

## J 言語プログラミングとは—BASIC、C ユーザに向けて —Explicit と Tacit の 2 つの方式—

西川 利男

J 言語のプログラミングは BASIC や C のユーザから見て分かり難いという。プログラミングとは、相手はコンピュータとは言うものの、所詮は人間が発する言葉であり、文章を綴って行うものである。このような見方から、J のプログラミングを説明してみたいと思う。

題材として、江戸時代の和算家、松永良弼による次のような無限級数の近似式により、円周率  $\pi$  の値を計算するという例を取り上げる。

$$\frac{\pi}{3} = 1 + \frac{1^2}{4 \cdot 6} + \frac{1^2 \cdot 3^2}{4 \cdot 6 \cdot 8 \cdot 10} + \frac{1^2 \cdot 3^2 \cdot 5^2}{4 \cdot 6 \cdot 8 \cdot 10 \cdot 12 \cdot 14} + \dots$$

この近似式を使って松永は小数点以下 49 桁まで正しく求めた、という。[1]

[1]加藤文元「物語 数学の歴史」p. 112 中公新書(2007).

### 1. まずは、J を対話的に使用する

私の個人的考えでは、J の第一の特徴は、通常のプログラミング言語にない対話型での使用である、と思う。これは、J では値の宣言が不要で、かつ特別のエディターのような環境がいらぬことによる。これは、普通の文章の間に、J のコードを混ぜて実行できることを意味する。C ではもちろん、BASIC でもこんなことはできない。

松永の式にしたがって、対話的に始めてみよう。J では、電卓のように入力して RETURN キーを押すと、次の行に結果がすぐ返される。

級数の各項を 0 から始める。つまり、初項の 1 を 0 番目とする。1 番目は後にして、2 番目の項について、やってみる。

分子の部分は  $1^2 * 3^2$ 、つまり  $1 * 9$  になるが、J では 2 乗は次のようになる。

```
(*: 1), (*: 3)
```

```
1, 9
```

これを掛けて、分子とする。

```
(*: 1) * (*: 3)
```

```
9
```

ここで、引数が 2 つ以上の値のときを考えると、次のようにした方がよい。\*/ は引数の間に演算\*を入れて計算する。

```
*/ *: 1, 3
```

```
9
```

分母の部分も、同様にして、求める。

```
4 * 6 * 8 * 10
```

```
1920
```

```
*/ 4, 6, 8, 10
```

```
1920
```

そして、分子を分母で割って、2 番目の項が得られる。

```
(*: */ 1, 3) % (*: 4, 6, 8, 10)
```

0.0046875

1 番目の項は、もっと簡単に得られる。

$(\pi * 1) \% (* / 4, 6)$

0.0416667

上の結果を使って、松永の式にしたがい、0 項から 2 項目までの和をとり、3 倍すると、大まかな  $\pi$  の値が計算される。

$3 * 1 + ((\pi * 1) \% (* / 4, 6)) + ((\pi * 1, 3) \% (* / 4, 6, 8, 10))$

3.13906

さらに、3 番目の項も計算して、上の結果に加えると、もっと良い近似値が得られる。

$3 * 1 + ((\pi * 1) \% (* / 4, 6)) + ((\pi * 1, 3) \% (* / 4, 6, 8, 10))$   
 $+ ((\pi * 1, 3, 5) \% (* / 4, 6, 8, 10, 12, 14))$

3.14116

以上のようにして、J の対話的使用で、 $\pi$  の値が計算されることが分かった。

## 2. J の Explicit プログラミング

次に、もう少し一般的に任意の値に対して計算できるように J のプログラムを作る。ここでの J のプログラムは、あらわに引数をとる関数 (J では動詞) を定義するもので Explicit (明示的) 方法と呼ばれる。

例えば、松永の式で 3 番目の分数の項について考える。

分子では

1, 3, 5

分母では

4, 6, 8, 10, 12, 14

という数の並びを得ることがポイントとなる。

これから、J のプログラムを作ることになるが、J では動詞 (= 関数) の定義と呼ぶ。そして、配列処理言語としての J の本領が発揮される。つまり BASIC や C のように繰り返し処理にループ構造を必要としないでプログラムが作られる。

分子については、 $2n + 1$  の奇数であるので、次のようにプログラムは作られる。ここで J の動詞  $i.$  は 0 からの整数列を生成する。

```
bunsi =: 3 : 0
```

```
1 + 2 * i. y
```

```
)
```

実行すると次のようになる。

```
bunsi 2
```

```
1 3
```

```
bunsi 3
```

```
1 3 5
```

分母は少し工夫を要するが、次のようにプログラムされる。

```
bunbo =: 3 : 0
```

```
4 + 2 * i. 2 * y
)
```

実行すると、次のようになる。

```
bunbo 2
4 6 8 10
bunbo 3
4 6 8 10 12 14
```

これらの動詞を使って、各項の分数は、次のようになる。ここで、Jの動詞\*/は右引数の値をすべて掛けた値を返す。

```
term =: 3 : 0
(*/*: buns i y) % (*/*: bunbo y)
)
```

