計算機プログラミング I (4)

繰り返しによる問題解決

2000 年 10 月 31 日 増原 英彦

1 条件を満たすまでの繰り返し

「正の整数 p が素数かどうか」という問題を考える。p が素数であるとは、p の約数が 1 と p の 2 つしかないような数のこと、つまり、$2, \ldots, (p-1)$ では p を割り切ることができないような数のことである。

このような場合、2 から順に p の約数かどうかを調べてゆき、約数が見つかった場合は繰り返しを中断して「素数でない」と判定し、p まで到達した場合は「素数である」と判定すればよい。プログラムとしては下のようになる。繰り返しを続ける条件が 2 種類あることに注意:

```
1  int testNumber = 10000019;        // 素数かも知れない数
2  // 除数で割り切れず、除数が「素数かも知れない数」より小さい間繰り返す
3  for (int divisor = 2;             // 除数
4       (testNumber % divisor) != 0 && divisor < testNumber;
5       divisor = divisor + 1)
6    ; //繰り返しの中身は空なので、セミコロンを書く
7
8  // 除数が「素数かも知れない数」に等しくなったときは素数
9  if (testNumber == divisor)
10   System.out.println(testNumber + "は素数です。");
```

練習 **4-1:** (ユークリッドの互除法)　整数 a, b の最大公約数 (GCD) は、a, b に共通する約数のうち最大ものである。GCD を計算する方法として、ユークリッドが考えたものに、

> a を b で割った余りを r としたとき、a, b の GCD と b, r の GCD は等しい

というものがある。これを用いて 2 つの整数の GCD を計算するプログラムを作れ。(注: a を b で割った余りが 0 ならば b が GCD になる。)

2 集計

最も基本的な使われ方が「全ての場合について計算をし、その計算結果を集計する」というもの。例えば、合計を求める (試験の成績・数値積分) ものや最大・最小値を求めるといったものが代表的。このような問題には、下のような構造のプログラムを書くのが基本になる。

```
集計値を覚える変数を宣言
for (...) { // 全ての場合について繰り返し
    各場合についての値を計算
    集計値を更新
}
```

例えば、数値積分であれば、以下のようになる[1]。

[1] `Math.sin(x)` は x の正弦を求める関数。

```
1  double PI = 3.141592; // 円周率
2  double xStart = 0, xEnd = PI; // 積分区間 [0, π ]
3  int intervals = 10000; // 分割数
4  double delta = (xEnd-xStart)/intervals; // 微小区間の幅
5  double sum = 0;   // (積分値/delta) を覚える変数
6  for (double x=xStart; x<xEnd; x=x+delta)
7    sum = sum + Math.sin(x); // 集計: 和を計算
8  double integral = sum * delta; // 積分値
```

集計値を更新する際に、和や積を求めるかわりに「(これまでの集計値) と (今回計算した値) の大きい方」をとるようにすると、最大値を求めるプログラムも容易に作れるだろう。

練習 **4-2:** (整数の **2** 乗和)　$1 \sim 1000$ までの整数の 2 乗の和を求めよ。

練習 **4-3:** (約数の数)

1. ある数 n の約数の個数を数えるプログラムを作れ。
2. $1 \sim 1000$ までの中で、約数の個数が最大となる数を表示するプログラムを作れ。

練習 **4-4:** (円周率)　円周率の近似値は以下の式[2]で求められることが知られている:

$$\frac{\pi}{4} = \frac{2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdot 8 \cdots}{3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdot 7 \cdots} = \prod_{n=1,2,\ldots} \frac{2n \cdot (2n+2)}{(2n+1)^2}$$

これを用いて、$n = 1, \ldots, 1000$ とした場合の円周率の近似値を計算せよ。

3 Generate & test

「ある条件を満たすものを見つけよ」というパズル的な問題は「全ての場合を作り、それぞれが条件を満たしているかを調べる」という方法で解くのが基本である。例えばペントミノパズルであれば、(1) ありとあらゆるタイルの置き方を作ってみて、(2) その置き方が隙間のない長方形になっているかを調べる、という方法で解くことができる。(多くの場合、「ありとあらゆる置き方」を作り出していると可能性が多過ぎるので、「絶対に長方形が作れない置き方」になるようなものは途中で捨てるような工夫をする。)

