「情報処理」1年文I/IIクラス9-10　＃12

久野　靖*

1995.1.30

レポートの採点を半分くらいやりました (さすがに大変です)。しかし、直線1本でもいいと公言した割には皆様すごく頑張って下さっているので感動しました。(WWW の作品ページを見ても分かりますね。) それで、WWW でも掲示した通り、これ位やれば充分だと思うのでレポート 2R は撤回します。他科目の試験に存分に傾倒してください。ただし来週の 13A までは通常通りあります。本日の目標は次の通り。

- 手続きについて復習し、関数の作り方について学ぶ。
- いつもお世話になっている X-Window について理解する。

レポート 1R を採点していて、「課題をやっていたら手続きの重要性がわかった」という感想を多く拝見しました。手続きというのはプログラミング言語においてとても重要な概念なので、最後にしつこくこれを取り上げることにする (来週は最終回ですし、Pascal ネタはなしにします)。ぜひマスターして頂きたい。

1　手続きと関数

1.1　関数について

これまでのところ手続きは「値を返す」ことができなかった。そこで、今日はまず値を返せるような手続きの変種である、「関数」(function) からやろう。なお、プログラム言語の「関数」は数学でいう「関数」とはほとんど関係がないので、数学嫌いの人もご安心を。実は、これまでにも Pascal プログラムの中で abs(x)(絶対値)、sin(x)、cos(x)、sqrt(x)(平方根) などは使ってきた。これらは「標準関数」つまり予め用意されている関数なわけである。ここでは、自分で独自の関数を作る方法を説明する。まず、関数定義の形式は次の通り。

```
function 関数名 (パラメタ…) : 型指定 ;
var 変数,… : 型指定; …                    ←変数がなければ不要
begin 文 ; 文 ; … ; 文 end;
```

つまり、関数定義は手続き定義とほとんど同じ形で、ただし「procedure」が「function」になっていることと、関数の値がどの型かを指定するところだけ違っている。そして、関数定義を入れる場所は手続き定義と同じである。あと1つだけ、覚えることがあるのだが、それは実例で示そう。以下に、2つの数を読み込み平均を打ち出す、という超かんたんなプログラムを、ただし関数を使って書いたのを示す。

```
program sam18a(input, output);
var a, b: real;
```

*筑波大学大学院経営システム科学専攻

```
  function ave2(x, y: real) : real;  { (1) }
  begin                              { (2) }
    ave2 := 0.5 * (x + y)            { (3) }
  end;                               { (4) }

begin
  write('a = '); readln(a);
  write('b = '); readln(b);
  writeln('heikin = ', ave2(a, b):8:3)
end.
```

つまり、(1)〜(4) で関数 ave2 を定義し、メインプログラムではそれを呼び出して平均を計算している。(1) では関数の名前 (ave2) と、それが 2 つの実数型のパラメタ x、y を受け取ることと、この関数が計算した結果は実数型であることが示されている。計算の本体は (3) で、x と y を足して半分にし、その結果を関数の値としている。一見 ave2 という変数に値を入れているように見えるが、変数ではなく関数の結果を定めている。

演習 1 上の例題を参考に以下のような関数を作り、メインプログラムも適宜変更してその関数を動かし結果を表示せよ。

- 3 つの実数の平均を返す関数 ave3。
- 2 つの実数のうち大きい方 (同じ時はその値) を返す関数 max2。
- 3 つの実数のうちいちばん大きいもの (〃) を返す関数 max3。

1.2 再帰関数

ところで、ある関数の処理のなかでその関数自身を呼び出すことができる。これを「再帰関数」と呼ぶ。そんなの堂々めぐりでおかしいと思いますか? 「漸化式」って知っていますよね。たとえば、$N!$(N の階乗) は次のように表せる。

$$factorial(N) = \begin{cases} 1 & (N \leq 1) \\ N \times factorial(N-1) & (o.w.) \end{cases}$$

これをそのままプログラムにしたのが次のものである。

```
program sam18b(input, output);
var x: integer;

  function factorial(n: integer) : integer;
  begin
    if n <= 1 then
      factorial := 1                      { (1) }
    else
      factorial := n * factorial(n - 1)   { (2) }
  end;

begin
```

```
  write('x = '); readln(x);
  writeln('factorial = ', factorial(x):12)
end.
```

なぜこれがうまく行くか納得行きませんか? たとえば、5 の階乗を計算しようとすると、(2) のところで 4 の階乗を計算しようとして自分自身を呼び出す。堂々めぐりか? しかし、その中では 3 の階乗を、その中では 2 の階乗を、その中では 1 の階乗を呼び出す。ところが、1 の階乗は (1) の方へ来るので、ちゃんと堂々めぐりが止まって答えが出て、あとは段々元に戻ってくればいいだけになる (図 1)。よくできているでしょう?

