第5章

再帰的アルゴリズム

5.1　再帰の考え方

(1)再帰的定義

ある事象が，自分自身を含んでいたり，それを用いて定義されていることを，**再帰的**(recursive)といいます。たとえば，自然数は，

(a) 1 は自然数である。
(b) ある自然数の直後の整数は自然数である。

と再帰的に定義されますが，このような定義を**再帰的定義**(recursive definition)といいます。

(2)再帰的定義の例

3.7 節の木構造の探索は，再帰的な問題の代表例です。たとえば，もっとも簡単な 2 分木の探索では，

ⅰ）自ノードと等しければ探索完了
ⅱ）左部分木を探索
ⅲ）右部分木を探索

と処理します。左部分木，右部分木も 2 分木ですから再帰的な問題となります。

いわゆる数学的帰納法で定義されるような問題も，再帰的な問題となります。たとえば，階乗値を求める数学的帰納法による定義

ⅰ）$f(0)=1$
ⅱ）$n>0$ に対して $f(n)=n\times f(n-1)$

は，$f(n)$を求めるのに，$f(n-1)$を使っています。 したがって，再帰的な問題として捉えることができます。これを C#のプログラムとして定義すると，たとえば以下のように定義できます。

```
    private long fact(long n)
    {   if(n == 0) return 1;
        else return(n * fact( n -1));
    }
```

(3)再帰的プログラムの動き

再帰的なプログラムは，自分自身を呼び出しているように記述しますが，イメージが沸きにくいので，図 5-1 のように，次々と自分自身のクローンを作って，クローンを呼び出すと考えれば分かりやすいでしょう。

復帰する際は，自分自身を削除して，呼び出した側に関数値を戻すという動きになります。

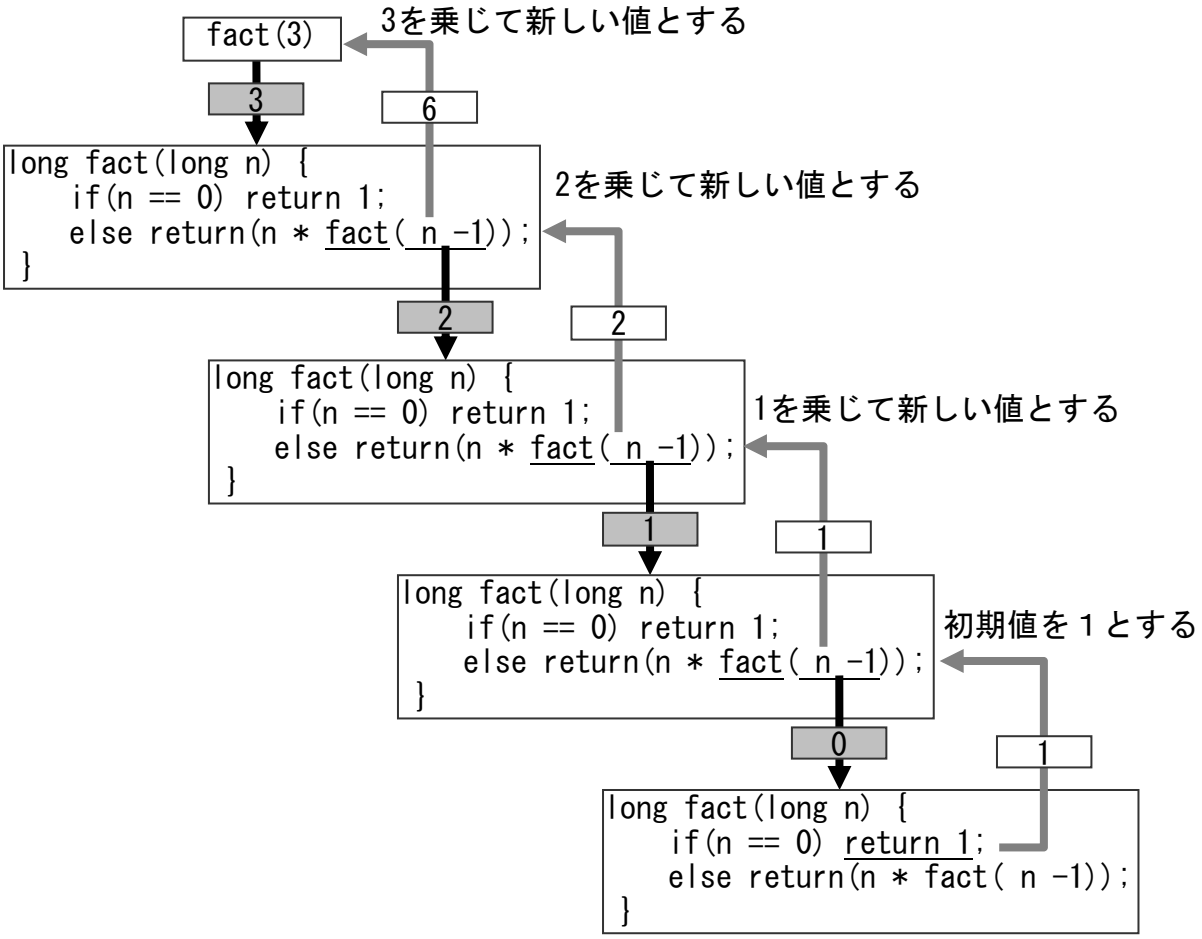
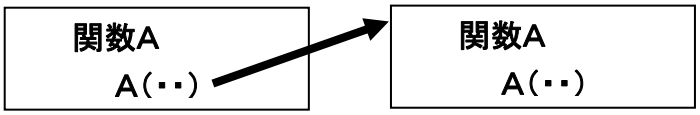


