「情報処理」1年文I/IIクラス11-12　＃10

久野　靖*

1995.12.18

0　本日の目標

年内の講義はこれでおしまいで、来年は1/8が開始、その次は(成人の日が月曜日なので)1/23ということになります。休講ではありませんのでよろしく。前回予告の通り、今回からはプログラミングの内容はすべてoptionです(が、聞いておくとレポート1Rのヒントになるでしょう)。そして今日の内容ですが、「プログラミング以外でも復習が必要」との声があるので、ファイル関係の復習をして、多少新しい内容を追加します。

- 「ファイル」について復習する＋多少の追加
- 「ディレクトリ」について復習する＋多少の追加
- 「ファイルの保護モード」について学ぶ

△　「プログラムによるお絵描き」で円や直線を描く方法について

1　ファイルとその役割/＊

1.1　ファイルの種別

「ファイル」というのは、「計算機の中の情報を入れる容れもの」のことでしたね? 「入れる内容」としては、既に学んだだけで次のようなものがある。

- テキスト(歌詞とか自己紹介とか電子メールの内容とか)
- Pascalプログラム
- WWWページの内容(HTMLコード)
- 実行可能なプログラム(`a.out`ファイル)

*筑波大学大学院経営システム科学専攻

- 絵のファイル (GIF ファイル)
- 絵のファイル (PPM ファイル)
- PostScript ファイル (idraw の出力ファイル)

これらはすべて計算機の中では「ファイル」として同じように扱う。我々の日常では、CD とかフロッピーとかレポート用紙とか本とか、さまざまな「もの」はさまざまな形をしているから、間違えようがないが、計算機の「ファイル」の場合は全然違うものでもほとんど同じような姿なのが問題なわけである。

ではどうするか? 答は、ファイルには名前をつけるので、「名前を工夫する」ことで区別するわけである。具体的には、ファイル名の末尾を次のようにつける。

テキスト	`.txt`
Pascal プログラム	`.p`
WWW ページの内容 (HTML コード)	`.html`
実行可能なプログラム	なし
絵のファイル (GIF ファイル)	`.gif`
絵のファイル (PPM ファイル)	`.ppm`
PostScript ファイル	`.ps`

UNIX もこの方法での区別をするので、名前のつけ方がこの通りになっていないと思ったように扱ってもらえないことがある。ただし、「テキスト」や「実行可能なファイル」は多少自由にしてよい。

ところで、これらのファイル種別のなかで、「内容が文字ばかりから成るファイル」のことを「テキストファイル」と呼ぶ。具体的には「テキスト」(当たり前!)、Pascal プログラム、HTML ファイル、PPM ファイル、PostScript ファイルがそうである。テキストファイルはすべて、Mule に読み込んで修正したり、画面に表示したりできる。

一方、テキストファイルでないファイルのことを「バイナリファイル」と呼ぶ。具体的には、実行可能なプログラムや GIF ファイルがそうである。これらのファイルを間違って Mule に読み込んだり画面に表示させたりすると、ぐちゃぐちゃなものが見えるだけでうまく操作できない。これらのファイルを役立てる方法はそれぞれ種類ごとに決まっている。つまり

- 実行可能なプログラムファイル→実行させる
- GIF ファイル→ WWW のページに入れたり xv を使ったりして表示させる

1.2 ファイルの種別の移り変わり

ここでいちばん混乱するのが、どのファイルをどうやって加工したらどのファイル種別になるか、ということであろう。その概要を図 1 にまとめてみた。まず、左側の四角く囲んであるのはすべて「テキストファイル」だから Mule を使って作成したり修正できる。Pascal プログラムは pc コマンドで実行形式に変換し、続いて実行する。実行すると (writeln などにより) 出力が出るが、それはやはり文字を書き出すわけだからテキストファイルである。先週はその一例として、PPM ファイルを出力したわけだ。