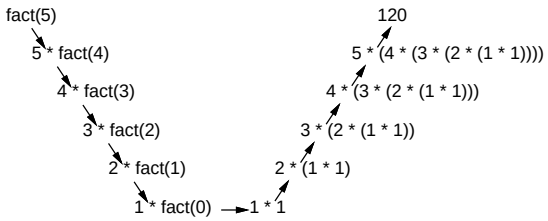

図 1: 階乗の再帰呼び出し

演習 2 上の例題を参考に以下のような関数を作り、メインプログラムも適宜変更してその関数を動かし結果を表示せよ。

- $1^2 + 2^2 + 3^2 + \cdots + N^2$ を計算する関数 sqsum。ただし、

$$sqsum(n) = \begin{cases} 0 & (n = 0) \\ n^2 + sqsum(n-1) & (o.w.) \end{cases}$$

- 組合せを計算する関数 comb。ただし、

$$comb(n, r) = \begin{cases} 1 & (r = n, r = 0) \\ comb(n, r-1) + comb(n-1, r-1) & (o.w.) \end{cases}$$

1.3 手続きと変数パラメタ

さて、関数では値が 1 個しか返せないのは明らかである。では、2 つ以上値を返したい時はどうすればいいか? また、「2 つの変数の値を交換する手続き」のようなものはどうすればいか? 一見、次のようなもので良さそうに見える。

```
procedure swap(a, b: real);
var c: real;
begin
  c := a; a := b; b := c
end;
```

しかし、この手続きを「`swap(x, y)`」のように呼び出しても変数 x と y の値を交換することにならない。というのは、これまで学んで来た書き方ではパラメタはすべて「手続きや関数に値を渡す」だけで、その内部でパラメタ変数に値を入れても外側であてはめた変数 (ここでは x や y) には影響が及ばないからである。実はあてはめた変数に影響が及ぶようにするには、次のように直せばよい。

```
procedure swap(var a, b: real);   ←ここだけ違う!
var c: real;
begin
  c := a; a := b; b := c
end;
```

Pascal ではこれまでの (値が渡せるだけの) 書き方を「値パラメタ」と呼び、ここで示した、あてはまた変数にも影響が及ぶ書き方を「変数パラメタ」と呼ぶ。なお、値パラメタには任意の式があてはめられるが、変数パラメタには変数しかあてはめることができない。つまり、

```
swap(x1, z);        ←○
swap(x + 1.0, z);   ←×
```

ここでちょっとだけお絵描きの問題に戻ろう。といっても「色」の話である。つまり、これまでのお絵描きは絵の具が「不透明」だったが、これを透けて見えるようにするにはどうしたらいいだろう? つまり「色セロハン」のようにするわけである。3 原色の明るさ α_R、α_G、α_B(ただし α_c はすべて 0〜1 の間の数とする) のセロハンに 3 原色が β_R、β_G、β_B(β_c もすべて 0〜1) のセロハンを重ねてできる色は $\alpha_R\beta_R$、$\alpha_G\beta_G$、$\alpha_B\beta_B$、 つまり各原色ごとの掛け算になる。(なぜかよく考えてみよう。) この計算をする手続きを次に示す。

```
procedure subcolor(r1,g1,b1,r2,g2,b2: integer; var r,g,b: integer);
begin
  r := r1*r2 div 255; g := g1*g2 div 255; b := b1*b2 div 255
end;
```

