プログラミング基礎'94 ＃3

久野　靖*

1994.7.23

8　計数反復

前回の練習問題の最後に「組合せの数を求める」というのがあった。一般に n 個のものから r 個のものを取り出す組合せは

$$\frac{(n-r+1)\times(n-r+1)\times\ldots\times n}{1\times 2\times\ldots\times r}$$

例えば 6 個から 3 個取り出す場合は

$$\frac{4\times 5\times 6}{1\times 2\times 3}$$

になる。これを計算するには図 1 の PAD のようにすればよい。

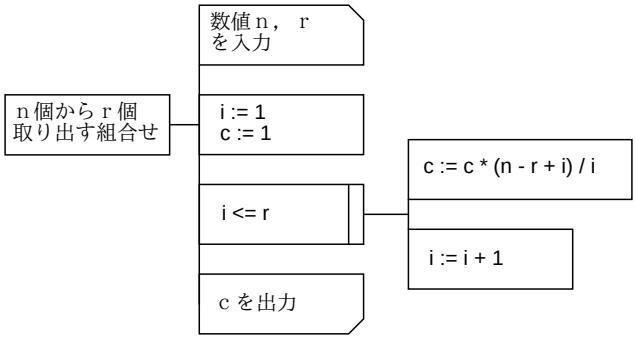

図 1: 組合せの数を求める

これを Pascal に直したものは次の通り。

```
program comb1(input, output);
var c, i, n, r: integer;
begin
  write('N> '); readln(n);
  write('R> '); readln(r);
  c := 1;
  i := 1;
  while i <= r do begin
    c := c * (n - r + i) / i;
    i := i + 1
```

*筑波大学大学院経営システム科学専攻

```
  end;
  writeln('combination = ', c:0)
end.
```

ところで、この中で変数 i は 1、2、3、…、r まで順に変化していく。このように、ある変数を予め決まった数まで 1 つずつ増加 (ないし減少) させいくようなループを「計数ループ」ないし「for ループ」という。先の平方根や合計のためのループは、予めループの回数を決めておくことはできないので for ループではない。一方、階乗や素数の問題に出て来るループは for ループである。

for ループは非常によく現われるので、これを専用の構文で表すことができる言語が多い。Pascal もそうである。また、PAD でも while とは違った書き方でそれを表すことにする。先のを for ループに直したものを図 2 に示す。

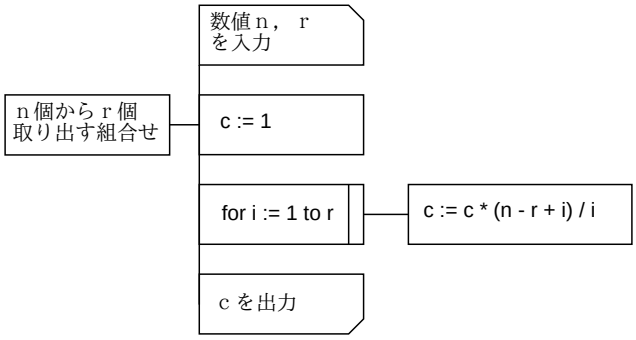

図 2: 組合せの数を求める (for ループ版)

そして、Pascal では文字通り for 文でこれを表す。その形式は

```
for 変数 := 式 1 to 式 2 do begin
  ...
end;
```

である。この場合、変数にはまず式 1 が入り、それ+1 が入り、…と繰り返され、式 2 まで来ると終る (to のところを downto と書くと変数が 1 つずつ減る計数ループになる)。これをもちいて先の PAD を Pascal にしたものは次の通り。

```
program comb2(input, output);
var c, i, n, r: integer;
begin
  write('N> '); readln(n);
  write('R> '); readln(r);
  c := 1;
  for i := 1 to r do begin
    c := c * (n - r + i) / i
  end;
  writeln('combination = ', c:0)
end.
```