このような問題では「全ての場合」を作り出すのに多義の繰り返しを使うのが基本である。例えば、「$x^2 + y^2 = z^2$ を満たす整数 x, y, z の組を見つける」問題は、3 重の繰り返しをすることで全ての x, y, z の組み合わせを作り、それが $x^2 + y^2 = z^2$ を満たしているかを調べればよい[3]ので下のようになる:

```
1  // 1 から maxNumber までの整数の組について調べる
2  for (x = 1; x <= maxNumber; x = x + 1)
3    for (y = 1; y <= maxNumber; y = y + 1)
4      for (z = 1; z <= maxNumber; z = z + 1)
5        if ( x*x + y*y == z*z ) // 条件を満たしているか?
6          System.out.println("("+x+","+y+","+z+")");
```

[2] Abelson and Sussman, *"Structure and Interpretation of Computer Programs"*, MIT Press, 1985. p.56 より引用。

[3] これは説明のために愚直に書いたものなので、maxNumber を少し大きな値にするととんでもなく時間がかかる。明らかに条件を満たさない組み合わせを作らないように工夫をするのが普通。

練習 **4-5: (10 を作る)** 「4 桁の数から 1 桁の数を 4 つ作り、それらを加減乗除した結果が 10 になる計算を探す」という問題を考える。例えば 6679 であれば、$6+6+7-9=10$ という具合である。

1. 簡単のために足し算だけで 10 になる場合を全て表示させるプログラムを作れ。
2. 足し算と引き算だけで 10 になる場合を全て表示させるプログラムを作れ。 10 進数で書いたときに '$abcd$' なら $(a+b+c+d), (-a+b+c+d), (a-b+c+d), (a+b-c+d), (a+b+c-d), (-a-b+c+d), (-a+b-c+d), (a-b-c+d), (-a+b+c-d), (a-b+c-d), (a+b-c-d)$ のどれかが 10 になればよい。

練習 **4-6: (覆面算)** "$SEND+MORE=MONEY$" という式があったとき、S, E, N, D, M, O, R, Y にあてはまる 1 桁目の数の組み合わせを探すプログラムを作れ。ただし、それぞれの文字に使われる数は重複せず、一番上の桁は 0 でないとする。

4 漸近解法

繰り返しを使う問題解決の中には、適当な値から出発して、序々に正確な値へと近付けてゆく漸近的な方法がある。例えば、電話帳から名前を探す場合などがよい例だろう。

このような問題では、「適当な値」と「正しい値」の大小関係などのように、「正しい値」がおよそどのあたりにあるかという情報を使う。

例題として

> 人間に 0 ~ 100 の間の数 x を 1 つ考えさせ、『x と比べて n は同じ/大きい/小さいか?』という質問だけで数を推定する

というものを考えよう。これを漸近的に解くには、人間が考えた数の存在する範囲を序々に狭くしてゆく方法を使う。

最初、解は区間 $[0, 100]$ の間にある。ここで「x と比べて 50 は同じ/大きい/小さいか?」という情報があれば、解が $[0, 49]$ あるいは $[51, 100]$ の間のどちらかにあること (あるいは x が 50 であったこと) が分かる。さらに質問を続ければ、その度に解のある区間の幅が半分になってゆくので、いずれ正解を見つけることができる。

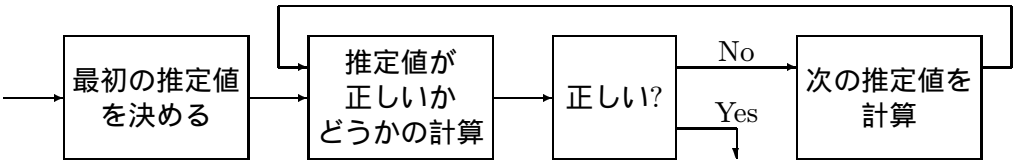
さて、これを繰り返しを使ったプログラムとするにはどうするのだろうか。計算の順序としては、次のようになるだろう。