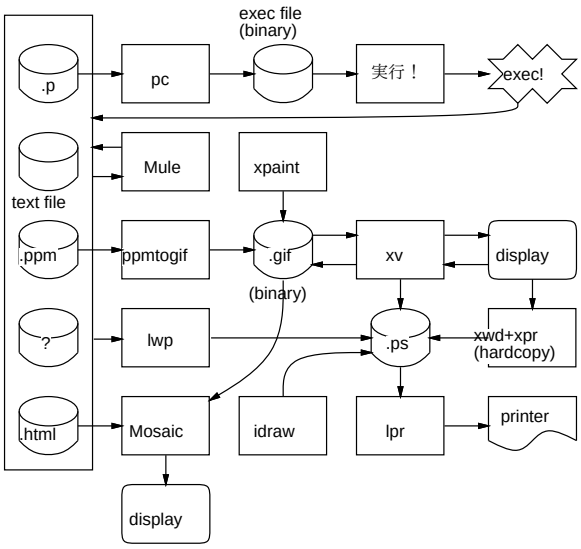

図 1: さまざまなファイル種別の移り変わり

PPM ファイルは ppmtogif を使えば GIF ファイルになる。また GIF ファイルは xpaint で作ることもできる。GIF を画面に表示するには xv を使う。xv は画面の任意の部分を切りとって GIF ファイルに保存したりもできる。また、WWW のページは HTML ファイルだが、その中に埋め込んだ絵は GIF だった。Mosaic などのブラウザはこれらを一緒にして画面に表示させる。

ところで、我々のシステムではプリンタは「PostScript プリンタ」であり、すべて PostScript(PS) 形式にしないと打ち出せない。打ち出す基本コマンドは lpr である。いつも使っている lwp はテキストファイルを PS に変換して、その中で lpr を呼び出して印刷している。xv も印刷させようとすると PS ファイルを作って lpr を呼び出す。おなじみ hardcopy も画面の内容を PS ファイルに変換して lpr を呼び出している。idraw の出力はそのまま PS ファイルだから lpr で打ち出す。

このように、ファイルの種類と移り変わりをマスターすればずっとUNIXでの操作の見通しがよくなると思う。いかがですか? ここまでに出てきたコマンド群の使い方をまとめておこう。

- Mule — 「`mule &`」。ファイルを読み込む (または新規作成) には `^X^F` ファイル `[RET]`。保存は`^X^S`。別のファイルに保存するには `^X^W` ファイル `[RET]`。
- pc — 「`pc` ファイル」。実行は `a.out`。
- xpaint — 「`xpaint &`」。
- ppmtogif —「`ppmtogif` 入力ファイル >出力ファイル」、「`a.out | ppmtogif >` 出力ファイル」
- xv — 「`xv` ファイル`&`」。
- lwp — 「`lwp -Pprinter` ファイル」。
- hardcopy — 「`hardcopy -Pprinter`」。
- lpr — 「`lpr -Pprinter` ファイル」。
- idraw — 「`idraw &`」。
- Mosaic — 「`xmosaic &`」。

2 ディレクトリの活用/☆

2.1 ディレクトリの木と絶対パス名/∗

これまで、ディレクトリというのは自分のファイルを整理するための「場所」という説明をしてきた。しかし、ディレクトリの役割はそれだけではない。また、ファイルが多数たまってきて、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もいることと思う。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
/home/ユーザ名	ユーザのホームディレクトリ

したがって、私や皆様のホームディレクトリは

```
/home/kuno
/home/g000001
```

などの場所にあり、そのサブディレクトリは

```
/home/g000001/Mail
/home/g000001/pascal                    ←※3
```

これらのディレクトリにあるファイルは

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

などとなるわけである。

実は各自のホームディレクトリは多数あるファイルサーバのどれかに割り当てられているので、上の説明は多少簡略化されているのだが、まあ普通はこう考えていてよい。つまり、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 2 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。

