「情報処理」1年文I/IIクラス9-10　＃9

久野　靖[*]

1994.12.19

今回も一応、前々回の課題の解説があります。その後今日はここ2〜3回と順番を逆にして、まずLaTeXをやって、最後にPascalをやります。冬休み課題があるので、今日は「次回までの課題」はありません。皆様、よいクリスマスとお正月をお迎えください(ならレポートを取り消せというのはご勘弁を)。

1　前々回の課題の解説

まず2分探索による求根から。図1にPADを再掲しておく。そして、Pascalは次の通り。PAD

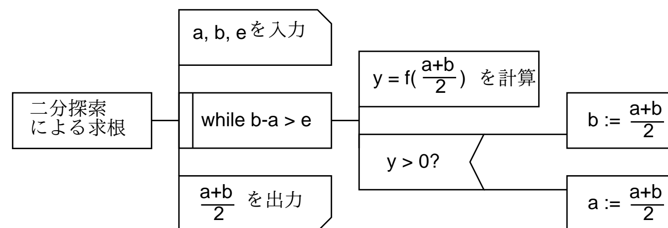

図 1: 根の2分探索のPAD

には x という変数はないが、Pascalにした時は一旦 $\frac{a+b}{2}$ を x とおいて、それを使う方が見やすいですね。

```
program sam11a(input, output);
var a, b, e, y, x: real;
begin
  write('a = '); readln(a);
  write('b = '); readln(b);
  write('e = '); readln(e);
  while b-a > e do begin
    x := 0.5 * (a + b);
    y := x * x - 2.0;   ←※
    if y > 0 then b := x else a := x
  end;
  writeln('root = ', 0.5 * (a + b):8:5)
end.
```

もちろん、「3の立方根」の場合は※のところを「`x * x * x - 3.0`」とすればいいわけだ。実行例も示しておこう。

[*]筑波大学大学院経営システム科学専攻

```
% a.out
a = 0
b = 2
e = 0.000001
root =  1.41421
```

演習 4 は、さすがに PAD は略して Pascal だけ示す。

```
program sam11b(input, output);
var n, i: integer;
begin
  write('n = '); readln(n);
  i := 1;
  while i <= n do begin
    writeln(i:4); i := i + 1
  end
end.
```

やはり for を使った方がずっと短い。

```
program sam11c(input, output);
var n, i: integer;
begin
  write('n = '); readln(n);
  for i := 1 to n do writeln(i:4)
end.
```

演習 5 は、c を除けばまあ例題の変形のようなものですが。まず最大値だが、これは例えば max という変数に「これまでのところの最大値」をいれておき、新しく入力した値と比べて max の方が小さければ max を新しい値に置き換えることを次々にやればいいわけだ (図 2)。ここで、最初

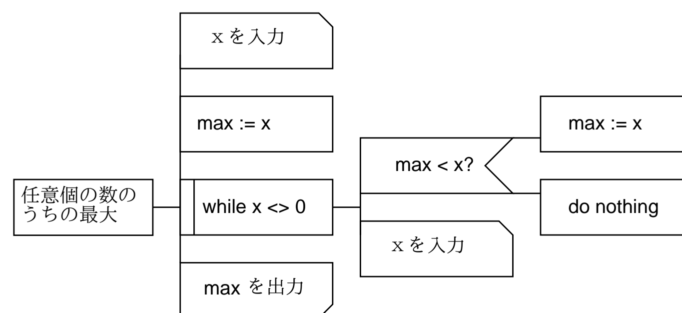

図 2: N 個の中の最大値の PAD

の max をいくつにするかが問題。例えば -10000 とかならいいと思うか? データが全部負の大きい値で、最大が -10001 だったらどうしよう? 答えは簡単で、最初の 1 個を読んだところで、とりあえずそれを最大だと思えばいい。Pascal は次の通り。

```
program sam11d(input, output);
var x, max: real;
begin
```

```
  write('x = '); readln(x); max := x;
  while x <> 0.0 do begin
    if max < x then max := x;
    write('x = '); readln(x)
  end;
  writeln('maximum = ', max:8:3)
end.
```

次は for を使えば簡単ですね。PAD も図 3 の通り簡単。ただ、「見やすく」するためには各出力の幅とかを結構こまめに調整しないとうまく行かない。

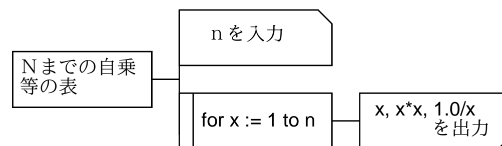

図 3: N までの値の自乗等の表

```
program sam11e(input, output);
var n, x: integer;
begin
  write('n = '); readln(n);
  for x := 1 to n do
    writeln('x =', x:4, '    x*x =', x*x:6, '     1.0/x =', 1.0/x:9:5)
end.
```

