3...の豆旅

3.1 円周率

円周率は不思議なもので、古くから多くの人々を魅了してきた。いちばん簡単に円周率を表す数は 3 である。また、もっとも効率のよい近似値なら 3.14 というところだろうか。円周率は、小数点以下に不規則な数が並ぶので、昔から有効桁数を高める競争が行われてきた。コンピュータが発達した現代でも、それは続けられている。

円周率を求める古典的な方法は、アルキメデス[1)]が考案した。円に内接する正 n 角形の周長と円に外接する正 n 角形の周長を計算し、それにはさまれた値を円周率とすることである。はさまれた値といっても、ある範囲にはさまれてるわけだから、値が確定しない。したがって、内接正 n 角形の周長と外接正 n 角形の周長で、一致している部分までが円周率の正しい値を示していることになる。

これらの計算をするには、三角比に関する知識があるとよいのだが、ここではアルキメデスの方法で円周率の近似をすることが目的ではない。期待した諸君には申し訳ないが省略させてもらおう。

円周率はギリシア文字 π で代用されている。円周率が通常の分数で表せないので当然の処置だろう。通常の分数で表せないけれど

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots \tag{1}$$

といった、**無限級数**で表すことは可能だ。これはグレゴリー[2)]もしくはライプニッツ[3)]の功績だ。式が $\pi = \ldots$ ではなく $\pi/4 = \ldots$ と書いてあるのは、その求め方に由来しているが、詳しいことは別の書物を参考にしてもらいたい。右辺を無限に計算することができるなら、その結果はぴったり円周率の値となる。しかし、これは計算機向きの式ではない。なぜなら**収束**が非常に遅いからだ。収束が遅いことは、計算機の処理速度がいかに速くても致命的なものである。

1) シラクサのアルキメデス (287?B.C.–212B.C.) : 古代ギリシアの数学・物理学者。
2) ジェームス・グレゴリー (1638–1675) : イギリスの数学者・発明家。
3) ゴットフリード・ウィルヘルム・ライプニッツ (1646–1716) : ドイツの哲学・数学者。

収束が遅いことは式を見ているだけでも理解できると思う。たとえば級数を 5 億項先まで加えても、分母は 10 億程度の大きさである。これでは小数点以下 8 桁の精度にしかならない。

しかし悲観ばかりしていても仕方ない。収束が遅いことを承知の上で、この方法で **Java** に計算させてみよう。

programming list [Greg.java]

```
 1: public class Greg {
 2:
 3:     public static void main(String[] args) {
 4:         int sgn = 1;
 5:         double p = 0.0;
 6:
 7:         for(int n = 1; n < 30000; n += 2) {
 8:             p = p + (double) sgn / n;
 9:             sgn = -sgn;
10:         }
11:         System.out.println(4 * p);
12:     }
13: }
```

はじめに 4:行目の `sgn` という変数について話しておこう。`sgn` は符号を変化させるために用意した変数である。こんなものが必要になるのは、(1) が交互に足したり引いたりの計算をしているためだ。はじめは正の数から足しているので `sgn` には `1` が代入されている。しかし、符号を転換する目的なら別の方法もある。

5:行目の変数 `p` は、π の値を格納するために用意している。小数を扱うので倍精度の浮動小数点型だ。宣言と同時に `0.0` を代入しているが、この初期化は非常に大切なことだ。

TRY! 試しに 5:行目を `double p;` として実行してみよ。

とりあえず分母が 30000 ぐらいのところまで和を取るつもりで、7:行目の `for` 構文は `n < 30000` としておいた。いままでなら `<=` を使うところだが、大雑把な項数でかまわないと考えていることと、`n += 2` から分かるように、`n` は 2 ずつ増えるので `<` でも `<=` でも同じ結果になるからだ。ほとんどの場合 `<=` を使う方が望ましい。ただ、ここで `<=` を使うなら、`n <= 29999` や `n <= 30001` とするのが違和感のない書き方だろうか。それよりも、キリのよい数が見やすいので `n < 30000` にしている。

ところで、さらっと『n += 2 から分かるように、n は 2 ずつ増える』と述べたが、n += 2 は n = n + 2 を省略して書いたものである。**C** や **Java** ではよく用いられるが、言語によってはそう書けないものもある。分かりにくければ n = n + 2 と書く方がよいだろう。ついでに、i++ は i = i + 1 だったね。

8:行目と 9:行目がやっていることが理解できるだろうか。(double) の説明はあとに回すとして、8:行目で p に sgn / n を加えて、次は sgn / n の符号が逆になるように 9:行目で sgn の符号を変化させているのだが…。そこで (double) である。本来であれば p = p + sgn / n; で済みそうなものを、sgn でなく (double) sgn を用いている。一体 (double) は何の役に立っているのだろうか。

EX. (double) が使われているのは、sgn をそのまま計算に使うのでは不都合が生じるからだが、その不都合とは何か考えてみよ。

(double) の役目は、不都合を解決するために必要なのだ。8:行目において n, sgn は int 型の変数であるから、sgn / n の計算結果は当然のごとく整数値で返ってきてしまう。それは困る。しかし、**Java** は精度の違う変数どうしの演算では、精度の高い方に合わせてくれる。そこで (double) sgn とすることで一時的に sgn を double 型の変数にしているのだ。このような操作は**キャスト**と呼ばれている。

TRY! 試しに 8:行目の (double) をなくして実行してみよ。

TRY! (double) sgn / n を (double) (sgn / n) にすると結果がどうなるか予想し、予想が正しいかどうか確かめてみよ。

さて、以上の計算で求めたのは $\pi/4$ の値である。それを π の値として表示させるには、11:行目の System.out.println; 文で 4 * p としなければならない。