「情報処理」1年文I/IIクラス11-12　＃13

久野　靖*

1995.1.29

1　本日の目標

長らくおつき合い頂きました「情報処理」も今回と次回でおしまいですね。演習をやって頂く内容は今回でおしまいで、次回はこれまでと毛色の違ったものを見て頂いてアンケートのみとなります。ですから頑張りましょう。

あと、まだレポート1Rをやってる人がいるでしょうから、次回までの課題は今回はなしです。次回は…次回のおたのしみ (次々回はないわけですけど)。なお1Rですが、〆切を過ぎても受領しますけれど、次回2/5を過ぎると東大の人でない私はもはや駒場に来ませんから、それより後には提出はできません。ご了承ください。

では、本日の内容は次の通り:

- シェルスクリプトについてもう少し。
- (前回の積み残し) 手続きの意味の復習と構造化グラフィクス
- 絵を動かす! (アニメーション) + シミュレーションのお話

2　シェルスクリプト (その2)

前々回の説明では、シェルスクリプトは単に「コマンドをファイルに入れておいて実行できるようにするもの」だった。しかし、前回CGIのところで出て来たシェルスクリプトはなんだかもう少し込み入ったことができていた。

実は! シェルスクリプトではPascalと同じように次のようなことができる:

*筑波大学大学院経営システム科学専攻

- 変数に値を入れたり、その値を参照できる。
- キーボードからデータを読み込める。
- 画面にデータを出力できる。
- 四則演算などの計算ができる。
- if による条件判断、while によるループができる。

つまり、シェルスクリプトでも Pascal と同じようにプログラムを作ることができる。たとえば、2 つの数の合計を取ってみよう。

```
#!/bin/sh
echo -n "数 a を入れてください> "
read a
echo -n "数 b を入れてください> "
read b
c=‘expr $a + $b‘
echo "$a と $b の合計は $c です。"
```

これを簡単に説明すると:

- `echo -n` は write、ただの `echo` は writeln に相当する。
- 文字列 (表示したい文字のならび) は「""」で囲む。
- `read` は Pascal の readln に相当する。
- 変数は前もって宣言する必要はなく、いきなり使える。
- 変数に値を入れるときは「変数名　= なんとか」による。(注意!「=」の前後に空白を空けてはいけない!)
- 値の計算は「‘`expr` 計算式‘」による。
- 変数の値を使うときには前に「`$`」をつける。これは「""」の中でも使える。

これをファイルに入れて実行可能にするだけで、すぐ動かせる。

ところで、普通のコマンドはいちいち read のようなものでデータを読み込んで来ることはあまりなく、「`add2 10 15`」のようにコマンドの後に続けてデータを指定する。そこで、シェルスクリプトではコマンドの後のデータを「`$1`」「`$2`」…のように番号で指定できるようになっている。これを利用して先のスクリプトを書き直すと次のようになる。

```
#!/bin/sh
a=$1
b=$2
```

```
c=‘expr $a + $b‘
echo "$a と $b の合計は $c です。"
```

実はもっと簡単にこうしてもよい。

```
#!/bin/sh
echo "$1 と $2 の合計は ‘expr $1 + $2‘ です。"
```

演習 1/☆ 上 3 とおりのどれか (またはすべて) の内容を「add2」というファイルに書き込み、「chmod ugo+x add2」で実行可能にし、「add2」(最初の場合) または「add2 5 10」などといって実行させてみよ。

あといちおう、if の例と while の例も挙げておく。まず 3 つの数の最大値から。なお、「gt」というのは greater than、つまり「>」のことである。

```
#!/bin/sh
max=$1
if [ $2 -gt $max ]
then
  max=$2
fi
if [ $3 -gt $max ]
then
  max=$3
fi
echo "$1 と $2 と $3 のうち最大なのは $max です。"
```

次は while の例。ne は not equal で Pascal でいえば「<>」のこと。

```
#!/bin/sh
a=$1
b=$2
while [ $a -ne $b ]
do
  if [ $a -gt $b ]
  then
    c=$a; a=$b; b=$c
  fi
  b=‘expr $b - $a‘
done
echo "$1 と $2 の最大公約数は $a です。"
```

このように、シェルスクリプトでも Pascal とほとんど同じようなプログラムが書けるうえ、加えてシェルスクリプトでは任意のコマンドが書けるのだからとっても便利。ではなぜ Pascal があるか? それは、配列や手続きなどの高度な機能を使ったり、実行速度が問題になる場合にはシェルスクリプトは向かないから。