最後にフィボナッチ数列だが、これは while ループの方があっている。というのは、何番目がいくつかは計算してみないと分らないから。みそは、最近 2 つの値を x0 と x1 に取っておくのだが、新

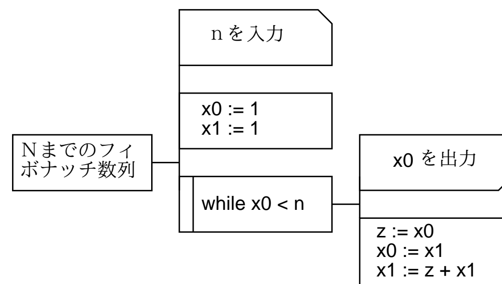

図 4: N までのフィボナッチ数列

しい値を計算する時、これまでの x1 を x0 に転送するが、その前に古い x0 の値を別の変数に保存しておかないと、新しい x1(というのは古い x0 と古い x1 の和) を計算できない、というところ。これは値の交換にちょっと似ている。

```
program sam11f(input, output);
var n, x0, x1, z: integer;
begin
  write('n = '); readln(n);
  x0 := 1; x1 := 1;
```

```
  while x0 < n do begin
    writeln(x0:4); z := x0; x0 := x1; x1 := z + x1
  end
end.
```

2 文書整形系 LaTeX

2.1 マークアップと WYSIWYG

さて、これまで皆様は毎週この配布資料を手にされてきたわけだが、これと他のクラスや科目で先生がたが配布される資料、さらには「補遺」までもがよく「似ている」と思いませんでしたか? 実はこれらはすべて「LaTeX」(ラテック、と読んでほしい) と呼ばれるシステムを駆使して作られている。

LaTeX では、文書ファイルの中に「ここは節の表題」「ここは脚注」「ここは箇条書き」といった「印」を埋め込んでおき、整形系と呼ばれるプログラムにそのファイルを処理させるときれいに整った出力ファイルができあがる、という方式を取っている。これを印つけ (Mark Up) 方式、と呼ぶ…というのは、WWW の HTML のところでいいましたね? HTML もやはりマークアップ方式だったが、LaTeX の場合は本なども作れるように細かい制御が可能で、その分印刷して美しい仕上がりが可能である (モノクロだけれど)。

なお、世の中の「ワープロ」と呼ばれる機器やソフトはマークアップとは対立する概念である見たまま方式 (What You See Is What You Get、略して WYSIWIG — ウィズィウィグと読んでね) に基づいている。見たまま方式では、計算機の画面に最終出力と同じ配置/大きさ/字形でテキストが表示され、(ちょうど idraw でやったように)「ここは大きい字」「ここは左を少しあけて」などマウス (かカーソルキー) とメニューで指示していく。

ここまで読んで、見たまま方式の方がよさそうだ…と思いましたか? そう、初心者にはその通りなのだが…しかし、idraw で絵を描くと分かるように、マウスとメニューであれこれ指定をして思い通りにするのは、実はひどく時間が掛かっていらいらする。一方、LaTeX では最初にコマンドを覚えてさえしまえば、打ち込むだけなので (タッチタイプできれば) 高速に文書が作れる。さらに、後から文書のスタイルを変えたり、一度作った文書を大幅改定して再利用したり、よそから持ってきた図や文書を取り込んで活用するのも容易である。というわけで、ぜひ LaTeX も体験しておいてほしい、というのがここで取り上げる趣旨である。

2.2 予備知識

まず TeX 系のシステムの大まかな構成を図 5 に示す。皆様はまず「なんとか`.tex`」というファイルを Mule などで作る。そして、「jlatex」コマンドを動かすと、JLaTeX がファイルに含まれる指示に従ってテキストを整形し、「なんとか`.dvi`」というファイルを作ってくれる。次に、その整形の様子がどんな具合かは「xdvi」コマンドを使えば画面で確認できる。(このようなプログラムをプレビュアとよぶ。いちいち紙に出していると面倒だし紙ももったいないので、できるだけプレビュアを活用すること。) 次に OK になったら「dvi2ps」コマンドで dvi ファイルから PS(ポストスクリプト) ファイルを作る。我々の手元のプリンタは PS プリンタなので、これを lpr コマンド (lwp ではない!!!) で打ち出せばでて来る。なお、PS ファイルを読み込むプレビュア `ghostview` もあるので、適宜使い分けるとよい。