> 最初の推定値を決める → 推定値が正しいかどうかの判定 → (正しくないので) 推定値を改良する → 推定値が正しいかどうかの判定 → ... → (正しくないので) 推定値を改良する → 推定値が正しいかどうかの判定 → (正しくないので) 推定値を改良する → 推定値が正しいかどうかの判定 → (正しかったので) 推定値 = 解だとして続く計算をする

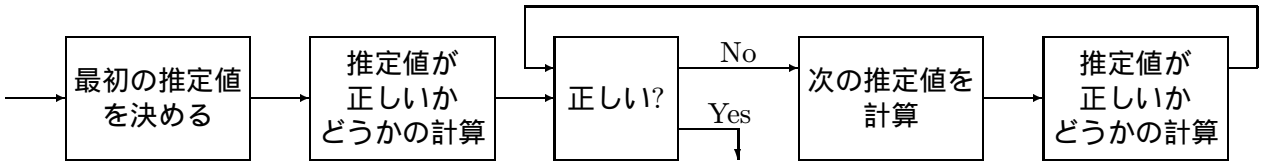
これを繰り返しで書くには、次のような構造でよいだろう:

```
最初の推定値を決める
while（推定値が正しいかどうかの判定）
    推定値を改良する // 正しくないので
// 推定値 = 解と分かる
```

しかし、「推定値が正しいかどうかの判定」が簡単でなく、いくつかの文によって書かれる場合も多い。そのような場合、次のような構造の繰り返しをしたくなる:



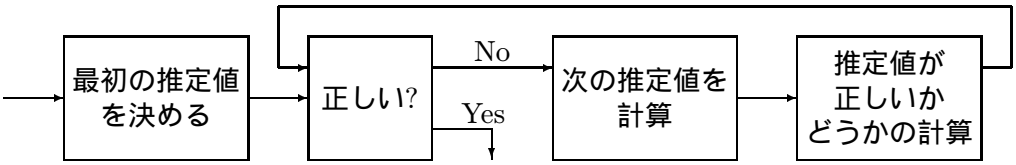
しかし、while 文では、「繰り返しの先頭で判定をする」ことになっているため、上のようなものは書きにくい。一つの方法は、下のように「推定値が正しいかどうかの計算」を 2 箇所に書く方法である。



つまり、プログラムの構造としては以下のようになる:

```
最初の推定値を決める
推定値が正しいかどうかの計算
while（正しい?）{
  次の推定値を計算
  推定値が正しいかどうかの計算
}
```

もう 1 つの方法は、最初に与える推定値の与え方を工夫して、1 回目の「次の推定値の計算」後から期待する計算を行わせるようにするものである。そうすると、下のように簡単な構造にすることができる。



これをプログラムにすると、下のような構造になる:

```
最初の推定値を決める
推定値は「正しくない」とする
while（正しくない?）{
  次の推定値を計算
  推定値が正しいかどうかを計算する
}
```

「数当てゲーム」に戻ると、推定値の正しさは、ターミナルから人間の入力する答えを読みとって決めているため、一つの条件式では調べられない。そこで、「最初の推定値」を -1、それに答して「小さい」(正しくない) と人間が答えたとして繰り返しを始めることで、すっきりとしたプログラムになっている (図 1)。(紙面の都合で一部略している。完全なプログラムは web ページに掲載する。)

練習 **4-7:** (逆数)　ある正の数 x の逆数 $(1/x)$ を加減乗算だけで (割り算を使わずに) 求める方法を考え、プログラムを作れ。(ヒント: 推測値 $g(>0)$ が $1/x$ より小さいかどうかは $g < 1/x \Leftrightarrow gx < 1$ で調べられる。)