2.2 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

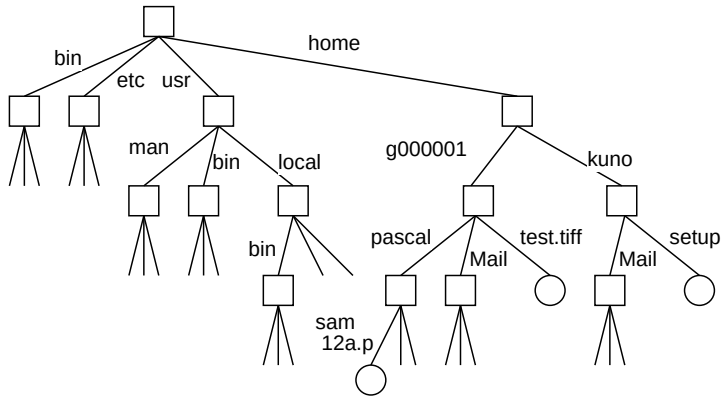

図 2: ディレクトリの木

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 woriknng deirecotry) コマンドで表示させられる。

```
% pwd
/home/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 3 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)。

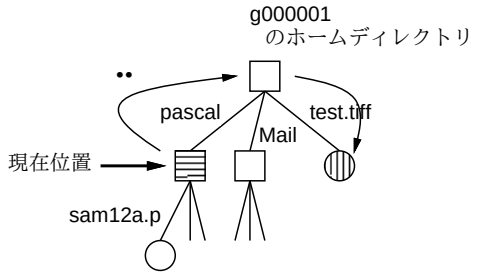

図 3: 「`..`」について

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 1 ☆ 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 2 ☆ 「`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 3 ☆ 私のホームディレクトリの下の方のどこかに「REPORT10A.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。

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 4 ☆ 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 5 ☆ 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。

4　プログラミングによるお絵描き (2)/△

4.1　長方形や円を描く

本来ならここで前回の演習問題の解説があるのだが、話の都合上順番を適当に入れ換えて説明する。そもそも、「2 重の for ループ」というのがわりと難しかったですか? たとえば

```
for i := 1 to 10 do
  for j := 20 to 24 do begin a[i, j] := 0 end;
```

というのがあったとすると、これは「i を 1, 2, 3, 4, 5, ...,10 と変化させながら、それぞれの場合について j を 20, 21, 22, 23, 24 と変化させながら、『a[i, j] := 0』を行なう」と読む。つまり具体的には

```
a[1, 20] := 0;
a[1, 21] := 0;
a[1, 22] := 0;
a[1, 23] := 0;
a[1, 24] := 0;
a[2, 20] := 0;
a[2, 21] := 0;
a[2, 22] := 0;
a[2, 23] := 0;
a[2, 24] := 0;
...
a[10, 23] := 0;
a[10, 24] := 0;
```

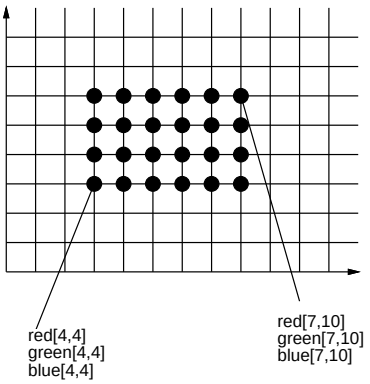

図 4: 塗りつぶした矩形

をやることになるわけだ。ところで、塗りつぶした長方形を描くということは、図 4 のように「i が 4〜7、j が 4〜10 のすべての点について色をこ

れこれにする」わけですよね。従って上のようなループを使えばそれだけでできる (以下簡単のため PAD を略します)。

```
program sam10a(input, output);
var i, j: integer;
    red, green, blue: array[1..200, 1..300] of 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;
  for i := 20 to 60 do              ←※1
    for j := 40 to 80 do begin
      red[i,j] := 80; green[i,j] := 255; blue[i,j] := 0
    end;                            ←※2
  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.
```

さて、円ではどうだろう? 円の場合は、図 5 にあるように、すべての点の色を変える代わりに**半径から距離 r 以内にある点の色だけ**変えればいいでしょう?

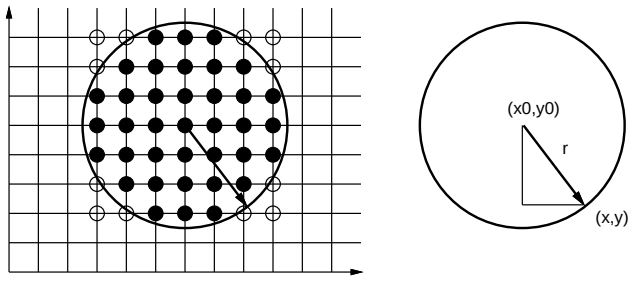

図 5: 塗りつぶした円

点 (x_0, y_0) と点 (x, y) の距離を r とすると、$r^2 = (x - x_0)^2 + (y - y_0)^2$ ですね? (これがわからんという人いますか? ピタゴラスの三角形の公式を思い出すように。) 従って、点 (x, y) が円内にあるという条件は

$$(x - x_0)^2 + (y - y_0)^2 \leq r^2$$

となる。そこで、プログラムの※ 1〜※ 2 のところを次のように直せば