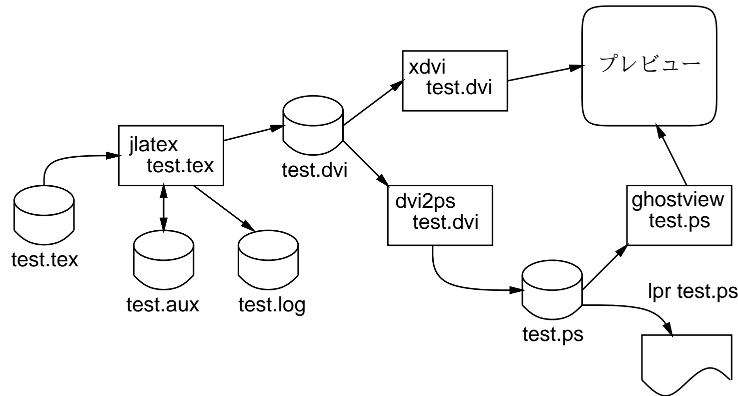

図 5: TeX システムの構成

2.3 LaTeX 文書の最初の例

まず、とりあえず最低限必要と思われるコマンドを一通り含んだ例を示す。鑑賞していただきたい。

```
\documentstyle[12pt]{jarticle}
\title{文書のタイトル}
\author{著者の名前}
\date{日付}
\begin{document}

\maketitle

\section{はじめに}

はじめに、documentstyle コマンドで基本的な活字の大きさ (ここでは
12 ポイント) とスタイル (ここでは日本語を使用した記事スタイル) を
指定します。

次に、文書のタイトル、著者、日付を指定します。日付は省略すると
整形した日の日付になります。これらの指定は指定を準備するだけで、
本当にタイトルができるのは「タイトル作成」コマンドのところです。

その次に「文書開始」というのがあります。ここからいよいよ文書が
はじまります。

\section{文書の本体}

文書の本体はタイトル、節タイトル、本文などがまざったものです。
普通は「タイトル作成」コマンドでまずタイトルを出します。その後
は「節タイトル」コマンドと本文のパラグラフが適宜混ざったものが
来ると思ってください。
```

```
パラグラフの中では適宜改行してください。多くの日本語ワープロと
違って、改行してもそこはパラグラフの区切りにはなりません。パラ
グラフの区切りには空行を入れてください。

もちろん、1つの節には複数のパラグラフがあるのが普通でしょう。

\section{おしまいに}

こうやって書いて来て、文書の最後になったら「文書おわり」を入れ
ます。これで全部おしまいです。

\end{document}
```

2.4　最低限必要なコマンドとコマンドの形式

先の例から判るように、最低限必要なコマンドは次のものである。

1. `\documentstyle`: 文書のスタイルとオプションを指定。
2. `\title`: 文書の表題を指定。
3. `\author`: 文書の著者を指定。
4. `\date`: 日付を指定。
5. `\begin{document}`: 文書の開始。
6. `\maketitle`: タイトルの出力。
7. `\section`: 節タイトルの出力。
8. `\end{document}`: 文書のおわり。

これだけだから、とにかく覚えてしまってほしい。ところで、これらの例から判るように、コマンドは

```
\コマンド名
\コマンド名{引数}
\コマンド名 [オプション,...]
\コマンド名 [オプション,...]{引数}
\コマンド名{引数}[オプション,...]
```

などの形をしている。3 番目以降の形はまだ現れていないが、後で使う。

2.5　その他 LaTeX テキストで重要なこと

もう 1 つ重要なことだが、これだけは覚えておいてほしい。「\」はコマンドの指定に使うので特別な意味を持つことは明らかですね。つまり、普通の文字として使うことはできない。これに加えて、「`{}$&#%^_~`」の各文字も特別な意味があるのでそのままでは普通の文字として使えない。これらの字をどうしても入れたければ入れる方法はあるが、当面はこれらを使わないで文書を書くように (使うと面倒なだけだから。)。

すこし後で、これらの字およびもっと色々な (キーボードにない) 字を使う方法もやる。あと全然違う話だが、いわゆる「半角カタカナ」や「全角空白」も TeX では扱えないので注意。

演習 7 たとえば「`sam1.tex`」というファイルに先の例にならった内容を打ち込め。ただし、タイトルは「レポート 9A」、名前はあなたの名前、section の名前は「今日のプログラム」「LaTeX の特徴」「感想と要望」とすること。パラグラフはとりあえず、それぞれ

- 今日の提出プログラムには〜を選びました。
- LaTeX の特徴は以下のことだと思います。
- おしまい。

とでも書いておく。

2.6 jlatex による整形

ではいよいよ、文書作成系を使ってみる。LaTeX ではマークアップつき文書は最後が「.tex」で終る名前のファイルに入れておく。そして「jlatex ファイル名」というコマンドによって整形すると、いろいろなメッセージが出はじめる…

```
% jlatex sam1.tex
This is JTeX, Version 1.52, based on TeX C Version 3.141
(sam1.tex
** JLaTeX ver.1.3......Isozaki
(/usr/local/libproj/tex/inputs/jarticle.sty
Document Style 'jarticle'. Released 4 Nov 1988
(/usr/local/libproj/tex/inputs/jart12.sty)
Conversion jlarge --> large etc.
) (sam1.aux)
Overfull \hbox (3.04321pt too wide) in paragraph at lines 24--28
[] 文書の本体はタイトル、節タイトル本文などがまざったも
 の で す。 普 通 は 「タ イ ト ル 作 成」
[1] (sam1.aux) )
(see the transcript file for additional information)
Output written on sam1.dvi (1 page, 3216 bytes).
Transcript written on sam1.log.
%
```