5　第 1 回 レポート課題

以下の 2 問両方についてプログラムを作成し、説明・実行結果とともに提出せよ。ただし、「(optional)」と書かれた小問は余力があるもののみ解けばよい。

```
1  // 人間に大小を教えてもらうための番号
2  int GUESS_IS_CORRECT=0, GUESS_IS_SMALLER=1, GUESS_IS_GREATER=2;
3  //  0:正解              1:推定値の方が小さい  2:推定値の方が大きい
4  int minimum = 0, maximum = 100;          // 推定範囲
5
6  (略: ターミナルから答えを受けるための準備)
7
8  int guess = -1;                   // 最初の推定値
9  int answer = GUESS_IS_SMALLER;  // それに対する答え
10
11 while (answer != GUESS_IS_CORRECT) {// 推定が正しくない間くりかえす
12   //推定範囲の改良
13   if (answer == GUESS_IS_SMALLER)// guess < 正解 の場合
14     minimum = guess + 1;         // 正解 は [guess+1..maximum] にある
15   else if (answer == GUESS_IS_GREATER) // 正解 < gusess の場合
16     maximum = guess - 1;         // 正解 は [minimun..guess-1] にある
17
18   guess = (minimum + maximum) / 2; // 区間の中央値を次の推定値とする
19
20   System.out.print("推定値: "+ guess); // 推定値の正しさを質問する
21   answer = (略: ターミーナルから整数値を 1 つ読みとる);
22 }
```

図 1: 数当てゲームのプログラム (概要)

提出方法: レポートは教育用計算機システムの電子メールによって以下の提出用アドレスに送ること。正しく受け付けられた場合、確認のメッセージが折り返し送られる。もし上手く受け付けられない/確認メッセージ来ない場合は相談用のアドレスに事情を説明すること。

提出用: masuhara-cp1-report1@lecture.ecc.u-tokyo.ac.jp
相談用: masuhara-cp1@lecture.ecc.u-tokyo.ac.jp

期限: 2000 年 11 月 8 日 (水) — 特別な事情がある場合を除いて、遅れて提出することは認めない。

配点: 全体の成績の 25 点分に相当する。

注意: なお、説明や実行結果のないレポートは採点しない。プログラムおよびレポートは、必ず自分で作成すること。他人のレポート (プログラムを含む) を写したものがあった場合は、写した者、写された者の両者ともに 0 点とすることがある。

課題 1-1 (コラッツの問題)

コラッツの問題とは、「任意の正の整数について下のような繰り返しを行ったとき、必ず $4 \to 2 \to 1$ の繰り返しになるかどうか」というものである。

操作:
ある数が
・偶数の場合: 2 で割る
・奇数の場合: 3 倍して 1 を足す
という操作を繰り返す。

ここで、「任意の数について上のようになる」ことは数学的な証明でするしかないので、コンピュータが扱える範囲の数についてのみ確めてみることにする。

1. ある数が与えられたときに、上の操作を繰り返し行い、数が 1 になったら停止するプログラムを作れ。繰り返しの間、数がどう変化するかも表示させよ。

   ある数を 106239 として実際に計算させてみよ。

2. (optional) 上のプログラムを拡張して、1 以上の整数について停止することを順に確め、可能な限り大きな整数について確めるプログラムを作れ。

   「3 倍して 1 を足す」ときに桁あふれが起きる可能性に注意せよ。(注: Java 言語に整数の型は何種類か用意されているが、好きなものを用いてよい。)

3. (optional) 「$4 \to 2 \to 1$ の繰り返しになる」までに要する操作の回数を数え、(見つけられた範囲で) 最長の回数が必要な数を探せ。

課題 1-2 (平方根)

ある数 x の平方根を漸近解法によって求めよ。

1. 正の推測値 g の方が x の平方根より小さい場合、$g < \sqrt{x} \Leftrightarrow g^2 < x$ であることを利用して、解のある範囲を狭めてゆき、$|g^2 - x| < \epsilon$ となるまで繰り返すことで求めるプログラムを作り、3 の平方根を求めよ。

   ただし、ϵ は適当に小さな値とする。また、ある数`x` の絶対値は`Math.abs(x)` という式によって求められる。

2. (optional) ニュートン法を用いて平方根を求める場合、推測値 g から次の推測値を $g/2 + x/2g$ としてゆけば解に近付いていくことが知られている。これを用いて $|g^2 - x| < \epsilon$ となるまで繰り返すことで求めるプログラムを作り、3 の平方根を求めよ。

3. (optional) 上の 1,2 で作ったプログラムを拡張し、繰り返しにかかる回数を数え、違いを比べよ。