■再帰呼び出し禁止
●スマートなプログラミング
プログラミングテクニックとして「再帰呼び出し」という方法があります。
「リカーシブコール(recursive call)」とも呼びます。

関数などの処理の中で自分自身（の関数）を呼び出すという方法で、
抽象的な概念のアルゴリズムを表現するのに便利です。

例えば1からnまでの整数の足し算を行うC言語関数については
再帰呼び出しを使うと以下のようになります。

```
int SumFunc(int n)
{
    if( n<=1 ) return n;
    return(n+SumFunc(n-1));
}
```

再帰呼び出しではない方法で上記の処理を行う場合はforループなどを
使用した処理が必要になります。

例えば、以下のようになります。

```
int SumFunc1(int n)
{
    int sum;
    for(sum=0;n>0;n--)
    {
        sum += n;
    }
    return sum;
}
```

●無限と有限
再帰呼び出しのメリットは以下になります。

(1) 複雑なアルゴリズムを少ないソースコードで表現できる
(2) 概念的には無限のパターンの処理を記述できる

少ないソースコードで任意の引数に応じた結果を返す処理を記述できるため、
ソフトウェアの開発効率としては非常に高いことになります。
同じアルゴリズムの考え方は言語にもCPUにも依存しません。
メリットだけを考えると非常に有効な手法となります。
しかし、再帰呼び出しには以下のデメリットがあります。

(a) 関数コールを繰り返すため、スタック領域を大量に消費する
(b) 例題のような処理の場合には単なるループ処理よりも処理時間を消費する

つまり、メモリ領域や処理時間の制限がある組込みシステム開発では
使ってはいけないテクニックの一つです。
組込みシステムでは常に「有限」を意識する必要があるからです。

●アルゴリズムの移植の問題
なお、自ら新規に開発する場合には再帰呼び出しを
封印することは可能ですが、他のシステム開発で実装された
アルゴリズムを流用するような場合には注意が必要です。

複雑なアルゴリズムが再帰呼び出しで実装される可能性として例えば、
画像処理、圧縮処理、暗号処理などが考えられます。

このようなアルゴリズムの流用を行う場合には、予めスタック使用量が
変動するかどうか、スタックの最大消費量はどれ位になるかなどの
検討を事前に行う必要があります。

●組込み開発流実装例
さて、再帰呼び出しの例として1からnまでの整数の足し算を挙げました。

組込みソフトウェア開発で頻繁に行われる実装例を以下に示します。

```
int SumFunc2(int n)
{
#define MAX \
(sizeof(tbl)/sizeof(int))
#define ERROR -1

    const int tbl[] =
            { 1,3,6 };

    if( n>=1 && n<=MAX )
        return tbl[n-1];
    else
        return ERROR;
}
```

CPUでは計算させずに予め計算結果を格納した配列を確保して、
計算結果は取り出すだけとする方法です。
この方法のメリットは以下になります。

(1) ループ処理を行わないため、処理速度がほぼ一定
(2) スタック消費量が一定

デメリットとしては以下になります。

(a) 処理できるパターンが有限
(b) 処理パターンとROMの使用量が比例して増加する

実際にはそれぞれの手法のメリットとデメリットを考慮して実装すること
になりますが、再帰呼び出しは許容できないケースが多くなるでしょう。