演習 2/△ 上の if や while の例も演習 1 と同様にして実行させてみよ。

最後に、上の最大公約数を CGI スクリプトにしたものを示そう。これだとすべてシェルスクリプトで計算するので、Pascal のプログラムは呼ばなくてもすむ。なお、`$FORM_x` のようなものも実はシェル変数だったのね。

```
#!/bin/sh
PATH=/usr/local/httpd_ncsa/www/bin:/home/ユーザ/WWW:$PATH
echo 'Content-type: text/html'
echo ''
eval `cgiparse -form`
a=$FORM_a
b=$FORM_b
while [ $a != $b ]
do
  if [ $a -gt $b ]
  then
    c=$a; a=$b; b=$c
  fi
  b=`expr $b - $a`
done
cat <<EOF
<HTML>
<HEAD><TITLE>最大公約数の結果</TITLE></HEAD>
<BODY>
<H1>$FORM_a と $FORM_b の最大公約数は $a です。</H1>
</BODY>
</HTML>
EOF
```

演習 3/△ このスクリプト用の HTML のフォームを用意して動かしてみよ。

3　手続きと構造化グラフィクス (再録)

3.1　手続きの意味の復習

さて、ここまで 2 回くらい Pascal の「手続き」とそのバリエーションである「関数」についてやってきたが、重要なところなので再度別の角度から眺めて復習しておこう。

最初に説明した時には、手続きというのは「新しい命令を増やしている」、たとえば `fillcircle` という手続きを定義したら、その後ではいつでも `fillcircle` という命令が使えるようになっている、ということを説明した。しかし実際には、どんなことが起こっているのだろう?

ふつう、プログラムの実行は (ループはあるにせよ) 図 1 の左側のように、上から始まって一直線に進む。ところが、手続き (たとえば `fillcircle`) の命令が書いてあるところまでくると、実行の流れは右側の fillcircl と書かれたところ (つまり fillcircle の本体の各命令がある場所) にジャンプして、そこから実行が続けられる。そして、fillcircle の最後まで来ると…す

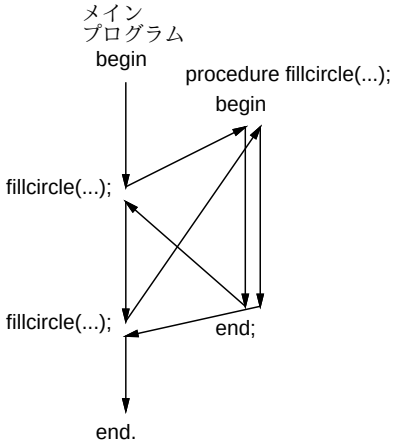

図 1: 手続きの実行のしくみ

ると、さっきジャンプしたところ (もちろん、ちゃんと覚えておくのだ) まで戻って、その続きを実行する。その後で、また別の `fillcircle` の参照箇所 (呼び出し、ともいう) に来るとまた fillcircle 本体のところにジャンプし、中を実行し終わるとさっきの続きに戻る。

このようにして、1 回書いたプログラムの部分 (fillcirlce の本体の命令) を何箇所からでも利用できる、というのが手続きの本質なのである。

3.2　手続きの階層と絵の構造

もちろん、このような呼び出しは 1 レベルではなく何段階にも渡って起きてよい。図 2 を見ると、main は A を 2 回呼び出し、A は B を 3 回呼び出し、B は C を 2 回呼び出すから…C の中は合計 12 回呼ばれることになる。