なお、for と while のどちらを使うかは、予めループを回る回数が決まっているかどうかで決めればおおむねよい (が、絶対の正解はない)。

練習 3-1 元金 m、利率 r、年数 n を読み込み、0 年目から n 年目までの元利合計の推移を表示するプログラムを設計せよ。

練習 3-2 ローン借り入れ額 m、金利 r、返済額 x を読み込み、完済するまでの毎年のローン残額と返済額累計を表示するプログラムを設計せよ。

9 配列

さて、ループが使えるようになったので、繰り返し大量の計算をすることはできるようになった。しかし、一時に扱えるデータの量はこれまでのところ変数の数だけに限られていた。これを打開するためには、数式で言えば

$$x_1, x_2, x_3 \ldots x_n$$

のように「添字のついた」変数のようなものがあればよい。これをプログラム言語の言葉では「配列」という。配列とは、N 個の箱が連なった形のものだと思えばよい。

配列を使うには、配列型の変数を用意する。それには、var の部分で

```
a: array[1..100] of integer;
x: array[0..50] of real;
```

のように配列である旨、添字の下限、上限、そして個々の箱の種類を指定すればよい。いくつ箱を用意するかはプログラムの必要に応じて決まる。これからはプログラムの設計のとき、PAD とは別にどんな配列を用意するかも記しておくことにしよう。

では最初の簡単な例題: 0 が来るまで次々にデータを読み込み、そのデータを逆順に打ち出すというものである。PAD を図 3 に示す。

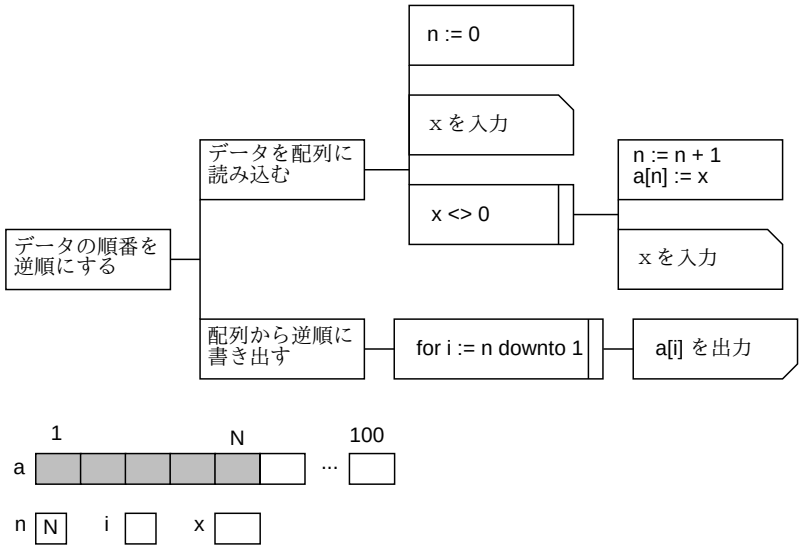

図 3: 一連のデータを読み込み逆順に表示

これを Pascal に直すと次のようになる。要は配列変数の宣言と、あとは配列の添字は「配列名[添字式]」のように書く、ということだけである。

```
program rev1(input, output);
var a: array[1..100] of real;
    x: real;
    n, i: integer;
begin
```

```
  n := 0;
  write('X> '); readln(x);
  while x <> 0 do begin
    n := n + 1;
    a[n] := x;
    write('X> '); readln(x)
  end;
  for i := n downto 1 do begin
    writeln(a[i]:8:4)
  end
end.
```

練習 3-3 配列にデータを読み込み、そのデータの平均と標準偏差を求めるプログラムを設計せよ。

練習 3-4 配列にデータを読み込み、そのデータのうち最大の値と、それが何番目のデータか (複数あるなら若い番号) を表示するプログラムを設計せよ。

練習 3-5 配列にデータを読み込み、それを小さい順に並べ替えて打ち出すプログラムを設計せよ。

練習 3-6 練習 3-1〜3-5 で設計したプログラムを Pascal でコーディングして動かせ。

10 文字型と文字列

ここまでデータの種別 (「型」ともいう) としては数値のみを扱ってきたが、そろそろ文字も扱ってみようと思う。文字を格納するための変数の種別 (つまり型) は「char」である。例えば次のようにして文字型の変数を用意できる。

```
var c: char;
```

ただし! **1 文字ぶん**の場所であることに注意。ではこれを利用して、2 進数 (0 と 1 のならび) を入力するとそれを 10 進数として出力するプログラムを書いてみる。

