16 nの階乗

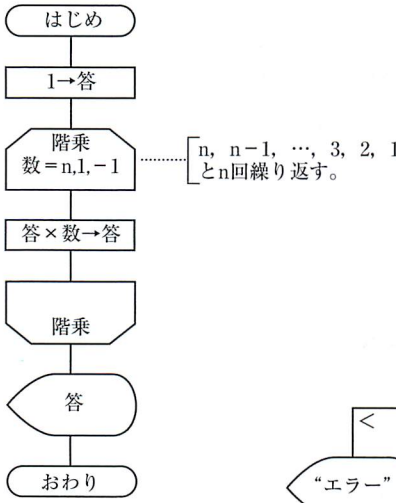
1 アルゴリズムの概要

- nの階乗を求める。

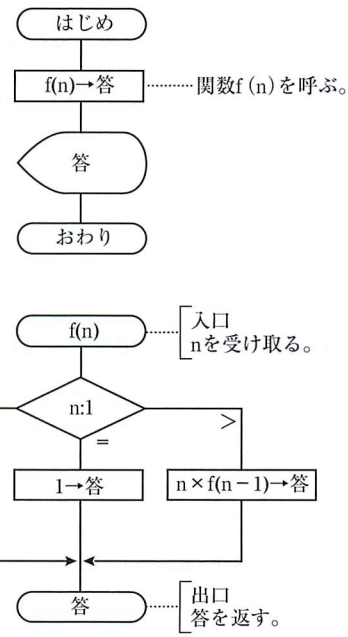
$n!=n\ (n-1)\cdot(n-2)\cdot(n-3)\cdots3\cdot2\cdot1$

2 流れ図

①再帰関数を用いない流れ図



②再帰関数を用いた流れ図



（注）0!は1だが，この流れ図ではn＞0とし，n＝0はエラーにしている。

3 計算の様子

● n＝5で，それぞれトレースしてみた。

①非再帰版

数	答
5	5
4	20
3	60
2	120
1	120

②再帰版

入口　f（5）	5×f（4）を計算
入口　f（4）	4×f（3）を計算
入口　f（3）	3×f（2）を計算
入口　f（2）	2×f（1）を計算
入口　f（1）	f（1）を計算
出口　f（1）＝1	
出口　f（2）＝2	2×f（1）＝2×1
出口　f（3）＝6	3×f（2）＝3×2
出口　f（4）＝24	4×f（3）＝4×6
出口　f（5）＝120	5×f（4）＝5×24

1 nの階乗程度では，再帰を使う利点はない

5!を求めるには，5×4×3×2×1を求めればいいので，1行で書くこともできます。しかし，20!になるとループを用いたほうが簡単に書けます。

1をかける必要もなく，5!なら2×3×4×5でよいわけです。しかし，流れ図①では，n!の式に忠実にnから1までかけていくようにしました。

1回目：答×数＝1×5＝5
2回目：答×数＝5×4＝20
⋮

2 再帰関数は自分自身を呼び出すためにメモリや実行時間を消費する

再帰関数の例として，n!がよく用いられます。再帰関数とは，自分自身を呼び出すことができる関数です。5!＝f（5）を計算するときに，5×f（4）と式を分解して，f（4）を求めるために自分自身を呼び出します。このとき，f（5）を計算している状態（変数の値など）をスタックに退避して呼び出します。このため，再帰を用いないものに比べ，メモリを消費し，関数を呼び出すための時間が余計にかかります。

$$5(f)=5\times\underline{f(4)}$$
$$4\times\underline{f(3)}$$
$$3\times\underline{f(2)}$$
$$2\times\underline{f(1)}$$
$$1$$

実務プログラムでは，再帰関数を用いることは滅多にありませんでした。再帰を用いれば簡単なアルゴリズムでも，再帰を用いずに書くことがほとんどです。しかし，コンピュータの性能向上は著しく，メモリ容量や実行速度の点から従来は再帰関数を用いることができなかったアルゴリズムでも，今後は再帰を利用できる可能性が大きくなっています。

