「情報処理」1年文I/IIクラス9-10　＃8

久野　靖*

1994.12.12

まず前々回の課題の解説がありますが、結構時間が厳しくなってきたので授業では簡単にやります。そして、配列とPBMファイルの出力をやりますが、いずれも冬休み課題に必要なものです。課題を早く知りたいという要望があったので、冬休み課題も付録につけておきますが、まだ期限まで40日もありますからゆっくり考えてください。本日の目標は次の通り。

- 配列について理解し、PBM形式ファイルをプログラムで作れるようになる。
- ディレクトリとファイルについて、もう少し知識を増やす。

1　前々回の課題の解説

まず「xとyの積」(図1)のPascalプログラムから。特に問題はないですね? ただ、ループ本体が2つの文だからbeginとendで囲む必要がある。

```
program sam8b(input, output);
var x, y, n: integer;
begin
  write('x = '); readln(x);
  write('y = '); readln(y);
  n := 0;
  while x > 0 do begin n := n + y; x := x - 1 end;
  writeln('n = ', n:1)
end.
```

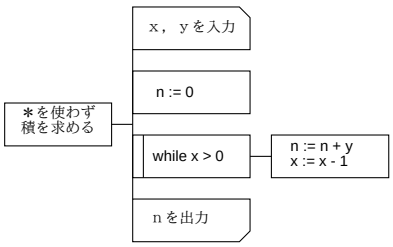

図1: 「*」を使わずに積を求めるPAD

次は「最大公約数」だが、まずPADを図2に示す。「xとyが等しくない間反復する」ことが分かればあとは問題文の通りだと思うが。Pascalは次の通り。

*筑波大学大学院経営システム科学専攻

```
program sam8c(input, output);
var x, y, z: integer;
begin
  write('x = '); readln(x);
  write('y = '); readln(y);
  while x <> y do begin
    if x > y then begin z := x; x := y; y := z end;
    y := y - x
  end;
  writeln('gcd[x,y] = ', x:1)
end.
```

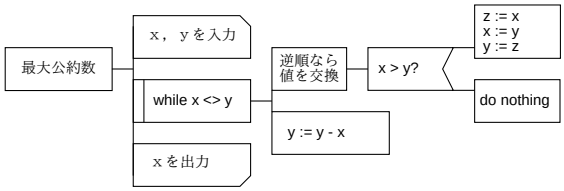

図 2: 最大公約数を求める PAD

次は演習 3 の a だが、割り算とは結局「引いたら何回引けたか」であり、余りは文字通り引いた余りだということ。while の条件として、「まだ引ける」つまり x がまだ m 以上である、と書くことがみそ。

```
program sam10a(input, output);
var x, m, n: integer;
begin
  write('x = '); readln(x);
  write('m = '); readln(m);
  n := 0;
  while x >= m do begin x := x - m; n := n + 1 end;
  writeln('shou = ', n:1, '  amari = ', x:1)
end.
```

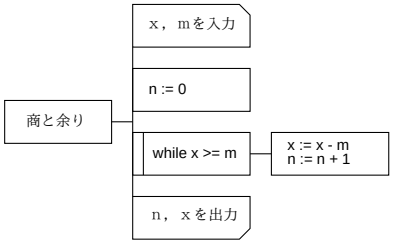

図 3: div を使わずに商と余りを求める PAD

b の階乗は、要するに減らしながら順に掛けていくこと。0 を掛けてしまうと 0 になってしまうので、その手前でやめるのがみそ。

```
program sam10b(input, output);
```

```
var n, x: integer;
begin
  write('n = '); readln(n);
  x := 1;
  while n > 0 do begin x := x * n; n := n - 1 end;
  writeln('factorial = ', x:1)
end.
```

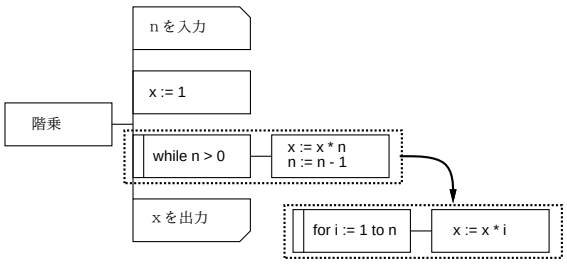

図 4: 階乗を求める PAD

なお、20 とか大きい数の階乗を求めようとしてうまく行かないという人が多数いらっしゃいますが、＃ 0 でやったように計算機では数を決まったビット数に対応させているので、そのビット数に納まらない大きな数はうまく計算できません。だいたい integer だと 10 桁くらいまでで限界ですね。