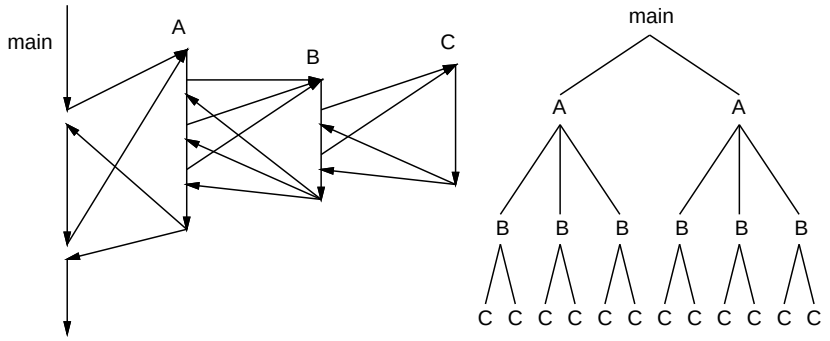

図 2: 複数レベルの手続き呼び出し

ところで、お絵描きの話に戻ろう。絵にはしばしば、階層構造が見られる。たとえば、自動車が描いてあったとすると、その自動車はタイヤ、窓、ボディといった「部品」から成っている。しかも、タイヤは同じものが 4 つついていたりする。このような、「全体としては 1 つの〜だが、中を詳しく見るといくつもの部品から成る」という構造が何レベルにも積み重なって、1 つの絵ができている (実は絵だけでなく現実世界の「もの」もだいたいそうなっている)。

そこで、このような絵を描くときに、1 つの「もの」の種類 (部品も「もの」である) を 1 つの手続きに対応させる。そうすれば、「もの」の種類の数だけ手続きを用意することで、かなり複雑な絵が描ける。どういうことか分かりますか?

たとえば、「自動車」を描く手続きの中では「タイヤ」を描く手続きを 4 回呼ぶ。すると、自動車が 3 台あった場合にはタイヤは 12 回、つまりずいぶん何回も描かなければならないのだけれど、先に延べた手続きの原理を応用すれば、単に自動車を描く手続きを 3 回呼べば、その中でそれぞれタイヤは 4 回呼ばれるからほっといても 12 個のタイヤが描けるわけだ。

このように、絵の「構造」を考えてそれに対応した手続きを用意して絵を生成する手法を「構造化グラフィクス」などと呼ぶ (こともある。私はグラフィクスの専門家ではありませんが、駒場の計算機の先生がたの半分くらいはこの手の専門家ですので)。

3.3 例題: 家と人と車

注意: 以下の例題の絵は、あくまでも (絵の下手な) 私が適当に考えたものなので、無理に真似する必要はないことをお断わりしておく。

さて、絵の中に車 (家や人でもいいけど) をいくつも描くとして、それぞれの大きさとか位置とか色は描くごとに調整したい。そこで、絵そのものは適当な単位 (unit) を基準としてその倍数でデザインしよう。図 3 の左上に「1」と書かれた線があるが、この長さの倍数の長方形、三角系、円などを組み合わせて車、家、人を作ってみた。それぞれの絵の原点は左下隅にあるものとする。

