J３プログラミング　簡単な再帰のプログラムーー頭の体操ーー

１．再帰について

　再帰という言葉を聞いたことがあるでしょうか。再帰的（recursive）とは、物事が自分自身を含んでいるという意味です。

　身近な例を挙げましょう。皆さんは鏡と鏡を向かい合わせると、何が見えるか知っていますか。無限に続く鏡の列が見えますよね。「物事が自分自身を含んでいる」という
感じは、この「合わせ鏡」を想像すればよいと思います。

　他の例で言うと、マイクロホンをスピーカに近づけたときに出るキーンというういやな音がでます。この現象はハウリングとよばれていて、スピーカの出す音をマイクロホンが「再帰的」に拾ってしまうことで起こる現象です。

　さて、３つめの例は、数学で習った「漸化式」です。

---

　　a(1)=1
　　a(n+1)=a(n) +1

---

a(1)=1,a(2)=2,　...,　a(n)= n, ... という数列を定義する際に、
上のような書き方をすることが出来て、これを漸化式と呼びます。
思い出しましたか？では、もう少し難しい数列で考えてみましょう。

---

　　a(1)=1
　　a(n)= n x a(n-1)

これは、どんな数列を表しているでしょうか。

a(1)=1,a(2)=2 x a(1)= 2, a(3)=3 x a(2) = 6,
となって、結局、a(n)= n!
となります。分かりますか？

```
a(n)    = n x a(n-1)
        = n x (n-1) x a(n-2)
        = ....
        = n x (n-1) x .... x 2 x a(1)
```

という計算になります。

２．C 言語における再帰

C 言語は、関数の再帰的な定義が認められています。すなわち、関数を定義する際に、
定義文の中に自分自身を呼び出しても良いのです。

例題１　以下の関数を
「定義文の中に自分自身を呼び出す方法」
で書け。

関数名：factorial
引数：　int n
戻り値：　n の階乗(n!) すなわち n x (n-1) x ... x 2 x 1
副作用：無し
備考：　0! = 1 とする。

解答例１）

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

int
factorial (int n)
{
  int tmp = 1;
  if (n > 0)
    tmp = n * factorial (n - 1);
  return tmp;
}

main ()
{
  printf ("%d¥n", factorial (3));

}
```

実行結果----------

```
[wada@chiba J3]$ ./a.out
```

6

次のように書いても結果は同じです。解答例１よりスッキリしています。

解答例２）

```
#include <stdio.h>
int
factorial (int n)
{
  if (n > 0)
    return  n * factorial (n - 1);
  else
    return  1;
}

main ()
{
  printf ("%d¥n", factorial (3));

}
```

関数を定義する際に、定義文の中に自分自身（その関数自身）を呼び出す方法は
「再帰的な関数呼び出し」
もしくは
「再帰呼び出し」
という用語で呼ばれています。

３．再帰呼び出しされた関数の動作

先の解答例２を例にして、再帰的な関数呼び出しが行われたプログラムの動作について説明しましょう。

(1)プログラムを実行すると、まず、printf の中の factorial (3)が実行される。

(2)コンピュータは、factorial (3)についての情報を求めて、factorial (3)の定義にもどる。

(3)定義文を読んで、コンピュータは
factorial (3)=3 * factorial (2)
だとわかる。

(4)するとコンピュータは factorial (2)についての情報が欲しくなる。

(5)定義文を読んで、コンピュータは
factorial (2)=2 * factorial (1)
だとわかる。

(6)するとコンピュータは
factorial (1)についての情報が欲しくなる。

(7)
定義文を読んで、コンピュータは
factorial (1)=1 * factorial (0)
だとわかる。

(8)するとコンピュータは
factorial (0)についての情報が欲しくなる。

(9)
定義文を読んで、コンピュータは
factorial (0)=1
だとわかる。

(10)
factorial (0)=1
が分かったので、
factorial (1)=1 * factorial (0)= 1
が分かる。

(11)
factorial (1)=1
が分かったので、
factorial (2)=2 * factorial (1)= 2
が分かる。

(12)
factorial (2)=2
が分かったので、
factorial (3)=3 * factorial (2)= 6
が分かる。

(13) factorial (3)の値がわかったので
printf でその値を表示する。

上の例題１では、階乗を求める関数を、あえて、再帰呼び出しの方法で書いてみました。
これはあくまで練習のためで、ふつうの方法（非再帰的な方法）で書いたほうがわかりやすいプログラムを、あえて、再帰を使って書く必要は在りません。
また、再帰で書いた方が分かりやすいプログラムも、実際あります（再帰を使わないで階乗のプログラムを書き、難しさを比べてみてください）。

例題２.
＊ を指定した個数だけ描く関数を再帰的な定義で書け。

関数名：stars
引数：int n
戻り値：なし
副作用： ＊ を横に n 個描く

解答