このようにたくさん出るが、とりあえず「普通に」終った場合には無視して結構。問題は、エラーで止まった場合だが:

```
% jlatex sam1.tex
This is JTeX, Version 1.52, based on TeX C Version 3.141
(sam1.tex
** JLaTeX ver.1.3......Isozaki
(/usr/local/libproj/tex/inputs/jarticle.sty
Document Style 'jarticle'. Released 4 Nov 1988
(/usr/local/libproj/tex/inputs/jart12.sty)
Conversion jlarge --> large etc.
) (sam1.aux)
! Undefined control sequence.   ←説明メッセージ!
l.9 sectoin                  ←ここに注目!
           {はじめに}
? x                          ←ここに注目!
No pages of output.
Transcript written on sam1.log.
%
```

このように説明メッセージが出てから「?」を表示して止まる。説明メッセージを読むと、だいたい理由がわかる。この例では「制御列 (コマンドのこと) が未定義」。その次 2 行で、エラーが出た場所が示されるが、これで見ると「sectoin」がタイプミスしていることが判る。

原因が判ったら「x[RET]」と打って処理を打ち切る。(最後まで進む方法もあるけど、1 箇所違っていたらそのためエラーの山になることが多いし、直して再度走らせても大したことはない。) そし

てMuleで問題のエラーを直してからまた走らせる。エラーがなくなると、この場合は「`sam1.dvi`」なるファイルができているはず。

2.7 プレビュー

dviファイルができたら、画面で整形結果を確認できる。そのためのプログラムをプレビュアという。ここでは`xdvi`を使い、「`xdvi sam1.dvi[RET]`」のように起動する。ちょっと待つと窓が開き、文書イメージが表示される。「Next」「Prev」などのところをつついて前後のページを移動し、OKなら「Quit」で終わればよい。ここでうまくない所が見つかったら再度`sam1.tex`を直して`jlatex`からやり直す。

2.8 印刷出力

プレビュアでOKになったらいよいよ紙に出す。それには

```
% dvi2ps sam1.dvi | lpr -Plw19
```

のように、dviファイルをPS(PostScript)に変換し、PSプリンタに送る。なお、次のように一旦PSファイルを作成して、次に打つという方法も可能だが、いらなくなったPSファイルは忘れずに消すこと。

```
% dvi2ps sam1.dvi >sam1.ps
% lpr -Plw19 sam1.ps
```

演習 8 先の文書をjlatexに掛け、エラーがなくなるまで直したらxdviでプレビューし、よくない所を直して完成したら印刷せよ。

演習 9 documentstyleのオプション`12pt`を`11pt`や`twocolumn`や`11pt,twocolumn`に直して整形し、結果がどう変化するか観察せよ。

なお、その他の細かい技については付録につけておくので、暇な時読んでぜひ活用していただきたい。

3 プログラムによるお絵描き(続き)と手続き

3.1 直線や円の描き方

さて、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6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。

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)までの直線上の任意の点(x, y)は次の式で現せる。

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

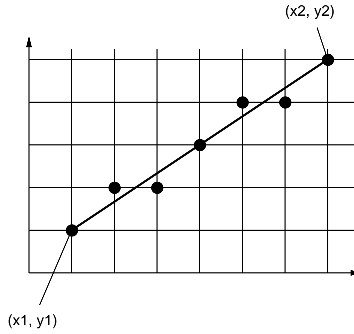

図 6: 点の集まりで直線を描くには

ただし $0 \geq t \geq 1$ とする。たとえば t が 0 のときは (x, y) は (x_1, y_1) に一致するし、t が 1 のときは (x_2, y_2) に一致する。