```
for i := 20 to 60 do
  for j := 40 to 80 do
    if (i-40)*(i-40) + (j-60)*(j-60) <= 20*20 then begin
      red[i,j] := 80; green[i,j] := 255; blue[i,j] := 0
    end;
```

これで「点 (40,60) を中心とした半径 20 の円」ができるわけだ。

演習△ 長方形と円のプログラムを動かせ。

演習△ 円のプログラムを直して、塗りつぶした楕円を描くようにさせてみよ。

ヒント: 上の円の公式を変形すると次のようになりますね?

$$\left(\frac{x-x_0}{r}\right)^2 + \left(\frac{y-y_0}{r}\right)^2 \leq 1.0$$

ここで x の項にある r と y の項にある r を別々の値にすれば縦横の比率が違う円つまり楕円になるわけです。

4.2 手続きの導入

ここまで見てきて「何と面倒な! どこにどういう矩形を描くとかどこにどの半径の円を書くとかいう命令があればいいのに!」と思った人もおありかと思う。もしそのような命令があれば、円に矩形が重なった絵を描くという問題は図 6 のような PAD で済む。簡単でしょう?

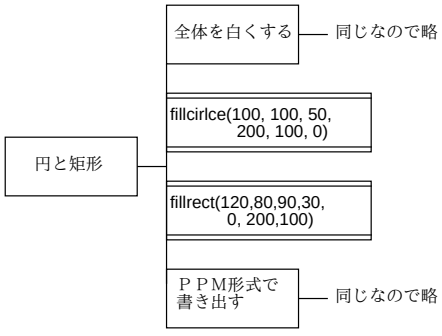

図 6: 円と矩形を描く

しかし、このヨコ線の入った PAD の箱は何だろう? 実はこれは、「自分で定義した命令を使う」ことを意味している。この、自分で定義する命令のことをプログラミング言語の世界では「手続き」という。

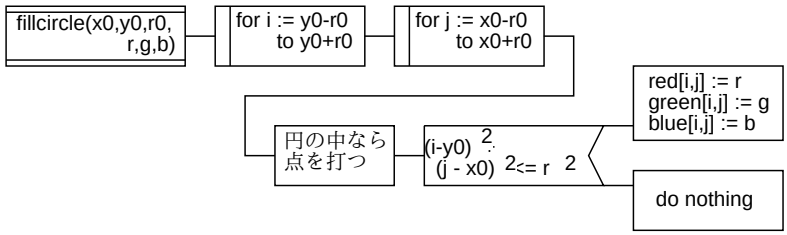

図 7: 円を描く手続きの PAD

そして、手続きの定義は別の PAD に分けて書く (図 7)。これを見ると、これまでの普通のプログラムの PAD とあまり変わらないが、根元の箱にヨコ線が入っていること (これが手続きであることを現す) と、そこに「x0, y0, r0, r, g, b」などと書いてあるところが違う。これらの名前は「パラメタ」と呼ばれ、手続きが呼び出される時に渡された値が入るようになっている。

つまり、図 8)。のように主プログラムで `fillcircle(...)` と書いた場所 (呼び出し場所) のかっこ内に記した最初の数値 100 は、手続き fillcircle の中ではパラメタ変数 x0 に入っている。従って、for や if の箱の中で x0 と書いてある場所では、100 という値が使われることになる。

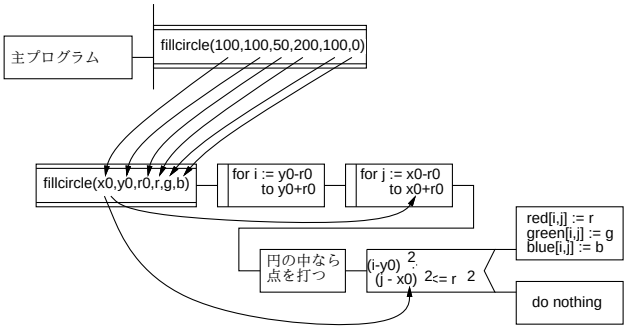

図 8: 手続きとパラメタ

なぜこういうものがあるか分かりますか? それは、「円を描く命令」を定義した後で使うとき、そのつど「どこにどの大きさや色で描く」というのを取り替えて使いたいからである。決まった位置にしか引けない命令があっても、あまり嬉しくないでしょう?

次に、手続きを Pascal にするやり方を説明しよう。今度はさっきと逆に、手続きを定義する方から説明する。手続き定義は次の形をしている。

`procedure` 手続き名 (変数名,...:型; … 変数名,...:型);