```
void stars(int n){
        printf("*");
        if (n>1) stars(n-1);
```

```
}
main(){
        stars(3);
}
```

実行結果

```
[wada@chiba J3]$ ./a.out
***[wada@chiba J3]$
```

上の例題のプログラムが、何故、上のような結果を出すか、分かりますか。何故

```
        if (n>1)
```

になっているか、分かりますか。これを

```
        if (n>0)
```

とすると、結果はどうなるでしょうか。

理解できるまで、先に進まない方が良いです。理解できるまで、検討、質問等してください。理解できたら、次の練習問題をやってみましょう。

練習問題 A
スペースを、指定した個数だけ描く関数を再帰的な定義で書け。

関数名：spaces
引数：int n
戻り値：なし
副作用：スペースを横に n 個描く

練習問題 B
以下のような図を描くプログラムを再帰的手法で書け。

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

関数名：sankaku1

引数：int n
戻り値：なし
副作用：底辺がｎの逆三角形を描く

上の絵は sankaku1(3)を実行した結果である。

練習問題 C
以下の関数を再帰的な定義で書け。

関数名：sankaku2
引数：int n
戻り値：なし
副作用：底辺がｎの三角形を描く

sankaku2(3)を実行させると、以下のような絵を描く。
*
**
***

練習問題 D
以下の関数を非再帰的な方法で書け（stars と spaces は使って良い）。

関数名：sankaku3
引数：int n
戻り値：なし
副作用：以下の三角形をｎ個横に描く。

****
 ***
  **
   *

練習問題E
以下の関数についてff(3)の実行結果を予測してください。

```
void ff(int n){
        if(n>0)
                if (n%2==0) {printf("****¥n"); ff(n-1);}
                else {printf("*¥n"); ff(n-1);}
}
```

練習問題F
以下の関数についてfff(3,4,1)の実行結果を予測してください。

```
void
fff (int n, int k, int s)
{
  if (n > 0)
    {
      fff (n - 1, k, s);
      spaces(k); stars(s); printf("¥n");
      fff (n - 1, s, k);
    }
}
```

練習問題G
以下の関数についてffff(3,4)の実行結果を予測してください。

```
void
ffff (int n, int s)
{
        if (n==0 && s>0) {
                printf("¥n");
                ffff(3,s-1);
        }
        if (n>0 && s>0) {
```

```
                spaces(s);stars(5-s);
                ffff(n-1,s);
        }

}
```

次は少し難しいかも．．．。

例題３．以下の関数を再帰呼び出しを使って書け。

関数名：string_length
引数：　char a[]
戻り値：　　文字列 a[]の長さ
備考：文字列 a[]は書き換えて良い

解答例）

```
#include <stdio.h>

void
shift (char a[])
{
  int n = 0;
  while (a[n] != '¥0')
    {
      a[n] = a[n + 1];
      n++;
    }
}
int
string_length (char a[])
{
  if (a[0] != '¥0')
    {
      shift (a);
      return string_length (a) + 1;
    }
```

```
  else
    return 0;
}

main ()
{
  char a[] = "abcdef";
  printf ("%d¥n", string_length (a));
  printf ("%s¥n", a);
}
```

実行結果----------

```
[wada@chiba J3]$ ./a.out
6

[wada@chiba J3]$
```

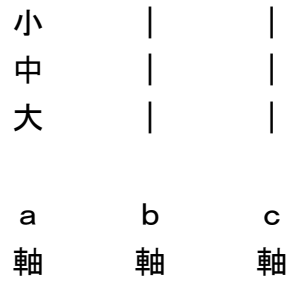
練習問題１
string_length を再帰呼び出しを用いて
作成せよ。但し、受け取った文字列は
書き換えてはいけない。

練習問題２
以下の絵を描く関数を再帰呼び出しを用いて作成せよ。

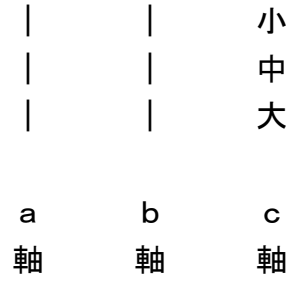
```
***** ***** ***** ***** ***** ***** *****
 ****  ****  ****  ****  ****  ****  ****
  ***   ***   ***   ***   ***   ***   ***
   **    **    **    **    **    **    **
    *     *     *     *     *     *     *
```

練習問題３（チャレンジ）
ハノイの塔のプログラムを、再帰呼び出しを用いて作成せよ。

ハノイの塔：
３つの軸（ａ，ｂ，ｃ）と、３種類の円板（大、中、小）がある。
今、以下の図のようにａ軸に円板が刺さっているとする。



ハノイの塔とは、円板を上手に動かして以下のような状態にする
手順を問う問題である。



円板の動かし方には、以下のような制約がある。
（１）大きい円板が小さい円板より上になってはいけないこと。
（２）一度に一枚ずつしか動かせないこと。

練習問題４（チャレンジ）
以下の関数を再帰呼び出しを用いて作れ

関数名　array_sort
引数　整数型配列名 a とそのサイズ
戻り値　なし
副作用　a に入っている整数データをソートする