ところで、ここでは while は単に n を 1 つずつ決まった回数だけ変化させるのに使われているだけである。だから、(かけ算はどちらかやっても結果は変らないので) 点線枠の部分を for で取り換えてやることもできる。するとプログラムの方は次のようになるだろう。どちらが好みですか?

```
program sam10b1(input, output);
var n, x, i: integer;
begin
  write('n = '); readln(n);
  x := 1;
  for i := 1 to n do x := x * i;
  writeln('factorial = ', x:1)
end.
```

c は実は一番簡単で x を 1 から 1 ずつ増やしながら試すわけだが、これも x*x が n 以上になったらおしまいなので x*x が n 未満の間、という条件がみそ。

```
program sam10c(input, output);
var n, x: integer;
begin
  write('n = '); readln(n);
  x := 1;
  while x*x < n do x := x + 1;
  writeln('x = ', x:1)
end.
```

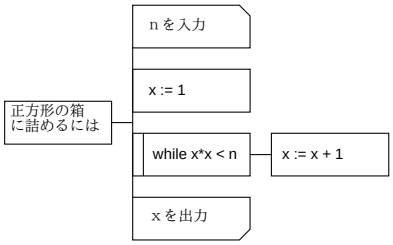

図 5: 箱詰め問題の PAD

2 配列

前回には、ループを使って一般に N 個のデータを次々に読み込んで処理したり、生成して出力することを学んだ。しかし、プログラムの中で扱っているデータは「現在高」とか「合計」など単一の数値で、それが 1 つの変数に入っているだけだった。それで済むのならそれでもよいが、計算機の中に N 個のデータを蓄えて扱いたい場合もある。例えば、次の例題を考えてみよう。

a. N 個のデータ (0 が来たらおしまい) を読み込み、読み込んだのとは逆の順に出力せよ。N は最大でも 200 とする。

ここで N がたとえば 4 個という風に決まっていれば話は簡単で、次のようなあほみたいなプログラムを書けばよい (PAD は略した)。

```
program sam11a(input, output);
var x1, x2, x3, x4: real;
begin
  write('x1 = '); readln(x1);
  write('x2 = '); readln(x2);
  write('x3 = '); readln(x3);
  write('x4 = '); readln(x4);
  writeln(x4:8:3);
  writeln(x3:8:3);
  writeln(x2:8:3);
  writeln(x1:8:3)
end.
```

しかしこれが 100 個とか 200 個になると、とても書く気にならないだろうし、一般に N 個ということになるとどうするのだろう?

ここで、もし変数に「添字」が使えたとすれば、この問題は次のように書くことができるだろう。「$x_1 \sim x_{200}$ までの変数を用意しておく。データを次々に読み、i 番目のデータを変数 x_i に入れる。0 が来たらその手前を N 番目として、$x_N \sim x_1$ の順に出力する。」

こういうことをするために、Pascal では「添字つきの変数」を使う機能が用意されている (プログラミング言語の用語では「配列」(array)、つまり何やらならんだもの、というコトバを使うのが伝統である)。もちろん、プログラムを打ち込んでいる時に小さい字は使えないので、添字は「[、]」で囲んで、たとえば x_i だったら `x[i]`、a_{n-1} だったら `a[n-1]` のように現す。変数宣言の方も、例えば次のようにする。

```
a: array[0..50] of integer;
x: array[1..200] of real;
```

ここで「0..50」とか「1..200」は使える添字の範囲を指定していて、「of integer」とか「of real」は 1 つひとつの変数が整数か実数かを指定している。配列は図 6 のように普通の変数と同じものが必要なだけ「並んでいる」ものと考えてよい。

