「情報処理」1年文I/IIクラス11-12　＃11

久野　靖*

1995.1.8

あけましておめでとうございます。さて今年の最初は、「美しいレポートを作る道具」のお話からやろうと思います。本日の内容は次のとおり。

- 文書作成ツールの2つの流儀 — WYSIWYG とマークアップについてと、マークアップ型の文書作成ツール LaTeX について学ぶ。
- コマンドとスクリプトについて学ぶ。
- (option) サブルーチンの復習と、関数について学ぶ。

1　文書整形系 LaTeX

1.1　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

*筑波大学大学院経営システム科学専攻

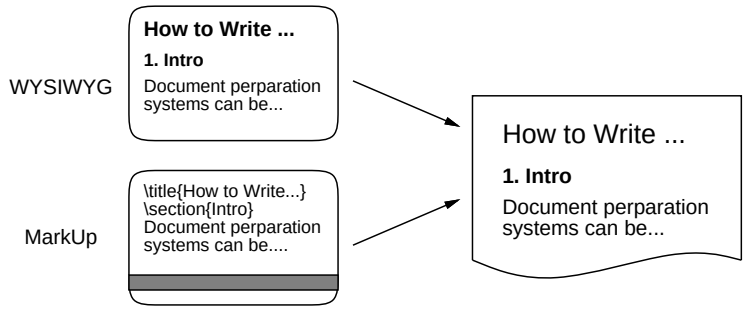

図 1: WYSIWYG とマークアップ

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。

1.2 予備知識

まず TeX 系のシステムの大まかな構成を図 2 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。

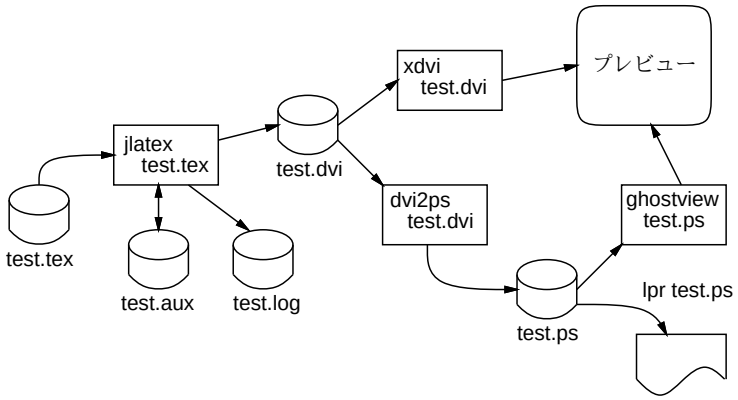

図 2: TeX システムの構成

1.3 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。

```
\documentclass[12pt]{jarticle}
\usepackage{graphicx}
  \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}
```

1.4 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。

1.5 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ができない。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 1 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 11A」、名前はあなたの名前、section の名前は「LaTeX の特徴」「UNIX のコマンド」「感想と要望」とすること。パラグラフ (本文) はとりあえず、それぞれ

- LaTeX の特徴は以下のことだと思います。
- UNIX のコマンドはいろいろあります。
- おしまい。

とでも書いておく。

1.6 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。

1.7 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。

1.8 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 -Plw11
```

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 2 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 3 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。

2 UNIX とコマンド

2.1 コマンドディレクトリとパス

さて、皆様はここまででずいぶん多くの UNIX コマンドを習って来たわけだが、ここでそもそもコマンドとは何か、という話をしておこう。まず、Pascal でプログラムを書き、pc コマンドで翻訳すると「`a.out`」というファイルができますよね。そして、そのプログラムを実行させるのには「`a.out`」と打ち込むのだった…これはどういうことだろう?

実は、「`mv a.out hello`」のようにしてファイルの名前を「`hello`」にしたとする。そして「`hello`」というと…そのプログラムが実行される! つまり、これまでコマンドだと思っていたのはファイル名だったのだ。だから、自分でファイルを作りさえすれば新しいコマンドはどんどん増やせるわけだ。

しかし、自分は `ls` とか `mule` とかいうファイルは持っていないが…と思われるかも知れない。それはつまり、共通してつかうコマンドについては、そのファイルは共通の場所 (ディレクトリ) に入れてあって共用するのが当然というものでしょう?

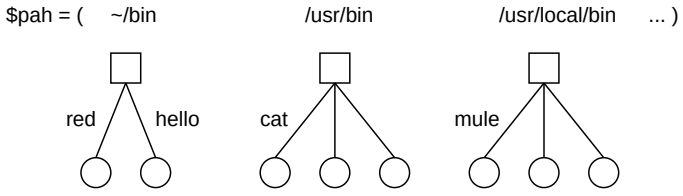

図 3: コマンドの探索

つまり、コマンドを打ち込むと、システムはいくつかのディレクトリを順番にさがして、コマンド名と同じ名前のファイルを見つけて、そのファイルに入っているプログラムを探してくれるのである (図 3)。UNIX ではこれらのディレクトリのリストを「探索パス」と呼んでいる。実際にどこのディレクトリのファイルかを知りたければ「`which` コマンド名」というコマンドを使って調べることができる。