そこで、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 7 のようになる。これを

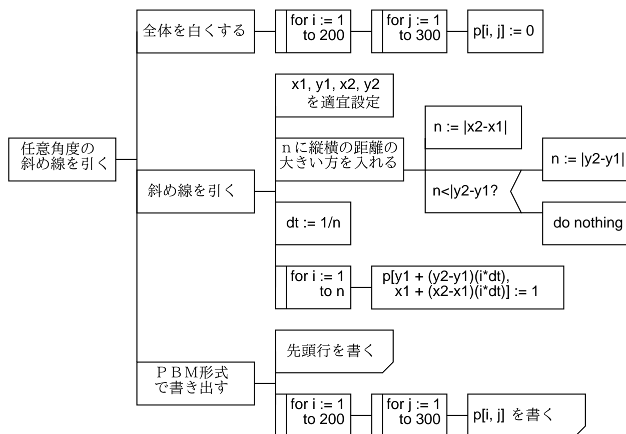

図 7: 任意角度の直線を描く PAD

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 コンパイラから「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)。

```
program sam14a(input, output);
var i, j, x1, x2, y1, y2, n: integer;
    p: array[1..200, 1..300] of integer;
    dt: real;
begin
  for i := 1 to 200 do
    for j := 1 to 300 do p[i,j] := 0;
  x1 := 10; y1 := 10; x2 := 250; y2 := 150;
  n := abs(x2 - x1);
  if n < abs(y2 - y1) then n := abs(y2 - y1);
```

```
    dt := 1.0 / n;
    for i := 0 to n do
      p[round(y1 + (y2 - y1)*dt*i), round(x1 + (x2 - x1)*dt*i)] := 1;
    writeln('P1 300 200');
    for i := 200 downto 1 do
      for j := 1 to 300 do writeln(p[i,j]:1)
  end.
```

ところで、最後の書き出す時の for 文が「downto」で逆向きに書き出すように変えてある。これは、これまでのやり方だと Y 座標の値が大きい方が下になって、指定が分りにくいから。

さて、円だとどうだろうか。中心 (x_0, y_0) で半径 r の円周上の点 (x, y) は

$$x = x_0 + r\cos t$$
$$y = y_0 + r\sin t$$

で現せる。あとは直線と同様にすればいいが、t のきざみ dt をどうするかを考える必要がある。要は点が飛ばないためには $r\sin dt$ が 1 ならいいわけで、$dt = \frac{1}{r}$ となる。[1]なお、Pascal では三角関数の角度はラジアンで与えることに注意。PAD 図を 8 に示す。Pascal では次の通り。

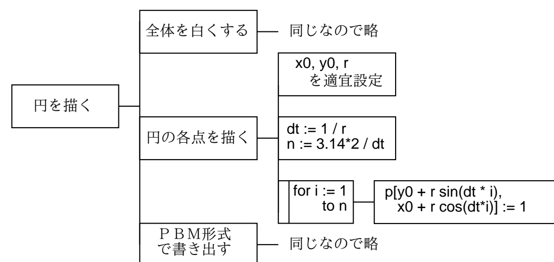

図 8: 円を描く PAD

```
  program sam14b(input, output);
  var i, j, x0, y0, r, n: integer;
      p: array[1..200, 1..300] of integer;
      dt: real;
  begin
    for i := 1 to 200 do
      for j := 1 to 300 do p[i,j] := 0;
    x0 := 150; y0 := 100; r := 80;
    dt := 1.0 / r; n := trunc(3.1416 * 2.0 / dt);
    for i := 0 to n do
      p[y0 + round(r*sin(dt*i)), x0 + round(r*cos(dt*i))] := 1;
    writeln('P1 300 200');
    for i := 200 downto 1 do
      for j := 1 to 300 do writeln(p[i,j]:1)
  end.
```

[1] dt はごく小さいから、$\sin dt \sim dt$ より。

3.2　手続きの導入

ここまで見てきて「何と面倒な! どこからどこまで線を引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、丸に斜め線を引いた絵を描くという問題は図 9 のような PAD で済む。簡単でしょう?

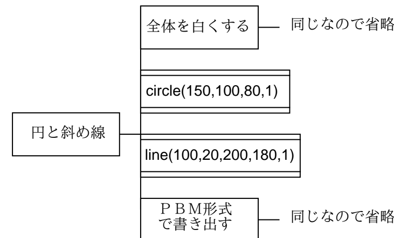

図 9: 円をと斜め線の PAD

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。

そして、手続きの定義は別の PAD に分けて書く (図??)。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「`c`」とか「`x1, y1, x2, y2`」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 (図 10)。

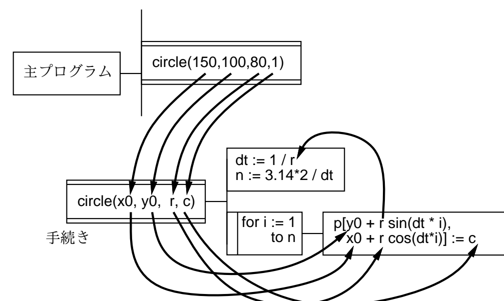

図 10: 手続きとパラメタ

なぜこういうものがあるか分かりますか? それは、「直線を引く命令」を定義した後で使うとき、そのつど「どこからどこへ引く」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, … 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 sam14c(input, output);
var i, j: integer;
    p: array[1..200, 1..300] of integer;

procedure line(x1, y1, x2, y2, c: integer);
var i, n: integer;
    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
    p[round(y1 + (y2 - y1)*dt*i), round(x1 + (x2 - x1)*dt*i)] := c
end;

procedure circle(x0, y0, r, c: integer);
var i, n: integer;
    dt: real;
begin
  dt := 1.0 / r; n := trunc(3.1416 * 2.0 / dt);
  for i := 0 to n do
    p[y0+round(r*sin(dt*i)), x0+round(r*cos(dt*i))] := c
end;

begin
  for i := 1 to 200 do
    for j := 1 to 300 do p[i,j] := 0;
  circle(150, 100, 80, 1);
  line(100, 20, 200, 180, 1);
  writeln('P1 300 200');
  for i := 200 downto 1 do
    for j := 1 to 300 do writeln(p[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、line や circle 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?