Memo 再帰関数については，n!程度の簡単なものが午前の問題でよく出題されます。午後の出題例は少ないですが，クイックソートなど，再帰に向くアルゴリズムもありますので，よく理解しておきましょう。

17 再帰処理の考え方

1 再帰を使わないn!のプログラム

○ プログラム名： 再帰なしnの階乗
○ 整数型： 答
① • 答 ← Fa(5)
② • 表示命令(答) {答の内容を表示する}

○ 関数名： Fa(n)
○ 整数型： Ans
③ ▲ n ＞ 1
Yes
④ • Ans ← n × Fb(n − 1)
◇ No
⑤ • Ans ← 1
⑥ • 出口(Ans)

○ 関数名： Fb(n)
○ 整数型： Ans
⑦ ▲ n ＞ 1
Yes
⑧ • Ans ← n × Fc(n − 1)
◇ No
⑨ • Ans ← 1
⑩ • 出口(Ans)

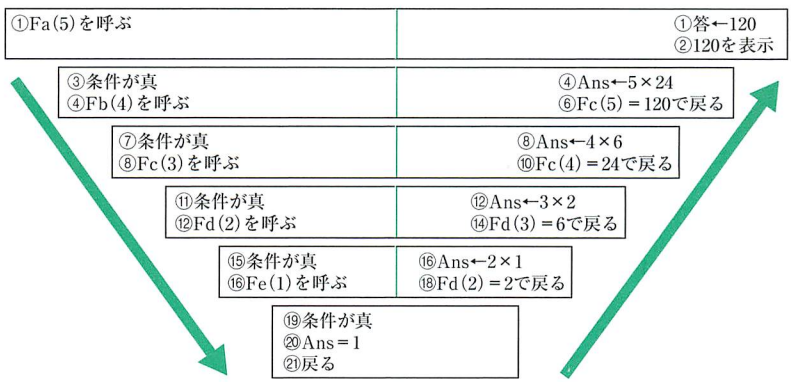
○ 関数名： Fc(n)
○ 整数型： Ans
⑪ ▲ n ＞ 1
Yes
⑫ • Ans ← n × Fd(n − 1)
◇ No
⑬ • Ans ← 1
⑭ • 出口(Ans)

○ 関数名： Fd(n)
○ 整数型： Ans
⑮ ▲ n ＞ 1
Yes
⑯ • Ans ← n × Fe(n − 1)
◇ No
⑰ • Ans ← 1
⑱ • 出口(Ans)

○ 関数名： Fe(n)
○ 整数型： Ans
⑲ ▲ n ＝ 1
Yes
⑳ • Ans ← 1
㉑ • 出口(Ans)

（注）紙面の都合で，関数の宣言を省略している。

2 実行の様子



1 同じ関数を別々に作るのは無駄だけど

再帰関数を使わずに，FaからFeまでの５つの関数を用いて，n!を求めるプログラムです。FaからFdの制御構造はまったく同じです。まず，トレースして，n!が計算されることを確認してください。

2 別々の関数がたくさんあると考えると再帰処理は簡単

同じ制御構造の関数をたくさん作るのは無駄です。そこで，１つだけ作って，何度も利用するのが再帰関数です。再帰関数を使用したプログラムでは，この例のように同じ構造の関数がたくさんある，と考えると簡単です。では，再帰処理でn!を求める擬似言語のプログラムを学習しましょう。