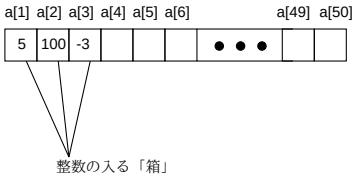

図 6: 配列の概念

これを使えば、先の問題は図 7 の PAD と、次のような Pascal プログラムで書くことができる。

```
program sam12a(input, output);
var n, i: integer;
    d: real;
    x: array[1..200] of real;
begin
  n := 0;
  write('data = '); readln(d);
  while d <> 0.0 do begin
    n := n + 1; x[n] := d;
    write('data = '); readln(d)
  end;
  for i := n downto 1 do writeln(x[i]:8:3)
end.
```

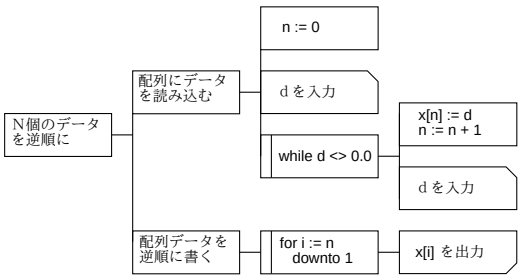

図 7: N 個の数を逆順にする PAD

なお、最後の方に出てくる downto というのは始めてだった。これは、for 文の変形で、次の形をしている。

for 変数 := b downto a do 文

これまでのものとに違いは、これまでの for が「変数」を 1 つずつ増やしながら数えるのに対し、こちらは 1 つずつ減らしながら数えるというだけである。(だからこの問題にはうってつけである。)

演習 1 上のプログラムを打ち込んでそのまま動かせ。

演習 2 N 個のデータを読み込み (0 が来たらおわり)、平均値と平均値より大きいものの個数を打ち出すプログラムを作れ (PAD を図 8 に示しておく)。

演習 3 N 個のデータを読み込み (0 が来たらおわり)、平均値と平均値に一番近いデータ値を打ち出すプログラムを作れ。[1]

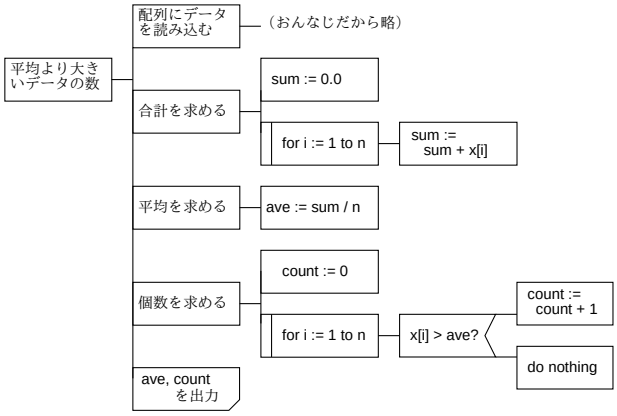

図 8: 平均より大きいデータの数を数える PAD

3 2 次元配列と PBM 形式ファイル

3.1 2 次元配列

さて、線形代数 (その言葉を聞きたくないという人はすいません) などでは、行列の要素を指定するのに 2 つ以上の添字を使いますね (x_{ij} のように)。Pascal の配列でも同様に、添字を 2 つ以上つけることができる。その場合には `x[i, j]` のように添字どうしをカンマで区切って書く。また、変数宣言のところでも

```
x: array[1..20, 1..100] of integer;
```

のように範囲を複数ならべてカンマで区切る (範囲は添字ごとに違っていてよいことに注意)。これは、図 9 のように整数の箱が縦横に並んでいるものと考えればよい。一般に添字の数を「次元」といい、先に出て来た添字が 1 つの配列を 1 次元配列、ここに示した 2 つのものを 2 次元配列とよぶ。(なお、3 次元以上の配列はややこしいのでめったに使われない。)

さて、2 次元配列を使った手頃な例題として、いよいよお絵描きをやろう。前に、計算機画面というのは様々な色の「点」から成っているという話をしましたね? そして画面は平面 (2 次元) だから、これを 2 次元配列に対応させて考えればちょうどよい。まずは簡単な白黒の絵から始めることにして、従ってそれぞれの点は「0」と「1」で表すことができる。

3.2 PBM 形式ファイル

次に、Pascal プログラムの内部で作った絵をどうやって画面で見るかを説明しよう。それには、プログラムに「PBM 形式」の出力を出させて、それをファイルに保存して xv をつかって見る。PBM 形式というのはとても簡単で、次の形をしている。

