プログラミング基礎'94 ＃2

久野　靖*

1994.7.22

5　整数と実数

さて次に、これまで数を扱うのにすべて real(実数) を使ってきたが、個数や順番などを数えるのには整数 (integer) を使った方が何かと好都合である。そのため、変数 (値を入れておく箱) も real と integer を区別する。すなわち

```
var x, y, z: real;
    i, j: integer;
```

のように種別ごとに分けて書く。整数を分けると何が好都合か? まず、これまでの除算は結果が少数になっていたが、整数同志の計算では切捨て除算と剰余が使える。それらは (ちょっと見にくいけれど)

```
i div j
i mod j
```

のように書く。(*や/などの記号がもう足りないので名前で演算を表している。) そして、切捨てや丸めは

```
trunc(i)
round(i)
```

で表せる。なお、実数と整数をまぜて計算すると整数は自動的に実数に変換されるが、逆に実数を整数にしたい場合は必ず trunc か round を使うこと。では、これらを使って時分秒の差の問題を全部秒に直す方法で設計してみる。プログラムは次の通り。

```
program timedif(input, output);
var h1, m1, s1, h2, m2, s2, h3, m3, s3: integer;
begin
  write('h1> '); readln(h1);
  write('m1> '); readln(m1);
  write('s1> '); readln(s1);
  write('h2> '); readln(h2);
  write('m2> '); readln(m2);
  write('s2> '); readln(s2);
  s3 := (h2*3600 + m2*60 + s2) - (h1*3600 + m1*60 + s1);
  m3 := s3 div 60; s3 := s3 mod 60;
```

*筑波大学大学院経営システム科学専攻

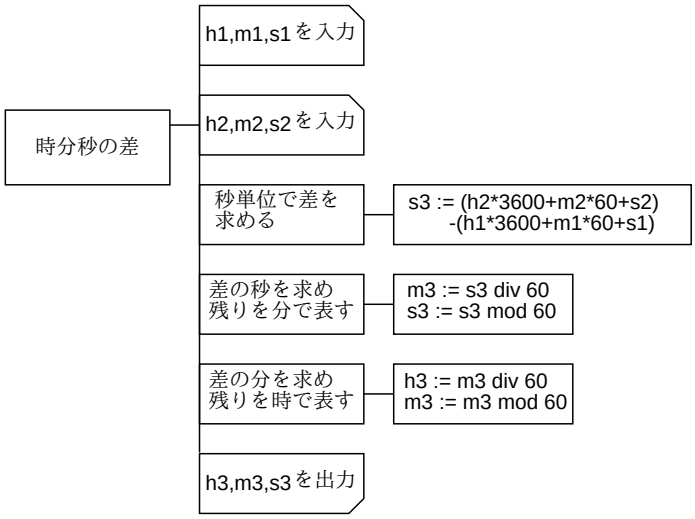

図 1: 時分秒の差 (秒換算版)

```
  h3 := m3 div 60; m3 := m3 mod 60;
  writeln('diff: ', h3:0, 'H ', m3:0, 'M ', s3:0, 'S')
end.
```

ところで、なぜ前回 if 文を使ったのに今回は if 文が要らなくなったのだろう? それは、整数の除算というのは「これ以上引けなくなるまで引いてみて何回引けたか数える」という、判断を含んだ操作だからだと考えればよい。

練習 2-1 ポケットの小銭の金額 (1000 円未満) を入力し、それを硬貨の数ができるだけ少なくなるように両替した場合、いくらの硬貨がそれぞれ何枚になるかを計算する PAD を設計せよ。

6 条件の複合、多方向の枝別れ

ここまで、if 文に使う条件というのは二つの数値の大小などだったが、実はこれらの条件を「組み合わせて」使うことができる。具体的には

```
(条件 1) and (条件 2) : 2 つの条件がともに成り立つという条件
(条件 1) or (条件 2)  : 2 つの条件の少なくとも一方が成り立つという条件
not (条件 1)          : 条件 1 が「成り立たない」という条件
```

の 3 つがある。そして、これらをいくらでも組み合わせることができる。なお、条件の組み合わせ順序が間違うと困るので、積極的にかっこを使うこと。

これを用いて、「3 つの数のうち最大のものを求める」問題を書き直してみよう。図 2 のようになるが、ここに新しい形の PAD の箱がある。これは「多方向分岐」といい、ぎざぎざの箱にいくつもの条件をならべて書いてある。その内容は、まず最初の条件を見てそれが成り立っていればその右の箱を実行、成り立っていなければ次の条件を見てそれが成り立っていればその右の箱を実行、…のようにして右側の複数の枝のうちどれか 1 つだけ (最初に成り立った条件に対応するもの) を選んで実行するというものである。条件の数はいくつでもよい。