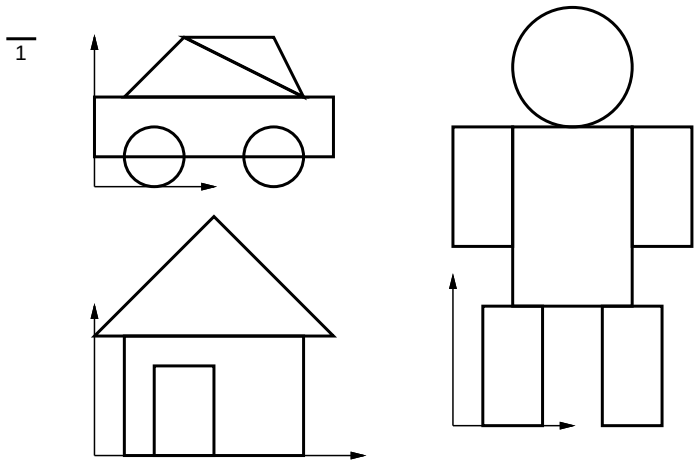

図 3: 家、人、車のデザイン

これを Pascal の手続きにしたものを次に示す。なお、ボディや胴体や家の本体は他の部分と同じ色合いで、ただしちょっと「暗く」なるように色の明るさを全体に減らしてある。

```
procedure house(x0, y0, u, r, g, b:integer);
begin
  fillrect(x0+u, y0, u*6, u*4, trunc(r*0.8),trunc(g*0.8),trunc(b*0.8));
  fillrect(x0+u*2, y0, u*2, u*3, r, g, b);
  filltriangle(x0, y0+u*4, x0+u*8, y0+u*4, x0+u*4, y0+u*8, r, g, b)
end;

procedure car(x0, y0, u, r, g, b:integer);
begin
  fillrect(x0, y0+u, u*8, u*2, trunc(r*0.7),trunc(g*0.7),trunc(b*0.7));
  fillcircle(x0+u*2, y0+u, u, r, g, b);
  fillcircle(x0+u*6, y0+u, u, r, g, b);
  filltriangle(x0+u, y0+u*3, x0+u*7, y0+u*3, x0+u*3, y0+u*5, r, g, b);
  filltriangle(x0+u*7, y0+u*3, x0+u*3, y0+u*5, x0+u*6, y0+u*5, r, g, b)
end;
```

```
procedure human(x0, y0, u, r, g, b:integer);
begin
  fillrect(x0+u*2, y0+u*4, u*4, u*6, trunc(r*0.6),trunc(g*0.6),trunc(b*0.6));
  fillrect(x0+u, y0, u*2, u*4, r, g, b);
  fillrect(x0+u*5, y0, u*2, u*4, r, g, b);
  fillrect(x0, y0+u*6, u*2, u*4, r, g, b);
  fillrect(x0+u*6, y0+u*6, u*2, u*4, r, g, b);
  fillcircle(x0+u*4, y0+u*12, u*2, r, g, b)
end;
```

なお、これら以外の手続きは先に出て来たので省略。これを呼び出して絵を描かせるだけのメインプログラムを示す。

```
begin
  clearcanvas;
  house(20, 40, 10, 200, 0, 150);
  human(120, 20, 7, 0, 150, 200);
  car(180, 30, 10, 100, 250, 100);
  writecanvas
end.
```

これでできた絵を図 4 に示す。

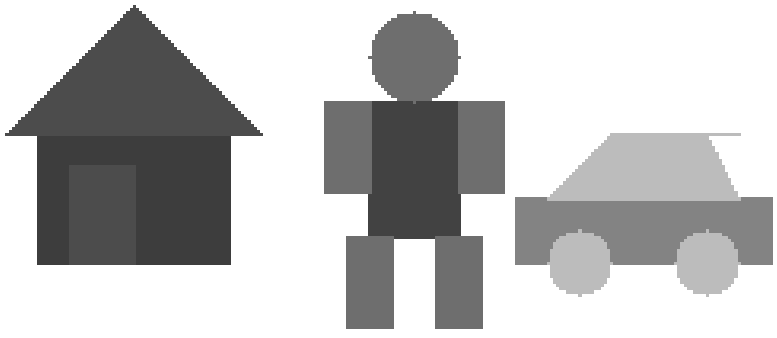

図 4: 家、人、車の絵

では次に、「家の戸口に人が立っている」のを作ろう。それには、家と人の手続きを呼び出す新しい手続き manandhouse を作ればよい。

```
procedure manandhouse(x0, y0, u, r1, g1, b1, r2, g2, b2:integer);
begin
  house(x0, y0, u*5, r1, g1, b1);
  human(x0+u*11, y0, u, r2, g2, b2);
end;
```

これを呼び出すメインプログラムは次の通り。

```
begin
  clearcanvas;
  manandhouse(20, 40, 3, 200, 150, 0, 100, 250, 0);
  manandhouse(100, 30, 4, 100, 50, 250, 200, 150, 0);
  car(200, 15, 10, 255, 100, 100);
  writecanvas
end.
```