```
var 変数宣言; …
begin 文; 文; … 文 end;
```

ただしパラメタがない時はかっこも不要である。見て分かるように、手続きというのはちょうどプログラム全体のミニチュア版のようなものである。(PAD でもそうだったでしょう?)

そして、手続き定義はプログラムの「変数宣言」と「begin」の間に入れることになっている。なお、プログラム全体の変数宣言にある変数 (広域変数と呼ぶ) はどこででも (手続きの中でも主プログラムでも) 使えるが、手続きのところで宣言した変数はその手続きでしか使えない。それなら全部広域変数にした方が簡単? それをやるとプログラムが少し大きくなるとごちゃごちゃになる。できるだけ手続きの方で宣言する方が何かとうまく行くものである。では、先の PAD に対応する Pascal プログラムを示そう。

```
program sam10c(input, output);
var i, j: integer;
    red, green, blue: array[1..200, 1..300] of integer;

procedure fillrect(x0,y0,w,h,r,g,b:integer);
var i, j: integer;
begin
  for i := y0 to y0+h do
    for j := x0 to x0+w do begin
      red[i,j] := r; green[i,j] := g; blue[i,j] := 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 begin
        red[i,j] := r; green[i,j] := g; blue[i,j] := b
      end
end;

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;
  fillcircle(100, 100, 50, 200, 100, 0);
  fillrect(120, 80, 90, 30, 0, 200, 100);
  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.
```

長くなっただけで、ちっとも嬉しくないって? よく考えてみてほしい。もっと複雑な絵を書く場合でも、fillcircle や fillrect という命令の呼び出しをその分増やすだけで済むのは嬉しいでしょう?

4.3 配列のチェックをする手続き

ところで、前回も話したが、配列で添字の範囲を間違えると…つまり、「1..200」という範囲を指定したのに 210 とか-5 のように範囲から外れた値を添字に使うと「Illigal Instruction」「IOT trap」「bus error」などわけの分からないメッセージを出してプログラムが停止してしまう。これを避けるのに、もちろん正しいプログラムを書けばいいのだが、人間は間違いを犯して当たり前だからプログラムでもちょっと手当をしておこう。つまり、1 つの点の色を設定するときに、予め添字の範囲に入っているか確認して、入っていなければ何もしないという手続き point を作り、fillcircle や fillrect はこれを呼ぶようにする。

```
program sam10d(input, output);
var i, j: integer;
    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 fillrect(x0,y0,w,h,r,g,b:integer);
var i, j: integer;
```