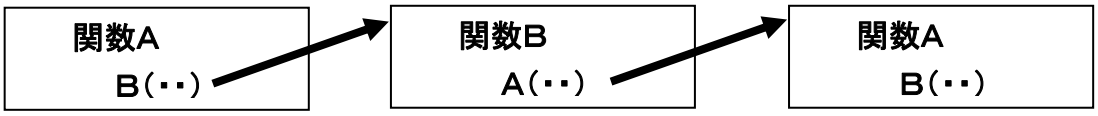
図 5-1　階乗の再帰版の動き

(4)直接的な再帰と間接的な再帰

階乗の計算のように,自分と同じ関数を呼び出すことを**直接的な再帰**(direct recursion)といいます。



次のように，別の関数を呼出し，その関数が自分と同じ関数を呼び出すことを**間接的な再帰**(indirect recursion)といいます。



(5)末尾再帰

図 5-1 に示した階乗の計算のような関数最後で呼び出される再帰呼出しを**末尾再帰**(tail recursion)といいます。

末尾再帰の場合，再帰呼び出しを簡単に除去できます。まず，図 5-1 の再帰呼び出しを観察してみましょう。引数 n の値は，呼び出すたびに 1 ずつ減算されて，最終的に 0 になります。

引数の値が 0 になると，リターン値が 1 になります。リターンするたびに，それぞれの段階の n の値を乗じます。すなわち，初期値を 1 にして，1 から n までを乗じる計算を行っています。これを繰返しで表現すると，以下のようになります。

```
private long fact2(long n)
{
  long r=1;
  for(long i=1;i <= n;i++) r *= i;
  return r;
}
```

(6)真に再帰的な場合

2 回以上再帰呼び出しを行う関数を**真に再帰的**(genuinely recursive)であるといいます。真に再帰的な関数の動きを見るために，次のプログラムを例にとってみましょう。

```
private void test(int n)
{
        if(n <= 0 ) return;
        test(n - 1);
        MessageBox.Show(n.ToString());
        test(n - 2);
}
```

このプログラムは，特に意味のある処理ではありませんが，再帰的な動きを容易に観察するためのプログラムです。引数を 4，すなわち Test(4)として呼び出すと，図 5-2 の実線のように呼び出しが行われ，点線のようにリターンします。

表示は，丸数字の順，すなわち 1，2，3，1，4，1，2 の順に表示されます。以前出てきた，2 分木の場合の通りがけ順と同じような表示順序になっていることに気づきましょう。