```
% which cat
/usr/bin/cat          ← cat は/usr/bin にある
% which mule
/usr/local/bin/mule   ← mule は/usr/local/bin にある
%
```

そして、探してくれるディレクトリの一覧は

```
% echo $path
/home/kuno/bin ... ディレクトリが一杯表示される ...
```

によって表示できる。この中に「.」(カレントディレクトリ) があるから、それで「a.out」とか「hello」とかいうファイルがコマンドとして実行できたのである。

でも、新しいコマンドのファイルを作ったら、どの場所にいても (カレントディレクトリがどこになっていても) 同じように実行できて欲しいですね? それで、上のディレクトリ一覧を見ると「/home/自分のユーザ名/bin」というのが入っている。つまり、このディレクトリ (自分のホームディレクトリの下のディレクトリ bin) に入れておけばそのコマンドはどこででも実行できるわけ。

> 注意! ただし、新しいコマンドを追加したら「追加したよ!」とシステムに教えるため「rehash」というコマンドを 1 回実行してあげよう。

演習 4 まず、「Hello, how are you?」とだけ表示する Pascal プログラムを作り、動かせ。次に「mv a.out hello」を実行して、プログラムのファイル名を変更せよ。

a. hello というコマンドが実行できることを確認。
b. cd コマンドで別のディレクトリに移動すると実行できなくなることを確認。
c. さっきのディレクトリに戻ってから「mv hello ~/bin」でファイルを~/bin に移し、「rehash」を実行。
d. 今度は現在位置がどこでも hello コマンドが使えることを確認。
e. ファイル hello の名前を「ls」に変更して「rehash」を実行したら、その後 ls というコマンドはファイル一覧と自分のプログラムのどちらになるか? [1]

2.2 シェルスクリプト

これで Pascal でプログラムを書いて新しいコマンドを作れることは分かったけれど、そんなに高度なプログラムを沢山書けるものではない。そこで、もっとお手軽にコマンドを増やす方法も説明しておこう。

実は、コマンドを増やしたいというのは、まったく新しいコマンドを作るというより複数のコマンドを組み合わせたコマンドを (長々と打ち込む

[1] 実験が終わったらまた hello という名前に戻すか消すかしないと不便だと思うよ。

のが面倒だから)1 つのコマンドにしたい、ということが多い。たとえば、画面の背景色を赤色にするには

```
% xsetroot -solid red
```

というのを実行すればいいのだが、結構長くて覚えるのが面倒である。そこでただ

```
% red
```

といえば済むように、つまり red というコマンドを作ったりしたいわけである。

実は、UNIX ではコマンドをファイルに入れておいて実行させることが簡単にできる。つまり「red」というファイルに

```
#!/bin/sh
xsetroot -solid red
```

という 2 行を入れておき、そして「chmod ugo+x red」でこのファイルを実行可能にすれば、単に red というだけでこのファイルの中に入っているコマンドが実行される。これを「シェルスクリプト」と呼ぶ。そして、この新しいコマンドがどこででも使えるようにするには、そのファイルを ~/bin に入れておけばよい。また、自分で作った GIF ファイルを背景にするコマンドだったら

```
#!/bin/sh
cd /home/...    ← GIF ファイルのあるディレクトリを指定
xloadimage -onroot GIF ファイル名
```

という内容にすればよいわけだ。

演習 5 適当な名前で、画面の背景を決まった色または自分の好きな GIF イメージに設定するコマンドを 2 つ作り、そのコマンドがどこででも実行できることを確認せよ。

3 手続きの復習と関数/△

3.1 関数

さて、今日の Pascal の話題であるが、これまで「trunc(x)」だとか「sin(t)」だとかいうものを使ってきた。これらは、x や t のところに適当な値を渡すとそれに基づいた計算をして結果を返してくれるので、(数学でいう関数にならって)「関数」と呼ばれる。しかしその中身は前回やった「手続き」によく似ている。違うのは以下の点だけである。

- 「procedure」の代りに「function」と書く。
- 返す値の型を指定する。
- 返す値を「関数名 := 式」という形で指定する。

具体的には次の形になる。

```
  function 関数名 (引数:型; ...; 引数:型) : 型 ;  ←これが返す値の型
  begin 文 ; ... ; 文 end;
                ↑ 必要に応じて「関数名 := 式」で値を指定
```

たとえば、「2 つの整数を受け取り大きい方を返す」関数を定義してみる。

```
function max2(a,b:integer):integer;
begin
  if a > b then max2 := a else max2 := b
end;
```

これを適当な場所に (手続きなどと同じ場所に並べて) 入れておけば、それより後では式の中で「max2(x, y)」というのが自由に使えるようになる。

3.2 下請けの下請け…

ところで、上の例は「2 つの数の最大」だったが、「3 つの数の最大」だとどうだろう? もちろん、if の中で if を使えばできるが…次のはどうですか?