絵も図5に示す。どうです、この調子でそれらしい絵が作れるでしょう? 最

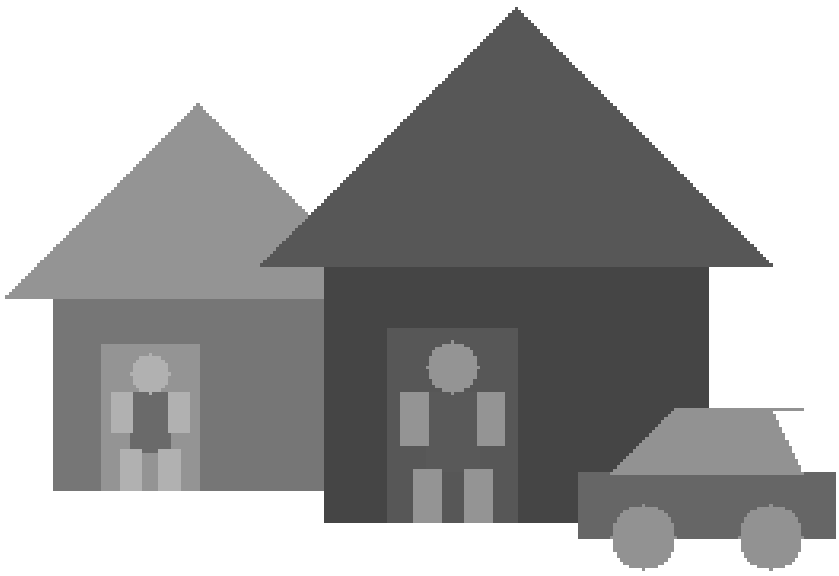

図 5: 家の戸口に人がいる絵

初は敷居が高いでしょうけれど、手続きはぜひ活用しましょう。なお、これらのプログラムのファイル (sam12b.p、sam12c.p) は例によって~kuno/pascal/の下にあります。

4　アニメーション

さて、ここまででずいぶんたくさんの絵を描いてきたが、これらの絵はすべて動かなかった。いよいよ、絵を動かしてみよう! その原理は簡単で、要するに映画のコマみたいに多数の絵をちょっとずつ違えて描き、それを連続して表示させるツールを作れば良い。実は、N 枚の GIF の絵をまとめて MPEG 形式というアニメーション用のフォーマットにするコマンド `mpeg_encode` と、できた MPEG 形式を画面で表示させるコマンド `mpeg_play` が既にあるので、N 枚の GIF の絵を用意すればあとはこれらのコマンドを使うだけでよい。

では、ちょっとずつ違えた多数の GIF の絵を Pascal プログラムで生成するにはどうしたらよいだろう? これまでは 1 つの PPM 形式のファイルを出力させて、それを

```
a.out | ppmtogif >t.gif
```

のようにして GIF ファイルにしていたから、1 つの GIF ファイルしかできない。

そこでこんどは、`a.out` を実行すると次のような出力が出るようにしよう。

```
ppmtogif <<EOF >t1.gif     ←※1
絵のデータ ....            ←※2
....
EOF                        ←※3
ppmtogif <<EOF >t2.gif
絵のデータ ....
....
EOF
以下くりかえし…
```

つまり、`t1.gif`、`t2.gif`、…、`t90.gif` をつくり出すような**シェルスクリプト**を生成して、その出力を

```
a.out | /bin/sh            ←※4
```

のようにしてシェルに渡す (コマンド行で`/bin/sh` を指定するので、これまでのスクリプトのような「`#!/bin/sh`」の行は不要である)。

では簡単な例題として、無重力空間で箱の中をボールがはずんでいるというものを示そう。

```
program sam13a(input, output);
var i: integer;
    x, y, vx, vy: real;
    red, green, blue: array[1..200, 1..300] of integer;

procedure point(x, y, r, g, b: integer);
begin
  if (1 <= x) and (x <= 300) and (1 <= y) and (y <= 200) then begin
    red[y,x] := r; green[y,x] := g; blue[y,x] := b
  end
end;

procedure fillcircle(x0,y0,r0,r,g,b:integer);
var i, j: integer;
begin
  for i := y0-r0 to y0+r0 do
    for j := x0-r0 to x0+r0 do
      if (i-y0)*(i-y0) + (j-x0)*(j-x0) <= r0*r0 then point(j,i,r,g,b)
end;

procedure clearcanvas;
var i, j: integer;
begin
  for i := 1 to 200 do
    for j := 1 to 300 do begin
red[i,j] := 255; green[i,j] := 255; blue[i,j] := 255
    end;
end;

procedure writecanvas;
var i, j: integer;
begin
  writeln('P3 300 200 255');
  for i := 200 downto 1 do
    for j := 1 to 300 do
      writeln(red[i,j]:1, ' ', green[i,j]:1, ' ', blue[i,j]:1)
end;
```

```
begin
  x := 150.0; y := 100.0; vx := 5.0; vy := 10.0;
  for i := 1 to 90 do begin
    writeln('ppmtogif <<EOF >t', i:1, '.gif');
    clearcanvas;
    fillcircle(round(x), round(y), 30, 255, 0, 0);
    writecanvas;
    writeln('EOF');
    x := x + vx; y := y + vy;       ←※6
    if x <= 30.0  then vx := -vx;   ←※7
    if x >= 270.0 then vx := -vx;   ←※7
    if y <= 30.0  then vy := -vy;   ←※7
    if y >= 170.0 then vy := -vy;   ←※7
  end;
end.
```