```
begin
  for i := y0 to y0+h do
    for j := x0 to x0+w do point(j, i, r, g, b)
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
  clearcanvas;
  for i := 1 to 10 do fillrect(250-i*30, 20+i*20, 100, 100, 0, 0, 55+20*i);
  for i := 1 to 10 do fillcircle(20+i*15, 50+i*10, 20+i, 0, 255-i*10, 0);
  writecanvas;
end.
```

このプログラムを動かして見ると、図 9 のような絵になる。矩形は画面か

ら「はみ出て」いるが、問題なく描けているでしょう? なお、ついでながら最初に画面を真っ白にするのと最後に書き出すのも手続きにした。この方がメインプログラムが何をやっているのかよく分かるでしょう?

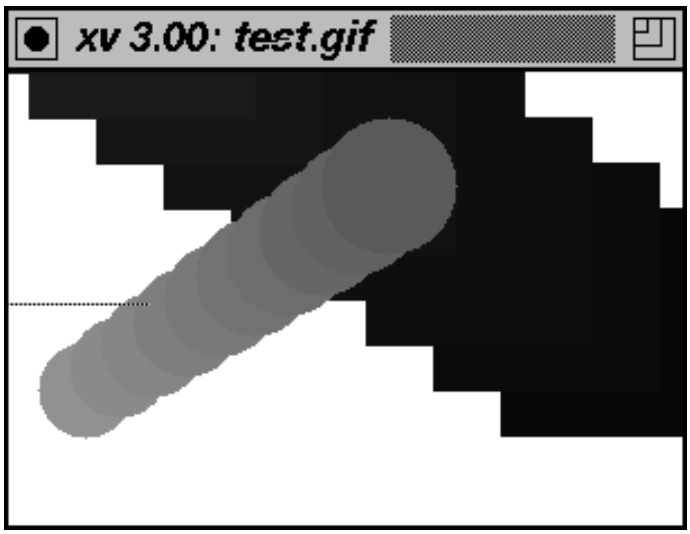

図 9: 枠からはみ出ていく絵

演習△ このプログラムは~kuno/pascal/sam10d.p に入っているので、コピーして構わない (打ちたければ打ってもいいけど)。まずそのまま動かし、その後絵を変更して動かせ。

4.4 任意の 2 点間を結ぶ直線の描き方

さて、最後に絵を描く時やっぱりいるだろうから、直線を引くのもやっておこう。前回の例題だと「ちょうど 45 度の斜め線」しか描けなかった。一般に、任意の点 (x_1, y_1) から (x_2, y_2) まで線を引くにはどうしたらよいだろう? 難しいのは、直線を「点の集まり」で現し、しかも「点」が打てるのは格子上の決まった位置 (ピクセルのあるところ) だけなので、図 10 に示すように「はんぱな所」を直線が通っている時にはその近くにある格子上に点を打たなければならないということである。

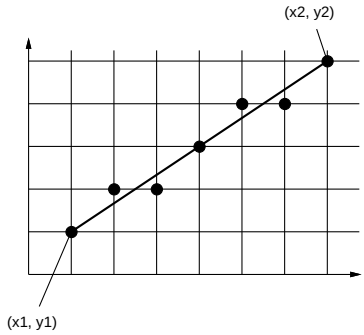

図 10: 点の集まりで直線を描くには

その解としてはいろいろな考え方があるが、例えば次のように考えてみよう。まず、(x_1, y_1) から (x_2, y_2) までの線分上のさまざまな点はどうやって表せるだろう? たとえば「中点」だったら $(0.5x_1+0.5x_2, 0.5y_1+0.5y_2)$ ですね。3:1 で最初の点に偏った位置の点だと $(0.75x_1+0.25x_2, 0.75y_1+0.25y_2)$ 99:1 で最初の点のそばなら $(0.99x_1 + 0.01x_2, 0.99y_1 + 0.01y_2)$ になるわけだ。そうしてみると、これらすべての点の集まり (x, y) は次の式で現せる。

$$\begin{aligned} x &= (1-t)x_1 + tx_2 \\ y &= (1-t)y_1 + ty_2 \end{aligned}$$

ただし t は $0 \leq t \leq 1$ の様々な値である。

では、直線の絵を描くには t を小刻みに変化させながら (x, y) を計算して、それを四捨五入したところに次々に点を打って行けばよさそうである。どれくらいの刻みがいいか? それは、点と点の間が離れないようにすればいいのだから、直線の傾きが 45 度よりきつければ縦の 1 目盛りごと、そうでなければ横の 1 目盛りごとに点を打つと考えればよい。これを PAD で現すと図 11 のようになる (もちろん手続き)。これを Pascal に直すと次のようになる (注意! y0 や y1 という変数があると、Pascal コンパ

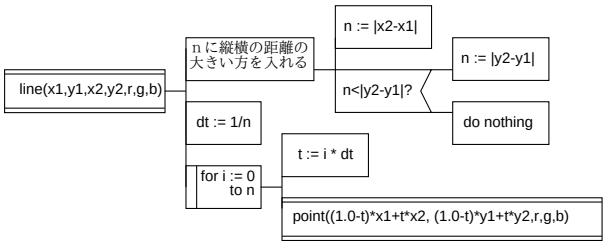

図 11: 任意角度の直線を描く手続きの PAD

イラによっては「ライブラリ関数と名前が一緒」という警告が出るが、無害なので無視する。ベッセル関数にそういうのがあるらしい)。

```
procedure line(x1,y1,x2,y2,r,g,b:integer);
var n, i: integer;
    t, dt: real;
begin
  n := abs(x2 - x1);
  if n < abs(y2 - y1) then n := abs(y2 - y1);
  dt := 1.0 / n;
  for i := 0 to n do begin
    t := i * dt; point((1.0-t)*x1 + t*x2, (1.0-t)*y1 + t*y2, r, g, b)
  end
end;
```

既に頭が痛いって? まあ、考えたくない人はあまり気にせず、とにかくメインプログラムの begin より前にこういうのを書いておけば line という命令が突然使えるようになるんだと思ってください。

A　本日の課題 10A

本日の課題は演習 3〜演習 5 の結果を報告することです。手書きでもいいですが、できれば Mule で打ち込んで lwp でプリンタに印刷してください。アンケートの内容は演習 3 で探したファイルに書いてあります。レポート番号は 10A です。では皆様よいお年を。

B　冬休みの課題 1R (再掲)

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。
2. 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、生成された絵を印刷出力したもの。
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 29 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。また、これと並行して、プログラムが生成した絵の GIF ファイルを「`WWW/report1r.gif`」というファイルに置くこと (互いに鑑賞できるようにしましょう)。