多方向分岐を Pascal に直す場合には、次のような書き方になる。

```
if 条件 1 then begin
  ...
```

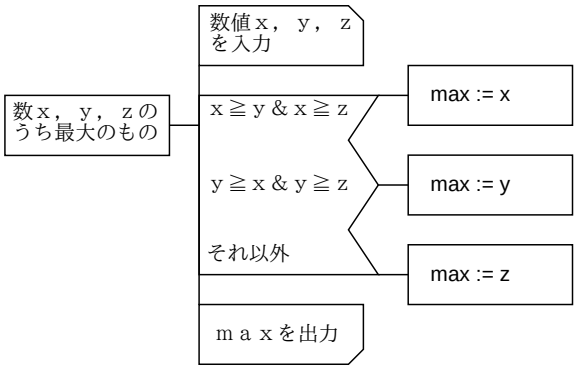

図 2: 3 つの数の最大を求める (複合条件版)

```
end
else if 条件 2 then begin
  ...
end
else if 条件 3 then begin
  ...
end
else begin
  ...
end;
```

条件がいくつになっても同様である。ではこれを用いて図 2 の PAD をプログラムにしたものを示す。

```
program max3(input, output);
var x, y, z, max: real;
begin
  write('X> '); readln(x);
  write('Y> '); readln(y);
  write('Z> '); readln(z);
  if (x >= y) and (x >= z) then begin max := x end
  else if (y >= x) and (y >= z) then begin max := y end
  else begin max := z end;
  writeln('max value = ', max:7:3)
end.
```

前にやったのとどれが判りやすいでしょうか? なお、効率はこれが一番悪い。けれど、効率と判りやすさのトレードオフを考えれば、これがいいという選択もあり得る。

練習 2-2 西暦を入力すると、その年がうるう年かどうか表示するプログラムの PAD を設計せよ。[1]

練習 2-3 西暦を元号表示に直すプログラムの PAD を設計せよ。

[1]西暦の年号が 4 の倍数ならうるう年。ただし 100 の倍数ならうるう年ではない。でも 400 の倍数ならやっぱりうるう年。

練習 2-4 元号表示を西暦に直すプログラムの PAD を設計せよ。ただし元号は明治=1、大正=2、昭和=3、平成=4 のように符号化して入力する。

練習 2-5 練習 2-1〜2-4 の PAD を Pascal に直して打ち込み実行せよ。

ところでアドバイス。十分慣れるまでは、Pascal プログラムを一気に上から打ち込むのではなく、PAD の箱の根元から順に打ち込み、だんだん細かい部分を入れるようにした方が begin-end の対応などが間違えにくい。

7 反復アルゴリズム

さて、前回の内容の中に「四則演算は計算機の命令としてあるけど平方根はない」というのがあった。今回はまず任意の正の数 x 平方根を求めるのからやってみよう。それには次のようにする。

- $a = 1$、$b = x$ とする。平方根は a と b の間の区間にある。
- ある方法で、この区間の幅を半分にする。それには a と b の中点 c を計算し、c^2 と x の大小を見る。c^2 が大きければ b を c で置き換え、そうでなければ a を c で置き換える。
- これを何回かやるとだんだん区間が小さくなる。

たとえば 4 回やることにして、PAD を書いてみたのが図 3 である。

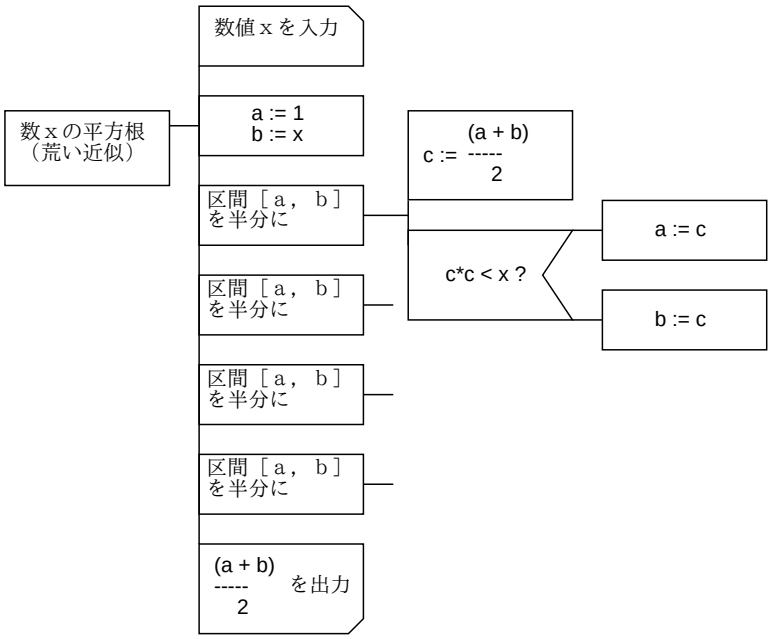

図 3: 平方根の近似を求める

「区間を半分…」のところは全部同じで面倒なので 1 回だけ書いた。そして、これを Pascal にしてみると次のようになる。

```
program root1(input, output);
var x, a, b, c: real;
begin
  write('X> ');
  readln(x);
```

```
  a := 1.0; b := x;
  c := 0.5 * (a + b);
  if c*c < x then begin a := c end else begin b := c end;
  c := 0.5 * (a + b);
  if c*c < x then begin a := c end else begin b := c end;
  c := 0.5 * (a + b);
  if c*c < x then begin a := c end else begin b := c end;
  c := 0.5 * (a + b);
  if c*c < x then begin a := c end else begin b := c end;
  writeln('root value = ', 0.5 * (a + b):6:2)
end.
```