18 再帰関数

1 nの階乗を求める擬似言語プログラム

● 関数 f は，nの階乗を再帰的に求める関数である。

```
宣言部
    ○ プログラム名：  再帰関数のテスト
    ○ 関数：  f(引数)      {引数で与えた数までの階乗を返す}
    ○ 関数：  表示関数(引数, …)
                  {引数に指定した文字列や変数の内容を表示する}
    ○ 整数型：  n

処理部
①   • n ← 5
②   • 答 ← f(n)
③   • 表示関数(答)

宣言部
    ○ 関数名：  f(n)
                  {引数で指定されたnの階乗を返す。
                   引数は値渡しである}
    ○ 整数型：  Ans

処理部
④   • 表示関数("入口 n=", n)
⑤   n < 0
    Yes
⑥       • 表示関数("エラー")
    ◇No
⑦       n > 1
        Yes
⑧           • Ans ← n × f(n − 1)
        ◇No
⑨           • Ans ← 1
⑩   • 表示関数("出口 n=", n)
⑪   • 出口(Ans)      {Ansを返す}
```

(注) 表示関数は省略。説明のために，番号をつけた。

2 実行の様子

M－① • n ← 5 M－② • 答 ← f(n)	f(5) で呼び出す。
5－④ • 表示関数(“入口 n= ”, n) 5－⑤ ▲ n ＜ 0 5－⑦ ▲ n ＞ 1 5－⑧ • Ans ← n × f(n － 1)	「入口 n＝5」を表示 条件式が偽なので，⑦へ 条件式が真なので⑧へ f(4) を呼び出す
4－④ • 表示関数(“入口 n= ”, n) 4－⑤ ▲ n ＜ 0 4－⑦ ▲ n ＞ 1 4－⑧ • Ans ← n × f(n － 1)	「入口 n＝4」を表示 条件式が偽なので，⑦へ 条件式が真なので⑧へ f(3) を呼び出す
3－④ • 表示関数(“入口 n= ”, n) 3－⑤ ▲ n ＜ 0 3－⑦ ▲ n ＞ 1 3－⑧ • Ans ← n × f(n － 1)	「入口 n＝3」を表示 条件式が偽なので，⑦へ 条件式が真なので⑧へ f(2) を呼び出す
2－④ • 表示関数(“入口 n= ”, n) 2－⑤ ▲ n ＜ 0 2－⑦ ▲ n ＞ 1 2－⑧ • Ans ← n × f(n － 1)	「入口 n＝2」を表示 条件式が偽なので，⑦へ 条件式が真なので⑧へ f(1) を呼び出す
1－④ • 表示関数(“入口 n= ”, n) 1－⑤ ▲ n ＜ 0 1－⑦ ▲ n ＞ 1 1－⑨ • Ans ← 1 1－⑩ • 表示関数(“出口 n= ”, n) 1－⑪ • 出口(Ans)	「入口 n＝1」を表示 条件式が偽なので，⑦へ 条件式が偽なので，⑨へ Ansが 1 になる 「出口 n＝1」を表示 2－⑧に戻る
2－⑧ • Ans ← n × f(n － 1) 2－⑩ • 表示関数(“出口 n= ”, n) 2－⑪ • 出口(Ans)	Ans←2×1 「出口 n＝2」を表示 3－⑧に戻る
3－⑧ • Ans ← n × f(n － 1) 3－⑩ • 表示関数(“出口 n= ”, n) 3－⑪ • 出口(Ans)	Ans←3×2 「出口 n＝3」を表示 4－⑧に戻る
4－⑧ • Ans ← n × f(n － 1) 4－⑩ • 表示関数(“出口 n= ”, n) 4－⑪ • 出口(Ans)	Ans←4×6 「出口 n＝4」を表示 5－⑧に戻る
5－⑧ • Ans ← n × f(n － 1) 5－⑩ • 表示関数(“出口 n= ”, n) 5－⑪ • 出口(Ans)	Ans←5×24 「出口 n＝5」を表示 M－②に戻る
M－② • 答 ← f(n) M－③ • 表示関数(答)	答←120 120を表示

(注) nの値，Mはメインの意味。

Memo そろそろ「トレースばかりで疲れる」と思われたかもしれません。擬似言語の問題やC言語の問題で，再帰関数が頻出しています。午後の試験では，擬似言語の再帰関数をトレースさせる問題なども出ています。