手続き類はすべて学んだものばかりなので、メインプログラムだけ見ていただく。まず、ボールの中心位置を変数 x、y で表し、x 方向と y 方向の速度を vx、vy で表すものとして、これらに適当な初期値を与える。次に、i を 1…90 まで順に変化させながら次のことを繰り返す。

まず先の説明の※1 の行を出力し、カンバスをクリアしてからボールをあるべき位置に描き、PPM ファイル (※2 の部分) を書き出し、最後に※3 の EOF を書き出す。それから、ボールの次の画面の位置を計算すが、1 画面ぶんを単位時間と思えば単に x 座標や y 座標に vx、vy をそれぞれ足し込めばよい。ただし、ボールが画面からはみ出しそうになった時には、「はねかえる」ためにボールの速度を反転させる。

以上を次々に繰り返して i が 90 まで来たらおしまい。これをコンパイルして※4 のように動かせばよい。ただし 90 個の絵を作るのは結構時間が掛かります。

次にこれら 90 枚の絵から MPEG ファイルを作るために mpeg_encode を使う方法を説明する。それには、

```
PATTERN IBBPBBI
OUTPUT test.mpg
BASE_FILE_FORMAT PPM
INPUT_CONVERT giftopnm *
GOP_SIZE 99
SLICES_PER_FRAME 1
INPUT_DIR .
INPUT
```

```
t*.gif [1-90]   ←※5
END_INPUT
PIXEL HALF
RANGE 10
PSEARCH_ALG LOGARITHMIC
BSEARCH_ALG CROSS2
IQSCALE 8
PQSCALE 10
BQSCALE 25
REFERENCE_FRAME ORIGINAL
```

のような内容 (おまじないですねえ) を test.param というファイルに入れておいて、

```
mpeg_encode test.param
```

を実行すると、test.mpg という MPEG ファイルが作られる。ここで入力ファイルや絵の枚数を変更したければ※ 5 の行だけ直せばよい。

ところで、上の場合は「無重力」で「箱の壁は完全弾性体」だったが、たとえば重力があれば、y 方向の速度は絶えず減少するので、※ 6 の後に「vy := vy - 1.0」などと追加すればよい。また、普通の壁のようにはね返るときにエネルギーが減る場合には※ 7 のところで「vx := -0.9 * vx」などとすればよい。逆に-1.1 などを掛けるとはね返るたびにエネルギーが増す不思議な壁になる。

このように、計算機の計算の上で現実世界のことがらを「まねる」ことを「シミュレーション」という。シミュレーションでは、実際には起こり得ないこと (上のエネルギーが増す壁のようなもの) も試してみられるし、お金や長い時間が掛かって実地に観測するのが大変な事柄もやってみられるという利点がある。

演習 4/☆ 上の例題プログラムや test.param は

```
cp /home/kuno/pascal/sam13a.p sam13a.p
cp /home/kuno/pascal/test.param test.param
```

などとしてコピーして来られるので、コピーしてきた後上で説明した「重力」や「不思議なはね返り」などを入れてみよ。その後

```
pc sam13a.p
a.out | /bin/sh
(しばらく待つ)
```

```
mpeg_encode test.param
mpeg_play test.mpg
```

によりアニメーションを表示させてみよ。

演習 5/△ ボールの数を増やしたり、ボールの色合いがボールの速度に応じて変化する、その他の工夫を考案して入れてみよ。または、以下の「おまけ」のプログラムから改良してもよい。

5 おまけ

最後に、前の節で作った「家と車」をアニメーションにして動かしてみよう。部品が画面上でそのまま動くだけだと迫力がないので、まず各部品の形が変化するように工夫改良する。まず、家は屋根が上下するように直す。

```
procedure house(x0, y0, u, r, g, b, d:integer);
begin
  fillrect(x0+u, y0, u*6, u*4+d, trunc(r*0.8),trunc(g*0.8),trunc(b*0.8));
  fillrect(x0+u*2, y0, u*2, u*3+d, r, g, b);
  filltriangle(x0, y0+u*4+d, x0+u*8, y0+u*4+d, x0+u*4, y0+u*8+d, r, g, b)
end;
```