動かしてみよう。

```
% a.out
X> 2
root value =   1.41
% a.out
X> 10
root value =   2.97
%
```

もっと精度を上げるには、半分にする回数を増やせばいいわけだが…しかしその分だけ同じものを何回も書くのはばかげていますね? どうすればいいでしょう? みえみえですが…もちろん、プログラムに繰り返し (ループ) を導入すればよい。そこで、精度 e も入力させて、区間の幅が e 以下になるまで繰り返し区間を半分にしていこう。PAD を図 4 に示す。

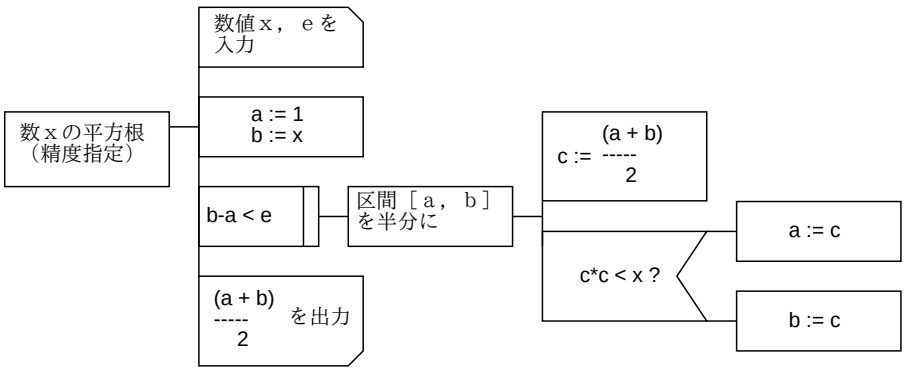

図 4: 平方根の近似 (反復版)

この中で新しく出て来た、右にタテ線のついた箱は「繰り返し」を表す。このように箱の中に条件が入っている場合には、その箱の右にある部分はその条件が成り立っている限り繰り返し実行される。ということは、その部分のどこかで条件に現われる変数を変更しているはずである。さもないと、永遠に繰り返しが終らないことになるでしょう?

このような「条件が成り立っている間の繰り返し」は、Pascal では while という構文で表す。その形は次の通り。

```
while 条件 do begin
  ...
end;
```

これを使って先の PAD をプログラムにすると次のようになる。

```
program root2(input, output);
var x, e, a, b, c: real;
begin
  write('X> '); readln(x);
  write('E> '); readln(e);
  a := 1.0; b := x;
  while b - a > e do begin
    c := 0.5 * (a + b);
    if c*c < x then begin a := c end else begin b := c end;
  end;
  writeln('root value = ', 0.5 * (a + b):10:8)
end.
```

ずっと短いでしょう? 今度は必要な精度を指定すればいいわけだから…

```
% a.out
X> 3
E> 0.0000000001
root value = 1.73205081
% a.out
X> 10
E> 0.000000000000001
root value = 3.16227766
%
```

これで平方根の求め方がわかったので、これからは平方根はわかったものとしてどんどん手順の中で使ってよいことにする。実は Pascal では式の中で `sqrt(x)` のように書くと平方根を求めてくれる。その他、対数関数や指数関数や三角関数もちゃんとある。(教科書参照。)

練習 2-6 いよいよ、繰り返しが入ったので「何回でも気が済むまで色々なデータを試せる」ようにプログラムを作ることができる。図 4 の PAD を直して、x の値が 0 でない限り繰り返し平方根の計算を行なうようにしてみよ。

練習 2-7 いくつかの数の平均を計算するプログラムの PAD 設計せよ。データは次々に入力し、最後に 0 を入れるとそこで終わりとする。[2]

練習 2-8 正の整数 n を入力し、その階乗を求めるプログラムの PAD を設計せよ。

練習 2-9 正の整数 x を入力し、それが素数かどうか調べるプログラムの PAD を設計せよ。

練習 2-10 元金 m と利率 r と目標金額 g を入力し、利率 r の貯蓄商品に m 円入れたとして、それが元利合計 g 円を突破するまで毎年の元利合計を出力する PAD を設計せよ。

練習 2-11 2 つの数 x、y を入力し、その最大公約数を求めるプログラムの PAD を設計せよ。

練習 2-12 m 個互いに異なるものから n 個を取り出すやり方 (組み合せ) の数を求めるプログラムの PAD を設計せよ。

[2]概略は次の通り。まず合計 s、データ個数 n とも 0 にしておく。データ x を読み込み、それが 0 でない限りそれを合計に加え、個数も 1 増やし、次のデータ x を読み込む。終りの 0 が来たら合計を個数で割って平均を求める。