に代入します。		

10-2. 以下の仕様を満たす関数 `intpower()` を完成させ、その動作を確認するプログラム「ensyu10-2. cpp」を作りなさい。

プロトタイプ	<code>int intpower(int a, int b)</code>	
引数	<code>int a</code> <code>int b</code>	a^b にそれぞれ対応。 ただし $b \geq 0$ とする。
返値	<code>int</code>	a^b の値
【解説】 整数のべき乗（ただし、指数は必ず 0 以上とする）を求める。 【例】 $x = \text{intpower}(y, 3);$ y^3 を求め、それを x に代入します。		

ヒント：

C 言語にはべき乗の演算子はありません。よって for 文などで掛け算を繰り返すようにしましょう。