[1]「一番近い」を判定するには「絶対値」を使うと便利。式 x の絶対値は `abs(`x`)` で求められる。

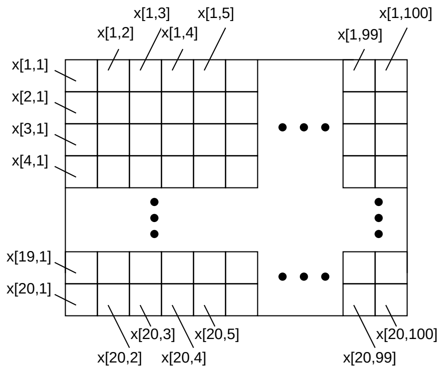

図 9: 配列の概念

```
P1 幅 高さ
必要な個数の 0 と 1 の並び…
```

つまり、たとえば

```
P1 20 10
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
```

という内容のファイルは白地に縦線の入った幅 20 ドット、高さ 10 ドットの絵を表している。なお、1 行にいくつ 0/1 を入れてもよいし、1 行に 1 つずつ 0 か 1 を書くようにしても構わない。

演習 4 Mule で上の内容と同じもの (0 や 1 は適宜変更してもよい) を作って「`test.xbm`」というファイルに書き出し、

```
xv test.xbm &[RET]
```

により xv で表示させてみよ。(小さくてよく見えない場合には右ボタンをつついて制御パネルを出し、「Dbl Size」をつつくと大きくできる)。

3.3 プログラムによるお絵描き

さて、今度はエディタではなくプログラムで絵を描いてみる。図 10 に「ななめ線の絵を描く」プログラムの PAD を示す。まず、絵の全体は 300 × 200 の大きさにすることにして (あまり大きいとファイルが大きく出力に時間が掛かる)、`p` という 2 次元配列を絵の全体に対応させる。最初に全体を白くするが、それには for ループの中にさらに for ループをいれて、その中で `p` の要素を 0 にすることでできる (おわかりかな?)。斜め線は線の上に対応する場所を白 (0) から黒 (1) に書き換えることで描くことができる。簡単のため、角度は 45 度にしてある。最後に、`p` の内容を書き出すが、最初に PBM ファイルの 1 行目 (大きさ指定) を書き、あとは再び 2 重の for ループの中で `p` の各要素の値を書き出せばよい。(1 行に 1 個ずつ 0 か 1 が出るが、これを人間が見るわけではないのでそれは特に問題にならない。) これをプログラムにしたものを次に示す。

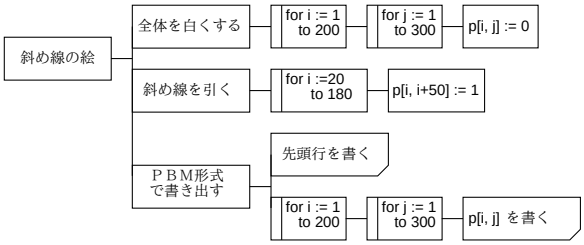

図 10: 斜め線の絵を描く PAD

```
program sam13a(input, output);
var i, j: integer;
    p: array[1..200, 1..300] of integer;
begin
  for i := 1 to 200 do
    for j := 1 to 300 do p[i,j] := 0;
  for i := 20 to 180 do p[i, i+50] := 1;
  writeln('P1 300 200');
  for i := 1 to 200 do
    for j := 1 to 300 do writeln(p[i,j]:1)
end.
```

このプログラムは入力データは全くなく、出力は PBM 形式ファイルのみなので、次のようにして出力をファイルに保存した後 xv を使って結果を眺める (xv の画面の様子を図 11 に示す)。

```
% pc sam13a.p
% a.out >test.pbm
% xv test.pbm &
```