演習 10 このプログラムは`~kuno/pascal/sam14c.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 色のついた絵を出すには…

さて、そろそろモノクロの絵ではつまらなくなりましたか? 一応、カラーの絵の出し方も説明しておこう。ただし、課題でカラーを使うかどうかは自由である (どうせハードコピーではカラーは見えないし)。まず、カラーの場合にはファイル形式は PPM 形式を使う。その形は次の通り。

```
P3 幅 高さ 255
赤 緑 青
赤 緑 青
…
赤 緑 青
```

なお、「赤」「緑」「青」はそれぞれの点の 3 原色の強さを 0〜255 の範囲で現す。

ところで、これまでのプログラムでは「0」が白、「1」が黒だったが、それ以外の番号をそれぞれ適当な色だと思って使ってもいっこうに構わない。そこで、例えば「2」を赤、「3」を緑だと思って花の絵を描く手続きを作ってみた。

```
procedure flower(x0, y0, r: integer);
begin
  line(x0, y0, x0, y0 - r*6, 3);
  while r > 0 do begin
    circle(x0, y0, r, 2); r := trunc(r * 0.8)
  end
end;
```

これを使って、メインプログラムも次のようにする。

```
program sam14d(input, output);
var i, j: integer;
    p: array[1..200, 1..300] of integer;

{ ここに line、circle、flower の定義 }

begin
  for i := 1 to 200 do
    for j := 1 to 300 do p[i,j] := 0;
  for i := 1 to 8 do flower(30 + 30*i, 150+i+i, 20);
  writeln('P3 300 200 255');
  for i := 200 downto 1 do
    for j := 1 to 300 do
      if      p[i,j] = 0 then writeln('255 255 255')
      else if p[i,j] = 1 then writeln('0 0 0')
      else if p[i,j] = 2 then writeln('255 0 0')
      else if p[i,j] = 3 then writeln('0 255 0')
      else                    writeln('0 0 0')
end.
```

演習 11 演習 10 のプログラムを直して (手続き flower を追加し、メインプログラムを修正) 動かせ。さらに、花の色や形を変えてみよ。

演習 12 ところで、絵を描く時に「p[y, x] := 色番号」により直接配列に書き込んでいると、次のような弱点がある。

- 人間は X 座標、Y 座標の順に指定したいのに添字の方は逆なので分りにくい。
- 配列の添字の範囲を超えると「segmentation fault」とってプログラムが停止してしまう。

実際には、絵の範囲はいわば「窓」のようなもので、窓の外にかかる絵を描いた場合その部分だけ切れて見えない方が使いやすい。そこで、点を打つのも手続きにしてみよう。

```
procedure point(x, y, c: integer);
begin
  if (x >= 0) and (x <= 300) and (y >= 0) and (y <= 200) then p[y,x] := c
end;
```

これを手続き群の最初に入れ、line や circle をこれを呼ぶように直せ。つまり「p[y, x] := 色番号」を「point(x, y, 色番号)」に直せばよい。そうしておいて、画面の端に掛かる花を描いてみよ。

A 本日の課題 9A

本日の課題は、LaTeX の出力を提出してもらいます。文書のタイトルは「レポート 9A」とすること。その各 section は次のようにする。

- 「今日のプログラム」の節: 演習 10 または演習 11 の Pascal プログラムを挿入する。ただし HTML の<PRE>、</PRE>と同様、詰め合わされてしまうのを防ぐため

  ```
  \begin{verbatim}
  プログラム本体…
  \end{verbatim}
  ```

  のように「\begin{verbatim}」「\end{verbatim}」で囲む。その後に、どこが難しかったか (または難しくなかったか) を本文として記す。
- 「LaTeX の特徴」の節: LaTeX についてどう思ったか、またその特徴はどのあたりにあると思うかを自由に記す。
- 「感想と要望」の節: 感想と要望を自由に記す。

B 冬休みの課題 1R (再掲)

冬休みの課題は次の通り。

前回と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 (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。

C　LaTeX をもう少し活用するには

C.1　verbatim 環境

さて、これまでの所では字はきれいだが、何しろ表題と「地の文」しかないのであんまり整形したという気がしないかも知れない。そこで次に「万能選手」をお教えしよう。それは「verbatim 環境」といい、HTML で言えば`<PRE>`のことである。

```
\begin{verbatim}
内容の行 1
内容の行 2
...
\end{verbatim}
```