つまり、2 つのセロハンの色の 3 原色を受け取り、重ねた結果できるセロハンの色を返す。255 で割っているのは、数値の範囲が 0〜1 でなく 0〜255 だから。返す方の 3 原色については、手続き内で値を書き込むとそれが外に伝わって欲しいのだから、変数パラメタでなければならない。これを使うように point、fillcircle をそれぞれ変更したプログラムを示しておく。

```
program sam18d(input, output);
const xmax = 300;
      ymax = 200;
var i, j: integer;
    red, green, blue: array[1..ymax, 1..xmax] of integer;

procedure clearscreen(r, g, b: integer);
begin
  for i := 1 to ymax do
    for j := 1 to xmax do begin
      red[i,j] := r; green[i,j] := g; blue[i,j] := b end
end;

procedure writescreen;
begin
  writeln('P3 ', xmax:1, ' ', ymax:1, ' 255');
  for i := ymax downto 1 do
    for j := 1 to xmax do
      writeln(red[i,j]:1, ' ', green[i,j]:1, ' ', blue[i,j]:1)
```

```
end;

{ ここに subcolor の定義 }

procedure subpoint(x, y, r, g, b: integer);
begin
  if (x > 0) and (x <= xmax) and (y > 0) and (y <= ymax) then
    subcolor(red[y,x], green[y,x], blue[y,x], r, g, b,
             red[y,x], green[y,x], blue[y,x])
end;

procedure subfillcircle(x0, y0, l, r, g, b: integer);
var x, y: integer;
begin
  for x := x0 - l to x0 + l do
    for y := y0 - l to y0 + l do
      if (x-x0)*(x-x0)+(y-y0)*(y-y0) <= l*l then subpoint(x, y, r, g, b)
end;

begin
  clearscreen(255, 255, 255);
  subfillcircle(180, 60, 50, 255, 200, 200);
  subfillcircle(220, 60, 50, 200, 255, 200);
  subfillcircle(200, 95, 50, 200, 200, 255);
  writescreen
end.
```

演習 3 この「セロハン重ね」で何か絵を作ってみよ。たとえば同じ色のを何枚も重ねるとどうか? また、「セロハン重ね」の反対にスキー場の「カクテル光線」がある。セロハンは 3 色重ねると真っ黒になるが、カクテル光線は 3 色合わせると真っ白になりますね? それを計算する手続き addcolor も作って使ってみよ。[1]

1.4 再帰手続き

先の再帰関数と同じように、再帰手続き、つまりある手続きの中で自分自身を呼び出すようなもの可能である。まあ、例題を見て頂こう。

```
program sam18c(input, output);
var s: array[1..20] of char;

  procedure cswap(var x, y: char);
  var z: char;
  begin
    z := x; x := y; y := z
  end;
```

[1] ものの本にはこれらは「減色混合」「加色混合」と書かれている。ヒント: 加色混合は $\alpha_c + \beta_c - \alpha_c\beta_c$ を各色について適用して計算すればよい。

```
  procedure anagram(i: integer);
  var k: integer;
  begin
    if s[i+1] = ' ' then
      writeln(s)
    else begin
      anagram(i+1);
      k := i + 1;
      while s[k] <> ' ' do begin
        cswap(s[i], s[k]); anagram(i+1); cswap(s[i], s[k]); k := k + 1
      end
    end
  end;

begin
  write('s = '); readln(s); anagram(1)
end.
```

これは、例えば自分の名前を入れるとそのあらゆる入れ換えを打ち出すというものである。実行例を示しておこう。

```
% a.out
s = kuno
kuno
kuon
knuo
knou
konu
koun
ukno
...
```

なお、anagram(i) というのは i 文字目以降をあらゆるやり方で組み変えながら文字列 s を出力してくれる手続きである。つまり、メインプログラムは「1 文字目以降あらゆるやり方で組み変えながら出力してくれ」とだけ言えばいい。もちろん、この手続きの中身が「みそ」であるが…難しいですか? PAD を示そう。

まず、i 番目の「次の文字」がない (ということは $i+1$ 番目が空白) かどうか調べる。もし i 番目が最後の文字なら、組み変えのしようがないのでそのまま出力する。

そうでなければ、まず $i+1$ 番目以降をあらゆるやり方で組み変えて出力してもらう。それには自分自身をパラメタ $i+1$ で呼ぶだけでよい。さて、問題は残る場合であるが、これはつまり i 番目にたまたま今入っている文字以外の文字が来る場合、ということになる。そこで、k を $i+1$ から最後の文字まで順に変えながら、i 番目と k 番目を交換した上でそれぞれ「$i+1$ 番目以降の組み変え出力」を行えばよい。ただし、終った後再度 i と k を交換して元に戻しておかないとぐちゃぐちゃになる。

どうですか、だいぶ頭がウニになったのでは? しかし、このように「自分自身がもうできてると思って」使う、というのは慣れればとても強力な道具になる。では Pascal も最後だから、これをネタにお楽しみ問題も用意しておこう。