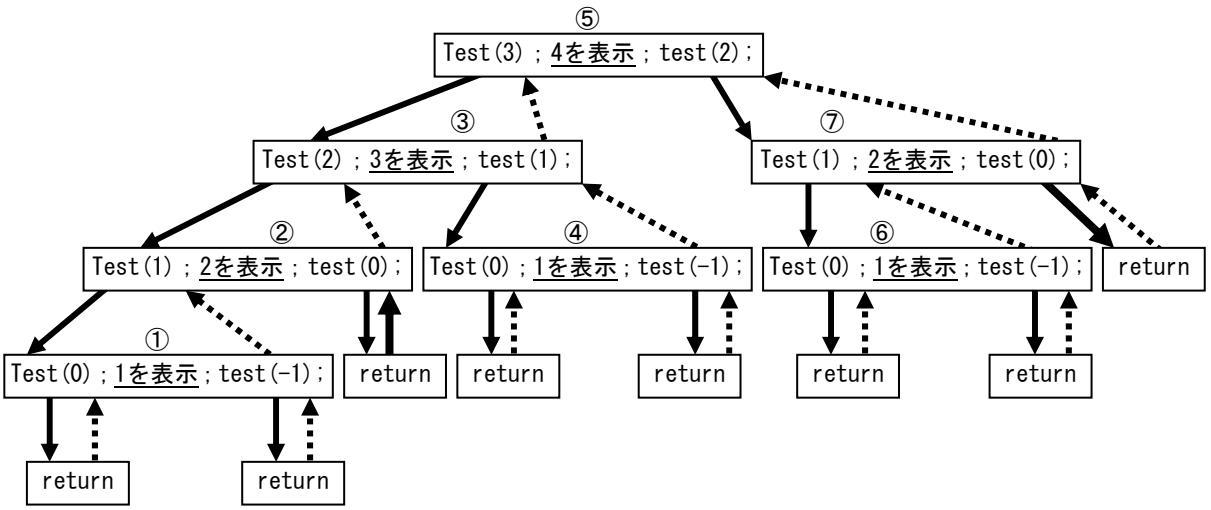


図 5-2　真に再帰的なプログラムの呼出しと復帰

(7)再帰的プログラムのループ化

末尾再帰は，簡単にループ化できますが，以下の下線部は再帰のまま残ってしまいます。

```
private void test(int n)
{ int nn = n;
  while (nn > 0 )
  {  test(nn - 1);
     MessageBox.Show(nn.ToString());
     nn = nn - 2;
  }
}
```

再帰的なプログラムを形式的にループ化するには，スタックを使います。図 5-2 の左側に進むとき，現在の値をスタックに積み，1 ずつ減じ，0 以下になったら，スタックをポップ(リターン処理)してその値を表示します。スタックが空になったら終了します。

右側に進むときは,末尾再帰ですから,n を 2 だけ減じて繰返します。スタックの動きについては，図 5-3 に示します。

```
private void test(int n)
{ int[] stk = new int[20];int p = 0; int nn = n; bool flag = true;
  while(flag)                    // 継続するとき true
  { while (nn > 0 ) { stk[p++] = nn; nn = nn - 1; }
    if(p == 0) flag = false;     // スタック空のとき停止
    else
    {  nn = stk[--p];
       MessageBox.Show(nn.ToString());
       nn = nn - 2;
    }
  }
}
```

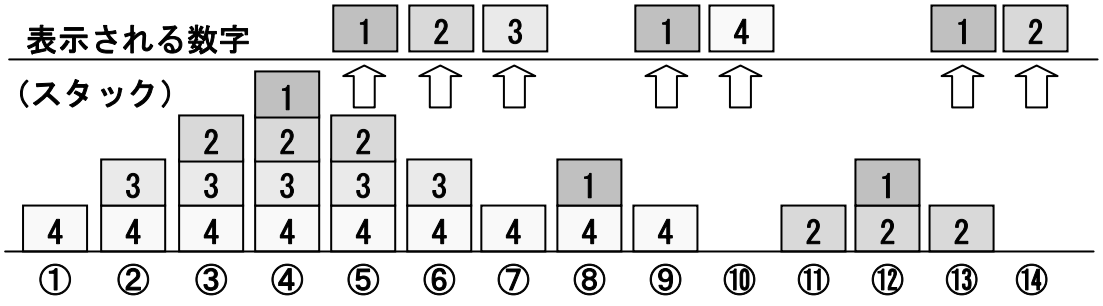


図 5-3　ループ化のためのスタックの動き