これによって次の様なものができる。

```
内容の行 1
内容の行 2
...
```

つまり、「`\begin{verbatim}`」から「`\end{verbatim}`」までの間にはさまった部分は「そのまんま」出る。このように、普通と違った扱いをする範囲は LaTeX では「`\begin{なんとか}`」と「`\end{なんとか}`」で囲む。そのような範囲のことを LaTeX では「環境」と呼ぶ。

verbatim 環境は文書にプログラム例などを載せたりするために作られたのだが、たとえば箇条書きもどきや簡単な表、たとえば

```
1) ああああああああああああああああああああああああああああ
   あああああああああああああああああああああああああああ
2) ああああああああああああああああああああああああああああ
   ああああ

+----------------+------------------+
|   .tex         | tex source text  |
+----------------+------------------+
|   .dvi         | format output    |
+----------------+------------------+
```

のようなものも出せる (全然「整形」になってませんが)。だから、ワープロ文書などを持って来て JLaTeX にする時など、追い込みのパラグラフでない部分はとりあえずすべて verbatim 環境にしてしい、あとで暇なときゆっくり直すという手が使える。ただし、TeX の世界では漢字と英数字の幅の比率が必ずしも 2:1 ではないので、漢字を入れるとその右側では縦にそろえるのは無理。(字下げは空白文字だけだから大丈夫。)

C.2 itemize と enumerate

このように verbatim は万能だが、箇条書きの文字数を自分でそろえるのはやっぱりいやだから、箇条書き用の環境をお教えしよう。まず番号なしの箇条書きは itemize 環境。

```
\begin{itemize}
\item itemize 環境は例によって begin-end で囲みます。その中では、
item というコマンドが項目の区切りに使えます。通常では項目はじめ
の印は中黒印です。

\item これに対して、enumerate の場合は環境の名前が違うだけで、
書き方は同じです。では何が違うかというと、中黒の代わりに 1 から
始まる番号が勝手につきます。

\item[***] itemize でも enumerate でも item コマンドに [] で囲んだオ
プションをつけると、中黒や番号の代わりに [] の中身が使われます。
\end{itemize}
```

これを整形すると次のようになる。

- itemize 環境は例によって begin-end で囲みます。その中では、item というコマンドが項目の区切りに使えます。通常では項目はじめの印は中黒印です。
- これに対して、enumerate の場合は環境の名前が違うだけで、書き方は同じです。では何が違うかというと、中黒の代わりに 1 から始まる番号が勝手につきます。

*** itemize でも enumerate でも item コマンドに [] で囲んだオプションをつけると、中黒や番号の代わりに [] の中身が使われます。

ちなみに、itemize を enumerate に変更すると次の通り。

1. itemize 環境は例によって begin-end で囲みます。その中では、item というコマンドが項目の区切りに使えます。通常では項目はじめの印は中黒印です。
2. これに対して、enumerate の場合は環境の名前が違うだけで、書き方は同じです。では何が違うかというと、中黒の代わりに 1 から始まる番号が勝手につきます。

*** itemize でも enumerate でも item コマンドに [] で囲んだオプションをつけると、中黒や番号の代わりに [] の中身が使われます。

itemize と enumerate は verbatim と並んでよく使う環境だからさっそく活用してほしい。これ以降に書くものは後回しでもいいです。

C.3 description 環境

itemize と enumerate は項目の区切りのところにあんまり長いものを書くようにできていない。項目ごとに題目をつけたい時は代わりに description 環境を使う。description 環境では item コマンドの後ろに必ず [] で囲んだ題目を置く。例えば上の例を description に変えて題目を付けると次の様になる。

itemize 環境: itemize 環境は例によって begin-end で囲みます。その中では、item というコマンドが項目の区切りに使えます。通常では項目はじめの印は中黒印です。

enumerate 環境: これに対して、enumerate の場合は環境の名前が違うだけで、書き方は同じです。では何が違うかというと、中黒の代わりに 1 から始まる番号が勝手につきます。

*** itemize でも enumerate でも item コマンドに [] で囲んだオプションをつけると、中黒や番号の代わりに [] の中身が使われます。

C.4 その他の環境

次は quote 環境で、これは item コマンドの不要な itemize といった感じ。

```
\begin{quote}
このように、quote 環境の中のテキストは両側とも引っ込めて整形さ
れるのでいかにも引用という感じになります。
\end{quote}
```

これを出力すると次の通りになる。

> このように、quote 環境の中のテキストは両側とも引っ込めて整形されるのでいかにも引用という感じになる。

あと、段落のつめ合わせをやめて行ごとに左/中央/右寄せするのが flushleft/center/flushright 環境。

```
\begin{flushright}
右右右
\end{flushright}
\begin{flushleft}
左左左左
\end{flushleft}
\begin{center}
中央中央中央
\end{center}
```