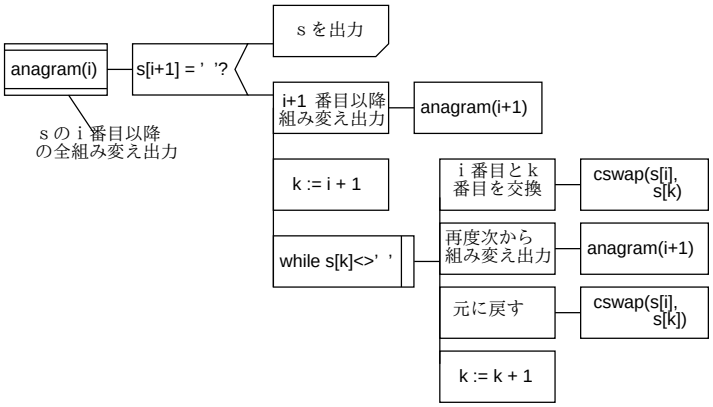

図 2: 手続き anagram の PAD

演習 4 この例題を次のように改良してみよ。

- あらゆる組合せが出て来ると大量なので、「ローマ字として読める」ものだけが出て来るように直せ。(ヒント: 書き出すところで、ローマ字として読めるかどうか調べて OK の時だけ書くように直せばよい。)
- 英語として読める、というのだとどうか?
- 入力に同じ文字が複数回現われると同じ出力がその回数だけ現われてしまう。これを避けるように直してみよ。

2 X-Window

2.1 ウィンドウシステムとは?

皆様の中には、この科目に入る前は「計算機」というとやたらと画面に文字の羅列が現われるもの、というイメージをお持ちだった方はいないだろうか? 実際、現在でも「画面が字ばっかり」の計算機は沢山使われている (現にプリンタサーバの画面はそうですね)。(図 3 左)。

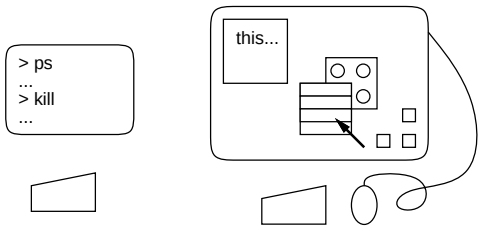

図 3: 文字端末とウィンドウシステム

しかし皆様が現在使っているシステムでは文字の他に様々な絵を表示させたり、沢山窓を開いたり、おなじ文字でも様々な形や大きさのものを使ったり、マウスで指示して色々な操作を行わせたりできますね? (図 3 右。) 計算機のソフトウェアのうちで、このような画面の活用を可能にしてくれている部分のことを「ウィンドウシステム」と呼ぶ。また、その上で使われる、絵やマウス入力を活用した対話の方式のことを「図的ユーザインタフェース」ないし GUI (Graphical User Interface の略) と呼ぶ。(「ウィンドウシステム」はソフトウェア、GUI はやり方ないしスタイルを指すので混同しないように。)

文字端末方式に比べた GUI の特徴は次の通り。

- 絵や画面の配置などを通じて、多量の情報をユーザに提示できる。
- 対話方式の工夫を通じてより使いやすくできる可能性がある。

世の中には、パーソナルコンピュータ用として Macintosh とか Windows となどのウィンドウシステムが広く使われている。また、Unix システムの上でも SunTools、NeWS、GMW など色々なウィンドウシステムが使われて来た。しかし現在では Unix システムにおけるウィンドウシステムとしては皆様が使っている X-Window(単に X とも書く) が標準として圧倒的に多く使われている。以下では X を前提として、その構造や機能について見てみよう。

2.2 X-Window の構造

Unix では、多数のプログラム (ないしプロセス) を同時に実行させられることを学んだ。X はその特徴を最大限に活かして作られている。つまり、皆様が見ている X の画面は多数のプロセスの共同作業の結果である。具体的には、これらのプロセスは次のように分類できる。

- X サーバ: これは、実際に画面に絵や文字などを表示したり、キーボードやマウスによる入力を受け付ける機能を担当する。
- ウィンドウマネージャ: これは窓の配置を制御したり、窓のまわりにかざりをつけたり、窓一覧を表示したり、背景でメニューを出したりする機能を担当する。
- クライアント: 「各種の窓」を作り出すプロセスであり、多くの種類がある。つまり、窓の種類とクライアントの種類は対応している。

ここで、X サーバはユーザは特に名前を知っている必要はない。ウィンドウマネージャは「twm」が皆様の使っているものである。クライアントとしては「kterm」「mule」「xpaint」「idraw」「oclock」「xbiff」「xv」などがある。大体わかりますね?