最初の項(=1)だけは、特別に扱い、これを考慮した動詞 termp は次のようになる。

```
termp =: 3 : 0
if. 1 = y
do. 1
else. term y
end.
```

これらを、実行してみる。

```
termp"0 0, 1, 2, 3
1 0.0416667 0.0046875 0.000697545
```

上のプログラムでは、動詞 termp は本来、単一の値(スカラー)を引数として動作する。ところが、ランク 0 を付加することにより、複数の値(ベクトル)を引数としてそれぞれの結果を得ることが出来る。これもJの強力な機能である。

```
+/* termp"0 0, 1, 2, 3
1.04705
```

以上の動詞を使って、 $\pi$  の値は次のようにして得られる。

```
3 * +/* termp"0 i. 4
3.14116
```

最終的な  $\pi$  の値を計算するプログラムは、次のようになる。

```
pi_calc =: 3 : 0
3 * +/* termp"0 i. y
)
```

実行例は次のようになる。

```
pi_calc 4
3.14116
```

桁数が多くなった小数計算では結果が丸められてしまう。Jではx:を作用させ、すべて分数計算で行い高精度の小数計算を可能にしている。しかし、最終結果を小数で表示するには、20j18 " というフォーマットで、小数点以下 18 桁、全体 20 桁として値が示される。

```
20j18 " : x: pi_calc 20
3.141592653589928339
```

### 3. JのTacitプログラミング

Jは関数型言語であり、それを生かしたTacit(暗黙)型と呼ばれる、引数をあらわに記さないプログラム記法がある。ここでは2, 3の注意点のみを記す。

計算処理のため、動詞(=関数)の連続的使用が行われるが、接続詞@または@:で次々とつなげる。このとき、いくつかの注意が必要である。

(1) JのコードのParsing(読み込み解釈)は一行の文頭から行われる。しかし、関数型言語の特性として実行は文の後ろからなされる。このため、実行順序を指定する多くのかっこが必要となる。

(2) 関数がとる引数に応じて、接続詞@と@:の使い分けが必要である。

- ・ 引数がスカラー(単一の値)でそれぞれに関数を作用させるときは@で結合。
- ・ 引数として、ベクトル(複数の値の集合)をとるときは@:で結合する。

例えば、次のように演算実行される。

```
+/@:(+:@(i. 3))
=> +/@:(+:@(0 1 2))
=> +/@:(0 2 4)
=> 0 + 2 + 4
=> 6
```

このように、極めてコンパクトにプログラムが出来る。

以下に動詞のTacit定義と実行のようすを示す。先のExplicitプログラムと同じπ計算の処理であるので、コーディングを比較してみてほしい。

#### ・ 定義

ss =: 1&,@(>:@(+:@(>:@(i.@(<:)))))) NB. 分子を返す動詞

rr =: 4&+@(+:@(i.@(+:))) NB. 分母を返す動詞

#### ・ 実行

```
ss 1
1
rr 1
4 6
ss 2
1 3
rr 2
4 6 8 10
ss 3
1 3 5
rr 3
4 6 8 10 12 14
```

#### ・ 定義

srn =: (\*/@: (\*:@ss)) % (\*/@: rr) NB. n項目の分数を計算

sr =: srn`1:@. (0&=) NB. 0項目(=1)の場合を考慮する

• 実行

```
sr 0
1
sr 1
0.0416667
sr 2
0.0046875
sr 3
0.000697545
sr"(0) i.4
1 0.0416667 0.0046875 0.000697545
+ sr"(0) i.4
1.04705
3 * + sr"(0) i.4
3.14116
```

• 定義

```
pi_value =: 3&*@(+/@: sr"(0)) NB. 最終的な  $\pi$  計算の動詞
```

• 実行

```
pi_value i.10
3.14159
pi_value i.20
3.14159
x: pi_value i.20
1535207845492r488671834567
20j18 ": x: pi_value i.20
3.141592653589928339
```

NB. pi calc. by series 2016/11/1=====

NB. 松永良弼による無限級数近似、加藤「物語数学の歴史」p.112

```
ss =: 1&,@(>:@(+:@(>:@(i.@(<:))))))
```

```
rr =: 4&+@(+:@(i. @(+:)))
```

```
srn =: (*/@: (*:@ss)) % (*/@: rr)
```

```
sr =: srn`1:@. (0&=)
```

```
pi_value =: 3&*@(+/@: sr"(0))
```

#### 4. 終わりに

以上のような  $\pi$  計算で、Jのプログラミングは、いかがであろうか。

J言語には、BASICやCなどには見られないループ不要の配列計算など、いろいろな機能があり、やや取り付き難いように見えるが、決して難しいものではない。

J Quick Referenceなどを必要に応じて参照して、トライ・アンド・エラーにより試して見て、自分のものにしていただきたい。それだけの価値のあるものである。