これは次のようになる。

右右右

左左左左

中央中央中央

C.5 文字サイズの変更

ここまででパラグラフの形などはだいぶ自由になると思うので、今度は個々の文字の方を見てみよう。まずは大きさの変更から。

```
{\huge あいうえお abcdefABCDEF}
{\LARGE あいうえお abcdefABCDEF}
{\Large あいうえお abcdefABCDEF}
{\large あいうえお abcdefABCDEF}
{あいうえお abcdefABCDEF}
{\small あいうえお abcdefABCDEF}
```

```
{\Small あいうえお abcdefABCDEF}
{\SMALL あいうえお abcdefABCDEF}
{\tiny あいうえお abcdefABCDEF}
```

なお、{}は出力には現れないけれども「範囲を区切る」のに使われ、その中で large などの指令を使って大きさを変える。範囲を出ると「元に戻」る。では上のを整形した結果。

あいうえお abcdefABCDEF
あいうえお abcdefABCDEF
あいうえお abcdefABCDEF
あいうえお abcdefABCDEF
あいうえお abcdefABCDEF
あいうえお abcdefABCDEF
あいうえお abcdefABCDEF

C.6 フォントの変更

といっても、日本語フォントは数がないので「ゴシックにする」しかない。それは dg(大日本フォントゴシック) コマンドによる。あと、英文フォントの方は tt(タイプライタフォント)、it(イタリック)、bf(ボールドフェイス) がある。

```
{あいうえお愛上尾 abcdefABCDEF<>?=} \\
{\dg あいうえお愛上尾 abcdefABCDEF<>?=}
{\tt あいうえお愛上尾 abcdefABCDEF<>?=}
{\it あいうえお愛上尾 abcdefABCDEF<>?=}
{\bf あいうえお愛上尾 abcdefABCDEF<>?=}
```

これを整形した結果は次の通り。dg は日本語、その他は英字のみにしか効かないのに注意。

あいうえお愛上尾 abcdefABCDEF¡¿?=
あいうえお愛上尾 abcdefABCDEF¡¿?=
あいうえお愛上尾 `abcdefABCDEF<>?=`
あいうえお愛上尾 *abcdefABCDEF¡¿?=*
あいうえお愛上尾 abcdefABCDEF¡¿?=

なお、<>は普通の英字フォントでは¡¿になってしまうのにも注意。tt フォントなら大丈夫。

C.7 制御文字とキーボードにない文字

そろそろ最初のほうで述べた、「使ってはいけない制御文字」の出し方をやっておく。

入力	結果
`$\backslash$`	\
`$\{$`	{
`$\}$`	}
`\$`	$
`\&`	&
`\#`	#
`\^{}`	ˆ
`_{}`	_
`\%`	%
`\~{}`	˜

このように、制御文字を入れるのは相当しんどいことがおわかりだろう。実はこれとは別にもう 1 つ、キーボードにある文字をそのまま出す方法がある。それは verbatim 環境のコマンド版。

```
\verb|\{}$&#^_%~abcABC|
```

つまり、verb コマンドの直後の文字が区切り文字になり、そこから同じ区切り文字が出て来るまでのあいだが verbatim 同様「そのまま」出る。

```
\{}$&#^_%~abcABC
```

ただしフォントは tt になる。また、これは万能ではなくてうまく使えない場所 (たとえば脚注の中) がある。実はそういう場所では verbatim もだめ。

さて、上の表で「\」を「`$\backslash$`」と書いていた。このように、「`$\`文字の名前`$`」という書き方を使うとキーボードにない文字 (ギリシャ文字とか数学記号とかその他色々) が出せる。これはきりが無いので参考書を見るように。例えば「野寺著、楽々LaTeX 第 2 版、共立」などが分りやすいだろう。

C.8 数式

実は`$`…`$`の中は「数式モード」といい、数式を書くのに便利な機能が揃っている (上の各種文字も数式用の記号だったわけ)。これも詳しくはきりがないので参考書を見ていただきたいが、代表的なものだけ挙げておく。

`$x^2$` → x^2。`^`は肩字をあらわす。
`$a_i$` → a_i。`_`は添字をあらわす。
`$\frac{a}{b}$` → $\frac{a}{b}$。分数。

なお、肩字や添字が 2 文字以上になる場合には`{}`で囲むこと。それ以外でも適宜グループ化する時にこれを使う。

ところで、`$`…`$`で作った数式は「追い込み」(段落の中に埋め込まれること) になる。そうでなくて、独立した行に式を書きたければ、displaymath 環境を使う。つまり、

```
\begin{displaymath}
x = \frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}
\end{displaymath}
```

とすると

$$x = \frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}$$

のようになるわけ。