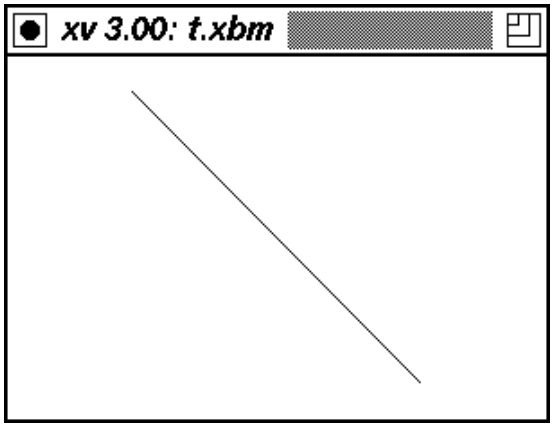

図 11: 斜め線の絵

演習 5 斜め線の例題プログラムを打ち込んで動かせ。成功したら、どんな風にでもいいから少し違った絵になるようにプログラムを直して動かせ。

演習 6 斜め線のプログラムを参考にして、次のいずれかのプログラム (PAD と Pascal) を作って動かせ。

a. ×印を描く。
b. 長方形を描く。
c. 塗りつぶした 3 角形を描く。

これだけだと物足りないという人もいそうなので、末尾に冬休みの課題を予告しておく。〆切は 1 月 23 日になるので、たっぷり楽しんで頂きたい。

4 ファイルとディレクトリ再訪

4.1 ディレクトリの木と絶対パス名

そろそろファイルが多数たまってきて、ディレクトリに分けて整理するのはいいとしても、ディレクトリの指定をいちいちするのが面倒くさいという人がいることと思う。そこで、その改善も兼ねてディレクトリ構造についてここでもう少し詳しく学んでおく。

まず、我々の使っている Unix システムには膨大な量のファイルが格納されていることは明らかですね? そのファイルはいったいどのように整理されているのだろう? それは皆様のファイルと同様に、やはりディレクトリ単位でまとめられている。そして、システム全体の「根もと」に当たるディレクトリは特別な名前「/」がつけられている。これを「ルートディレクトリ」といい、その下に各種のサブディレクトリが置かれている。これらの代表例を挙げておこう。

/	ルートディレクトリ
/bin	基本的なコマンドが格納されている
/etc	管理用コマンドやシステム設定ファイルがある
/usr	整理のための中間ディレクトリ
/usr/bin	基本コマンドの追加版がある
/usr/man	マニュアルページのおき場所
/usr/local	駒場システムローカルなもののおき場所
/usr/local/bin	ローカルコマンドのおき場所
/export	ネットワーク共有されるディレクトリのおき場所
/export/home??	ユーザのホームディレクトリをまとめる

そして、私や皆様のホームディレクトリは

```
/export/home10/g000001
/export/home14/kuno
```

などの場所にあり、そのサブディレクトリは

```
/export/home10/g000001/Mail
/export/home10/g000001/pascal                ←※ 3
```

これらのディレクトリにあるファイルは

```
/export/home10/g000001/test.tiff             ←※ 1
/export/home10/g000001/Mail/inbox/1
/export/home10/g000001/pascal/sam12a.p       ←※ 2
```

などとなるわけである。ここで home??の??のところはどのファイルサーバに格納されているかで異なるが、ともかくネットワーク機能を使用してどの端末から入っても自分のファイルは同じ場所にあるかのように使うことができる。(これはよく考えると結構すごい。パソコンが数百台あっ

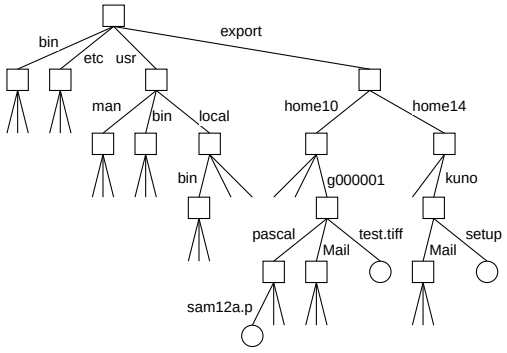

図 12: ディレクトリの木

て、どの前に座ってもいつも同じ自分用のファイルが使えるというのはほとんど考えられない。) このように、Unix ではありとあらゆるファイルが図 12 のように巨大な「ディレクトリの木」を構成している。

そして、上に沢山書いたように「/」から始めてサブディレクトリを順次記すことで、システム内にあるどのファイル/ディレクトリでもあいまいさなく指定できることはこの例をみればおわかりだろう (ちょうど住所のようなものですね)。このような指定方法を Unix では「絶対パス名」という。