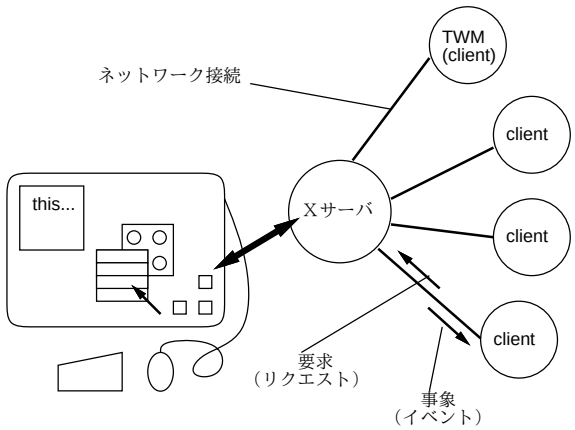

図 4: サーバとクライアント間のネットワーク接続

そして、X で特徴的なのは、画面を描いたり入力したりするのはすべて X サーバが行い、クライアントやウィンドウマネージャ(これも特別なクライアントだと考えてよい) は X サーバに「こういう窓を作ってください」「マウスボタンが押されたら教えてください」という風に頼むだけだということである。もちろん、X サーバはこれに応じて「窓ができて表示されたよ」「位置 XX でマウスの右ボタンが押されたよ」などと教えてくれる。ここでクライアントから頼む事柄を「要

求」ないし「リクエスト」、X サーバから通知してくれる事柄を「事象」ないし「イベント」と呼んでいる。これらのやりとりは、X サーバとクライアントの間にネットワーク接続を設定することで行われる (図 4)。

そして! 皆様の使っている「X 端末」というのは X サーバだけを動かしている計算機であり、皆様が使っている twm、mule、kterm などのクライアントはすべて「端末サーバ」と呼ばれる別の計算機で動いている。つまりネットワーク接続は文字通り別の計算機間の通信に使われている。

演習 5 背景で右ボタンを押して「Exit twm」を選び、twm を終らせると、窓のふち飾りがなくなり、窓が移動できなくなり、背景でのメニューも出なくなることを確認せよ。次に kterm の窓で「`fvwm`」を実行して fvwm というウィンドウマネージャを出し、窓操作などの方法の違いを探求せよ。最後に、fvwm を終らせ (終らせ方がわからない場合にはさっきの kterm の窓で`^C` を打てばよい)、「`twm &`」を実行してもと通りにせよ。

演習 6 次の各種クライアントから 2〜3 個選んで動かしてみよ。

- `xeyes` — マウスポインタを見つめる目玉ができる。
- `xcalc` — 電卓。
- `xmaze` — 迷路が表示される。
- `xcalendar` — カレンダーが表示される。
- `xpostit` — 「ポストイット」ができる。
- `puzzle` — 16 パズル。
- `xmag` — 画面の一部の拡大。
- `xfontsel` — 色々な文字フォントの選択。

なお、クライアントは

`-geom` 幅`x`大きさ`+X`位置`+Y`位置

(注意! 「geom」の直後以外は空白はあってはいけない。印刷の都合で空いて見えるので気を付けて!) というオプションを指定することで大きさが設定できるので、ついでに試してみるとよい。X 位置と Y 位置は画面左/上からの距離だが、直前の符号を「`-`」にすれば右/下からの距離になる。

2.3 ネットワーク透過と画面保護

ところで、あなたの X 端末から見たらどれか 1 つの端末サーバを特別扱いする理由は別にない。だから、あなたは「他の端末サーバで動くクライアントの窓」を作らせることもできる。それには「xon」という指令を使う。つまり

```
% xon マシン名 コマンド 引数 …
```

とやると、あなたが今使っているマシン以外で指定したコマンドを動かして窓を開かせることができる。何の役に立つかって? あるマシンがひどく混雑していたら、空いているマシンで窓を開いて作業した方が快適でしょう?