```
function max3(a,b,c:integer):integer;
begin
  max3 := max2(a, max2(b, c))
end;
```

つまり、「3 つのうち最大のものは、b と c の大きい方と a とを比べてその大きい方」というわけ。このように、一度作った関数は徹底して利用し倒そう。

3.3 塗りつぶした三角形

さて、前回やった図形 (円、楕円、長方形、直線) でけっこう色々な絵が作れるが、「塗りつぶした三角形」があるともっと嬉しい。というのは、

任意の多角形は三角形に分解して描けばできるから。なので、塗りつぶした三角形の描き方を教えておこう。

まず、点 (x_0, y_0) と (x_1, y_1) を通る直線の方程式は

$$y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0}$$

ですね? (高校の数学だぜ。) だから、この「=」を「>」に置き換えると、その不等式を満たす点の集まりは直線より上の「A」の領域、また「<」に置き換えると直線より下の「B」の領域を表す。ここまでいい?

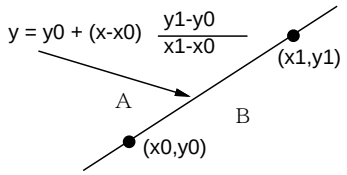

図 4: 直線の式

言い替えれば、次の式が正か負かによって点 (x, y) が直線のどちら側かが判定できる (0 ならば直線上にある):

$$(y - y_0)(x_1 - x_0) - (x - x_0)(y_1 - y_0)$$

ただしこの式は、さっきの式の右辺を左辺に移項して、それから垂直な線のときに x_0 と x_1 が等しくなって 0 で割ってしまうのを防ぐため、両辺に $(x_1 - x_0)$ を掛けたものである。この値を計算する Pascal の関数を作っておこう。

```
function ptv(x0,y0,x1,y1,x,y:real):real;
begin
    ptv := (y-y0)*(x1-x0) - (x-x0)*(y1-y0)
end;
```

さて、三角形では直線が 3 つあるから、これを組み合わせる。つまり、(x_0, y_0)、(x_1, y_1)、(x_2, y_2) から 2 つずつ選んで組み合わせて上の式にあてはめ、それの積を計算する。つまり次の計算をする。

```
function ptv3(x0,y0,x1,y1,x2,y2,x,y:real):real;
begin
  ptv3 := ptv(x0,y0,x1,y1,x,y)*ptv(x1,y1,x2,y2,x,y)*ptv(x2,y2,x0,y0,x,y)
end;
```

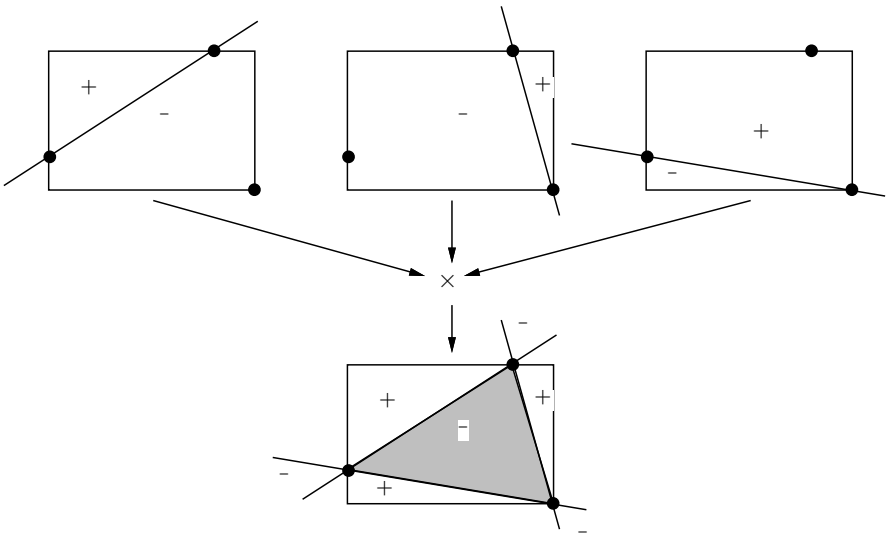

図 5: 三角形の領域

ptv3 の符号は、図 5 のように三角形の内側の符号がマイナスで、そのすぐ外側がプラスであるか、または (点の順序によってはその反対に) あるいは内側がプラス、そのすぐ外側がマイナスになるはず。で、内側の符号がどちらかは、三角形の重心 $(\frac{x_0+x_1+x_2}{3}, \frac{y_0+y_1+y_2}{3})$ について ptv を計算してその値 (v としよう) の符号を調べればわかる (それと同じになる)。なお、交差部の外側も中と同じ符号になるが、そこは調べないので関係ない。

で、どの範囲の (x, y) について調べればいいか? それは、図 5 の長方形の中について調べればよい。この長方形の左端の X 座標は $min3(x_0, x_1, x_2)$、右端の X 座標は $max3(x_0, x_1, x_2)$、上端の Y 座標は $max3(y_0, y_1, y_2)$、下端の Y 座標は $min3(y_0, y_1, y_2)$ ですね。というわけで、次の手続きができあがり。