4.2 カレントディレクトリと相対パス名

しかし、いつも絶対パスでファイルを指定させられては打ち込む量が多くてやりきれない。そこで、Unix では実行中の各プログラム (窓の中身なども実行中のプログラムである) が「現在位置」(カレントディレクトリ) を持つことができるようになっている。現在位置はいつでも「pwd」(print worikng deirecotry) コマンドで表示させられる。

```
% pwd
/export/home10/g000001
%
```

そして、「/」で始まらないファイル名を指定した場合には自動的に現在位置が前につけ加えられる。たとえば上の現在位置で「test.tiff」「pascal/sam12a.p」と指定するとそれぞれ※ 1 と※ 2 のファイルが指定できるわけである。このような「/」で始まらない指定方法のことを Unix では「相対パス名」という。Unix では一般にファイルやディレクトリを指定するのに絶対パス、相対パスのどちらを使うこともできる。

そこで話は、いちいちディレクトリを指定するのが面倒だという所に戻ってくる。実は、この現在位置は「cd」(change directory) コマンドで自由に変更できる。例えば「cd pascal」を実行すると、現在位置は※ 3 になるので、その後は※ 2 のファイルは単に「sam12a.p」で指定できる。「ls」も特にディレクトリを指定しなければ現在位置にあるファイルの一覧が表示される。なお、cd で新しい現在位置を指定するときも絶対パス、相対パスのどちらで指定してもよい。

ところで、現在位置をどこかのサブディレクトリにおいた場合、そこから木を下に向かってたどるのは問題ない (ただディレクトリの名前を言えばいい) が、上にむかってたどりたいこともある。例えば、図 13 で pascal サブディレクトリにいてホームディレクトリの test.tiff を指定したい場合などである。そこで、Unix では「..」という名前を指定すると「そこから 1 つ上へ戻る」ことを指定できるようになっている。だから、この場合は「../test.tiff」と指定すればいいわけである。ついでだが、「.」で現在位置そのものを指定することもできる (たまに使うことがある)。

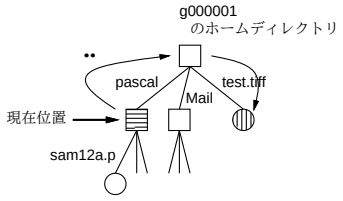

図 13: 「..」について

なお、cd コマンドでディレクトリを指定しないと、自分のホームディレクトリへ行くことになる (迷子になった時便利)。また、次の記法も知っておくと便利である。

~ … 自分のホームディレクトリ。例: `~/test.tiff`
~ユーザ名 … 指定したユーザの〃。例: `~kuno/setup`

ただし csh 系のシェルに限られる (といっても分らないでしょうけど、使えない場面もあるということ)。

演習 8 自分のどこかのサブディレクトリに現在位置を移動して、pwd や ls をやってみよ。また。そこで「`ls ..`」「`cd ..`」もやってみよ。

演習 9 「`cd /`」でルートディレクトリへ移動して、そこからあちこちのディレクトリをさまよい歩いてみよ。なお、ディレクトリは名前の末尾に「/」がついて表示されるのでしたね。

演習 10 私のホームディレクトリの下の方のどこかに「REPORT8A.txt」というファイルが置かれている。それを捜し出してみよ。

4.3 ファイルとディレクトリの保護

さて、ここまで来て「そうすると、誰のディレクトリの内容でも見放題なのか!」と思った人はいませんか。Unix は基本的に「誰でも一緒のシステムを使っている人は仲間うち」という思想でできているので、それはそれで思想にかなっている。が、やはり他人に見られたくない情報というのもあるわけで、それを説明しておこう。

Unix では各ファイル/ディレクトリについて、それを「誰が」「どうできるか」をある程度まで制御できる。この「ある程度」というのは次のようなものである。まず「誰が」は次の 3 種類しか区別できない。

- user — 自分 (ファイルの所有者) 自身。
- group — グループのメンバ。グループの構成はサイトごとに色々だが、ここでは「先生」というのは 1 つのグループ、「生徒」もまた 1 つのグループになっている…ようですね (よく知らない)。
- other — その他誰でも。