さて、クライアントが動くマシンを切り替えることができるのと同様に、クライアントを動かす時に「どの画面」というのを切り替えることもできる。ほとんどのクライアントは

```
% コマンド名 … -disp 端末名:0.0
```

と指定することで、指定した端末画面に窓を開くようにできる。では指定しなかったら? 実は、環境変数 (一種の設定変数)$DISPLAY にあなたの端末名が入っていて、クライアントはこれを参照してあなたの画面に窓を開いている。参照するには

```
% echo $DISPLAY
```

変更するには

```
% setenv DISPLAY 端末名:0.0
```

とする。(しかし不用意に変更しない方がよい。)

ところで、普段はあなたが動かしたクライアントはあなたの画面にしか窓が開けないようになっている。なぜか? そうして置かないと、他人があなたの画面に簡単にいたずらを仕掛けられるので危ないからである。でもちょっと試してみるのも面白い。Console の窓で「xhost +」を実行すると画面保護がはずれて、誰でもあなたの画面に窓を開けるようになる。元のように画面保護を掛けるには「xhost -」とすればよい。

演習 7 kterm の窓で「xon xss □ kterm」を実行して別のマシンの窓を開いてみよ (□は 18〜53 の適当な数。ただしそのサーバが動いていないこともあるので、その場合は別のにする)。先の窓とこの窓と両方で「ps ax」を実行して、プロセスの状況が全く違う、つまり別のマシンであることを確認せよ。

演習 8 隣の人と協力し、それぞれ「xhost +」で保護を外してから kterm の窓で「xclock -display xt □:0.0」で□に相手の端末の番号 (貼ったラベルに書いてある) を指定した xclock(ないし適当はクライアント) を動かし、確かに相手の画面に窓ができることを確認せよ。終ったら保護を元通りにしておくこと。

2.4 X の初期設定

さて、皆様が login した時どんな窓ができるかは、ホームディレクトリの下のファイル.xsession によって決まる。このクラスでの標準設定は次のようになっている。

```
oclock -transp -geom 110x110-224+0 -jewel red -hour pink -minute pink &
xbiff -geom 110x110-112+0&
xset m 8 2
xmodmap -e 'clear lock'
xrdb .Xdefaults
twm &
kinput2 -ccdef $HOME/.ccdef.kinput2 &
kterm -T console -n console -e /usr/local/bin/tcsh
```

つまり、このファイルは実はシェルスクリプトで、毎回手で打ち込む代わりに login 時にこれが自動実行されて oclock、xbiff、twm、kterm、kinput2(idraw その他用のかな漢字変換サーバ) などのクライアントを動かしているわけである。なお、Console 用の kterm の起動は必ず最後にないといけない。

それ以外の行の意味は次の通り。

- xset — マウスの感度を調整している。マウスの動きが敏感すぎると思う人は「8」をもっと小さい値にするとよい。

- xmodmap — 大文字ロックキーを無効にしている。間違って大文字ロックにする人が絶えないため。大文字ロックを使いたければこの行を消すとよい。
- xrdb — リソース設定用の.Xdefaults ファイルを読み込ませる。詳細は略 (WWW のページの Q&A を参照のこと)。

演習 9 Mule で`.xsession`を変更し、最初から動いていて欲しいクライアント (例えば電卓など?) の起動を追加せよ。kterm の窓から直接動かすのと同様に書けばよいが、行末に「`&`」を書くのを忘れないように。

A　本日の課題 12A

本日の課題は、演習 1(プログラムのみでよい) と演習 5〜9 のうちから時間中にやってみたものの結果報告です。試験も近いので残業したり全部やる必要はありません。興味を持ってやってみたことを報告すればよいです。レポートのタイトルは 12A です。アンケートは次の通り。

Q1. 簡単なものなら、関数や手続きを書けるようになりましたか? それとも「使うだけ」ですか?

Q2. X-Window について、知識/見聞が広がりましたか? また、それは今後役に立ちそうですか?

Q3. その他、感想や要望があればどうぞ。

B　次回までの課題 12B

次回までの課題は、演習 2〜4 のうちのどれか 1 個 (2 と 4 はその選択肢中の 1 つという意味) です。Pascal プログラムと実行例のみで結構です。レポート番号は 12B です。アンケートは次の通り。

Q1. ついに今回で Pascal プログラミングはおしまいになりました。2 度とやりたくないですか? 今後とも趣味としてやりたいですか? もし後者なら、どんなプログラムをやってみたいですか?

Q2. 結局、計算機ってどんなものだと思いましたか?

Q3. 課題に対する感想と今後の要望をお書きください。

課題は、次回授業開始時刻までに、レポートボックスに提出してください。

あと、せっかく皆様の絵を WWW にして全員で鑑賞できるようにしたので、人気投票を行います。WWW のページに掲示しますのでそれに従ってください。投票にも出席点をあげますので、ぜひどうぞ。そして、投票で上位になった絵の作者にはもちろんボーナス点をあげます。このクラス外の人の投票も歓迎しますので、宣伝してください。