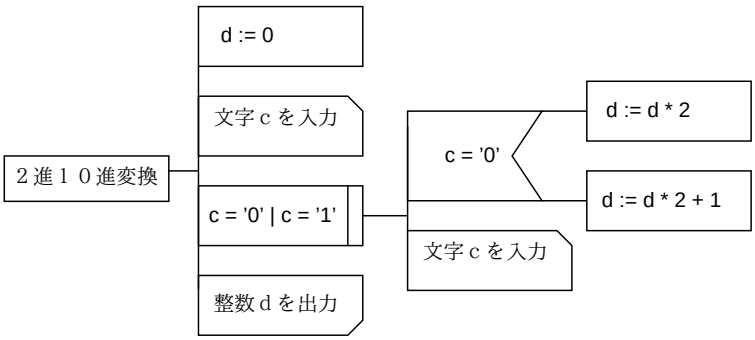

図 4: 二進十進変換

これを Pascal プログラムにすると次の通り。

```
program bindec(input, output);
var d: integer;
    c: char;
begin
```

```
  d := 0;
  write('binary number> '); read(c);
  while (c = '0') or (c = '1') do begin
    if c = '0' then begin d := d * 2 end
    else begin d := d * 2 + 1 end;
    read(c)
  end;
  writeln('decimal: ', d:12)
end.
```

ここで面白いのは、readln ではなく read を使っていること。readln では、データを読み込んだ後改行まで読み飛ばしてしまうが、read ではそれをしないので連続した文字がそのまま入力できる。つまり次のような感じになる。

```
% a.out
binary number> 100111
decimal:           39
%
```

この方がらしいでしょう? ところで、そうすると while ループは 0 でも 1 でもない文字が来たら終るが、そういう文字を入れなければならない? なんですが、改行を入れるのでそれが 0 でも 1 でもない文字として使える。(実は Pascal では改行文字はそのままは読めず、read によって空白にとりかえられてしまう。でもこのプログラムではそれで問題ない。)

練習 3-7 今度は逆に、10 進数を入力するとそれを 2 進形式で打ち出すプログラムを設計せよ。[1]

11 文字列、型定義、データ構造

それにしても、文字データはやっぱり 1 文字だけで扱うよりは、何文字かまとめて文字列として扱うのがよい。なので、Pascal では文字の配列に限って

```
var s: array[1..?] of char;
...
  readln(s);
  ...
  writeln(s);
```

のように、まとめて読んだり書いたりできるようになっている。なお?のところには必要な最大の長さを入れる。ところで、readln のところで入力した文字列がその最大長より短かったら? その時は不足ぶんには空白が入る。

ところで、integer や real や char は名前なのに文字列だけ array[1..?] of char などと書くのでは読みにくいし繰り返し使う時面倒だという気がしますよね? そこで Pascal では型の名前を自分でつけることができるようになっている。それには var よりも前に型定義のための type 宣言を書く。

[1] ヒント: 一番下の桁が 0 か 1 かは 2 の剰余をとればわかる。わかったらそれは覚えておいて、入力した数を 2 で割ってしまう。そこでまた 2 の剰余を調べると下から 2 番目の桁がわかる。これを数が 0 になるまでやる。覚えておくには当然、文字列の配列を使う。

```
type str = array[1..20] of char;
var s: str;
...
```

これからはなるべく型定義を活用することにする。

さて、数値と文字列とその配列があると、プログラムの内部にある程度込み入ったデータを格納していろいろ扱うことができる。例えば図 5 のような個人データの表を格納して処理したいとする。

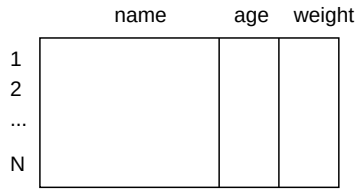

図 5: 概念的な表

それには、今まで習った範囲でやるとすれば、name、age、weight をそれぞれ str、integer、real の配列 (上限、下限は共通) にしてやればよさそうである。そして、どこまでデータが入っているかを別の変数 n に保持しておく (図 6)。