また、「どうできる」というのも次の 3 種類しか区別できない。

- read — そのファイルの内容を読むことができる。
- write — そのファイルに書き込むことができる。
- exexute — そのファイルに入っているプログラムを実行することができる。(ディレクトリの場合だと、そのディレクトリをたどることができる。)

ls -l コマンドを使うとこれらの情報を表示させることができる。

```
% ls -l msg1.txt
-rw-r--r--  1 kuno          151 Nov 11 10:35 msg1.txt
```

左側の「rw-r--r--」というのが、このファイルは所有者には読み書きができ、グループメンバには読むことができ、その他の人にも読むことができる。というのを表している。なお、ファイルの所属グループも表示させたければ「ls -lg」を使う。

```
% ls -lg msg1.txt
-rw-r--r--  1 kuno     teacher       151 Nov 11 10:35 msg1.txt
```

ここで、保護設定を変えるには chmod(change mode) コマンドによる。このコマンドは「chmod [ugo][+-][rwx]」という形をしていて、「どの範囲の人について」「追加する/削除する権限は」「どれ」をこの順で指定する。たとえば「自分」に「実行」権限を「追加」するには:

```
% chmod u+x msg1.txt
% ls -lg msg1.txt
-rwxr--r--  1 kuno     teacher       151 Nov 11 10:35 msg1.txt*
```

また「グループとその他」から「読む」権限を「取り除く」には:

```
% chmod go-rw msg1.txt
% ls -lg msg1.txt
-rwx------  1 kuno     teacher       151 Nov 11 10:35 msg1.txt*
```

というわけである。

演習 11 自分のファイルやディレクトリについて、「読めない」「書けない」などの設定を行い、確かに保護が効いているか確認せよ。特にディレクトリの「実行は可能だが読めない」という設定にはどういう意味 (用途) があると思うか?

演習 12 友人と協力して、友人がディレクトリを「誰でも書ける」にするとそこにあなたのファイルが作れてしまうことを確認せよ。また、その状態で友人の (もちろん、不要な!) ファイルを消せるかどうかもやってみよ。

A　本日の課題 8A

本日の課題は演習 2 または演習 5(直したもの) の好きな方の Pascal プログラムと実行例を提出してください。また、アンケートは演習 8 で探したファイルの中に書かれています。久々に紙で提出して頂きますので、アンケートとレポート番号氏名等を忘れないように。レポート番号は 8A です。

B　次回までの課題 8B

次回までの課題は、演習 3、演習 6 のどれか 1 つ、および演習 11 と 12 です。PAD と Pascal、および実行例 (演習 6 では結果の絵) のハードコピーを提出すること。演習 11 と 12 はやってみたらどうだったかを記述してください。レポート番号は 8B です。アンケートは次の通り。

- Q1. 配列について理解しましたか？しなかったとしたら、どの辺に問題がありますか? したとしたら、どの辺がポイントだと思いましたか?
- Q2. ファイル保護について、その必要性、利点、問題点を挙げて見てください。
- Q3. その他、感想、要望、質問があればどうぞ。

C　冬休みの課題 1R

冬休みの課題は次の通り。

今回の内容を参考に「美しい」絵を出力するプログラムを設計し作成せよ。

何をもって「美しい」と考えるかは各自に任されるが、自分のプログラミングの腕前から見て不当に易しい内容を選択しないこと。レポートは必ず A4 版の用紙を用い、以下の構成を取ること。

- [1.] 表紙。レポート課題番号 (1R)、学籍番号、氏名、提出日を記すこと。

  方針。問題を解くにあたって、どのようなことを考えどのような方針を立てたか。

3. 設計。プログラムの構造など (PAD 図はもちろんだが、PAD 図では把握しにくい情報があればそれも分るように工夫する)。

4. 回答。この場合にはプログラムと、生成された絵 (の窓の画面ハードコピー) ですね。

5. 考察。この課題をやったことで、新たにどのようなことが分ったか検討し考察する。今から考えるとどうした方がよかった (反省)、どう感じた (感想)、などの内容も含めてよいが、反省と感想が大部分というのでは困る。

6. 付録。何かつけ加えたい内容があれば。(今回はあまりなさそうですね…)

この課題 1R は、1 月 23 日の講義開始までに、1F のレポートボックスに提出すること。