```
procedure filltriangle(x0,y0,x1,y1,x2,y2,r,g,b:integer);
var x, y: integer;
    v: real;
begin
  v := ptv3(x0,y0,x1,y1,x2,y2,0.333333*(x0+x1+x2),0.333333*(y0+y1+y2));
  for x := min3(x0,x1,x2) to max3(x0,x1,x2) do
    for y := min3(y0,y1,y2) to max3(y0,y1,y2) do
      if ptv3(x0,y0,x1,y1,x2,y2,x,y) * v >= 0 then point(x, y, r, g, b)
end;
```

頭痛いですか? 理解したくない人はまあそのまま打ち込んで使ってくだ

さい。

3.4 下請け手続き…

さて、おまけとして、真っ赤な円と三角形で「顔」を描く手続きを作って見た。

```
procedure redface(x,y:integer);
begin
  fillcircle(x, y, 50, 255, 0, 0);
  filltriangle(x-40, y+15, x-20, y+15, x-30, y+25, 0, 0, 0);
  filltriangle(x+40, y+15, x+20, y+15, x+30, y+25, 0, 0, 0);
  filltriangle(x, y-25, x+15, y-15, x-15, y-15, 0, 0, 0)
end;
```

描く位置は引数として渡すようになっているので、メインプログラムから繰り返し呼び出すことでいくつも顔が描ける。たとえば次のメインプログラムだと:

```
begin
  clearcanvas;
  redface(80, 100);
  redface(190, 120);
  writecanvas;
end.
```

図6のような絵になる。なお、このプログラム全体は例によって`~kuno/pascal/sam11a.p`に入れておくので好きに使ってください。

A 本日の課題 11A

本日の課題は、演習1で作りかけたLaTeXの文書を完成させて出力を提出してもらいます。文書のタイトルは「レポート11A」とすること。その各sectionは次のようにする。

- 「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。
- 「UNIXのコマンド」の節: UNIXでコマンドがいろいろ作れることについてどう思ったか、実際にコマンドを作ってみてどうだったかを記す。

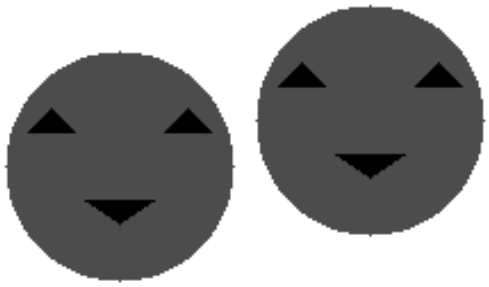

図 6: 「赤い顔」の絵

- 「感想と要望」の節: 感想と要望を記す。

なお、時間が余った人は idraw で簡単な図を作って挿入してみてください (下記参照)。

B　LaTeX をもう少し活用するには

B.1　図の入れ方

LaTeX に図を入れるには、図は PostScript でないといけない。idraw で描いたものはそのままで PostScript だが、ほかの形式なら xv などを使えば PostScript に変換できる。ファイル名は必ず「なんとか.ps」という名前でないといけない。

PostScript ファイルを用意したら、図を入れたい場所に次のようなコマンドを入れる。

```
\begin{figure}[htbp]
\begin{center}
\epsfile{file="なんとか.ps",scale=0.7}
\end{center}
\caption{図の説明文}
\end{figure}
```

あとはこれまで通り jlatex、xdvi、dvi2ps などを使って整形/プレビュー/出力するだけである。

B.2 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
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。)

B.3 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。

B.4 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。

B.5 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

B.6　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

B.7　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<>?=} \\
{\bf 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。

B.8　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
`$\backslash$`	\
`$\{$`	{
`$\}$`	}
`\$`	$
`\&`	&
`\#`	#
`\^{}`	^
`\_{}`	_
`\%`	%
`\~{}`	~

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。

B.9 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。