車は、前輪と後輪が反対方向に上下できるようにする。

```
procedure car(x0, y0, u, r, g, b, d:integer);
begin
  fillrect(x0, y0+u, u*8, u*2, trunc(r*0.7),trunc(g*0.7),trunc(b*0.7));
  fillcircle(x0+u*2, y0+u+d, u, r, g, b);
  fillcircle(x0+u*6, y0+u-d, u, r, g, b);
  filltriangle(x0+u, y0+u*3, x0+u*7, y0+u*3, x0+u*3, y0+u*5, r, g, b);
  filltriangle(x0+u*7, y0+u*3, x0+u*3, y0+u*5, x0+u*6, y0+u*5, r, g, b)
end;
```

人間は、「バンザイ」できるようにする (h=0 が通常、h=1 だとバンザイ)。

```
procedure human(x0, y0, u, r, g, b, h:integer);
begin
  fillrect(x0+u*2, y0+u*4, u*4, u*6, trunc(r*0.6),trunc(g*0.6),trunc(b*0.6));
  fillrect(x0+u, y0, u*2, u*4, r, g, b);
  fillrect(x0+u*5, y0, u*2, u*4, r, g, b);
  fillrect(x0, y0+u*6+h*u*4, u*2, u*4, r, g, b);
  fillrect(x0+u*6, y0+u*6+h*u*4, u*2, u*4, r, g, b);
  fillcircle(x0+u*4, y0+u*12, u*2, r, g, b)
end;
```

「家に入った人」は家の屋根とバンザイを両方制御できるようにする。

```
procedure manandhouse(x0, y0, u, r1, g1, b1, r2, g2, b2, d, h:integer);
begin
  house(x0, y0, u*5, r1, g1, b1, d);
  human(x0+u*11, y0, u, r2, g2, b2, h);
end;
```

で、メインプログラムではまず左から右に 2 台自動者が車をガタガタさせながら通過し、3 台目はなぜか家が屋根をガタガタさせながら通過するようにした。家が通過するときは背景の家の屋根もガタガタし、人間はその途中でバンザイする。

```
begin
  j := 1; d := 2;
  for i := 1 to 30 do begin
    writeln('ppmtogif <<EOF >t', j:1, '.gif'); j := j + 1;
    clearcanvas;
    manandhouse(20, 40, 3, 200, 150, 0, 100, 250, 0, 0, 0);
    manandhouse(100, 30, 4, 100, 50, 250, 200, 150, 0, 0, 0);
    car(310-i*15, 15, 10, 255, 100, 100, d); d := -d;
    writecanvas;
    writeln('EOF');
  end;
  for i := 1 to 30 do begin
    writeln('ppmtogif <<EOF >t', j:1, '.gif'); j := j + 1;
    clearcanvas;
    manandhouse(20, 40, 3, 200, 150, 0, 100, 250, 0, 0, 0);
    manandhouse(100, 30, 4, 100, 50, 250, 200, 150, 0, 0, 0);
    car(310-i*15, 15, 10, 0, 200, 250, d); d := -d;
    writecanvas;
    writeln('EOF');
  end;
  h := 0;
  for i := 1 to 30 do begin
    writeln('ppmtogif <<EOF >t', j:1, '.gif'); j := j + 1;
    clearcanvas;
    if i >= 10 then h := 1;
    manandhouse(20, 40, 3, 200, 150, 0, 100, 250, 0, d, h);
    manandhouse(100, 30, 4, 100, 50, 250, 200, 150, 0, d, h);
    house(310-i*15, 15, 10, 0, 250, 50, d); d := -d;
    writecanvas;
    writeln('EOF');
  end;
end.
```

このプログラム全体も「 kuno/pascal/sam13d.p」に入れてある。

A　本日の課題 13A

本日の課題は、演習 4 または 5 で作ったアニメーションを自分のホームページ以下に入れることです。つまり、「.mpg」ファイルを WWW ディレ

クトリに入れ、ホームページのどこかに

```
<A HREF="test.mpg">アニメーションだよん</A>
```

という感じでリンクを用意すればよいわけ。なお、今日はいいですが火曜日以降に新たに作った場合にはそのことをニュースグループ komaba.lectures95.jousho.kuno-2 に投稿してください。また、リンクの近辺に (ニュースではなく Web ページの中に!) 以下のアンケートの回答も入れてください。

- アニメーションをやってみてどうでしたか。また、以前に聞いた時「プログラムでお絵描きをする」ことに利点はないという意見が多かったですが、今あらためて考えるとどう思いますか。
- (あれば) 感想と要望。