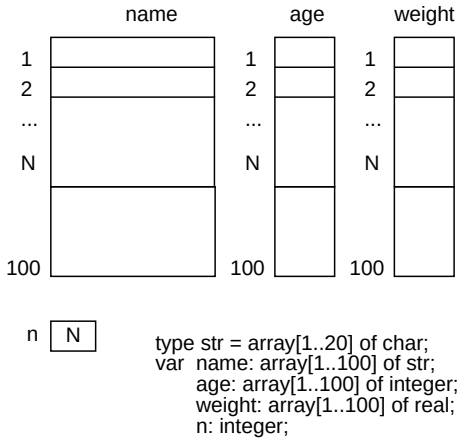

図 6: 表の実現

このように、必要な情報を格納するための変数などの構成を「データ構造」と呼ぶ。ある問題を計算機で扱いたい場合、それに適した「データ構造の設計」と「アルゴリズムの設計」が車の両輪となる。

ではせっかくデータ構造を設計したので、この表をコマンドに従って扱うプログラムを設計してみよう。こんどはだいぶ大きい PAD になるので、適宜分割してある。

そして、これを Pascal にしたものは次の通り。

```
program talbe1(input, output);
type str = array[1..20] of char;
var  name: array[1..100] of str;
     age:  array[1..100] of integer;
     weight: array[1..100] of real;
     n: integer;
```

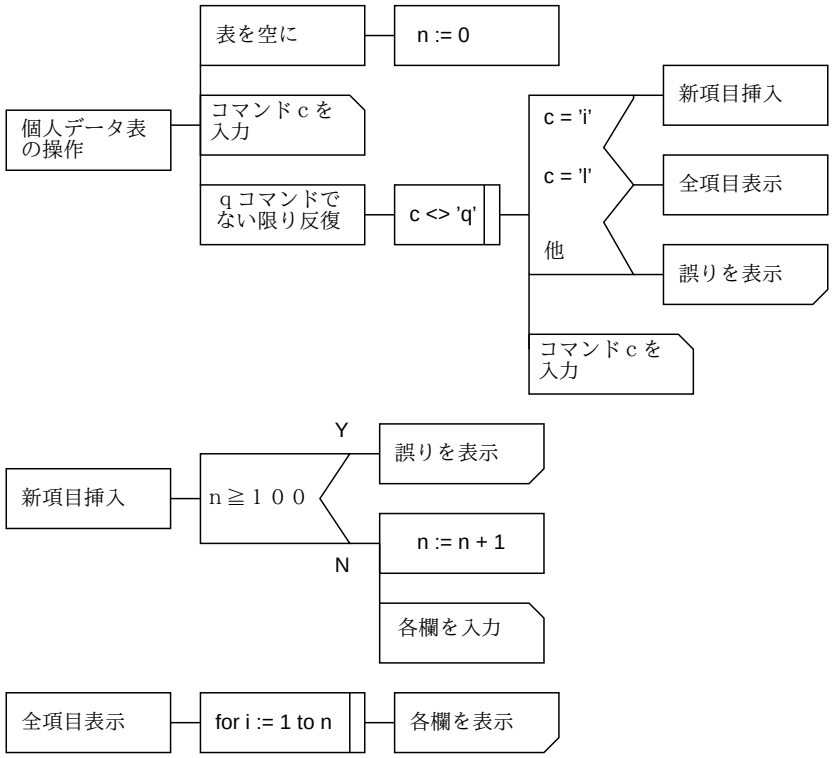

図 7: 表のコマンドによる操作

```
    i: integer;
    c: char;
begin
  n := 0;
  write('Command> '); readln(c);
  while c <> 'q' do begin
    if c = 'i' then begin
      if n >= 100 then begin
        writeln('table overflow.')
      end
      else begin
        n := n + 1;
        write(' name> '); readln(name[n]);
        write(' age>  '); readln(age[n]);
        write(' weight> '); readln(weight[n])
      end
    end
    else if c = 'l' then begin
      for i := 1 to n do begin
        writeln(name[i]:12, age[i]:2, weight[i]:8:2)
      end
    end
    else begin
      writeln('unknown command: ', c);
```

```
      end;
      write('Command> '); readln(c)
    end
  end.
```

どうです、だいぶ大きいプログラムが書けるようになったでしょう?

練習 3-8 上の表操作プログラムの設計に、次のようなコマンドを追加してみよ。

(a) 名前を指定すると何番目の項目かを表示する。
(b) 指定した番号の項目を削除する。
(c) 指定した 2 つの番号の項目を入れ換える。
(d) 表を名前順にならべ替える。

練習 3-9 練習 3-7、練習 3-8 の設計を Pascal にして動かせ。