インターフェイスの街角– Perl で TEX

増井 俊之

TEX の悩み

UNIX のユーザーであれば、文書の整形に TEX を使っている人は多いと思います。かくいう私も、論文や手紙をはじめ、この原稿の執筆などに TEX を利用しています。しかし、TEX が本当に使いやすいと満足している人は意外に少ないのではないでしょうか。

HTML を解釈するブラウザが出現する以前は、UNIX 上での文書の整形には TEX を利用する以外にほとんど方法がありませんでした。とはいえ、手紙や論文のような定型的な文書であれば、蓄積された膨大なライブラリ（マクロファイル）を用いて比較的簡単に作れるため、けっこう重宝がられていました。TEX には、文書整形のための機能だけでなくマクロ言語も備わっているので、長い文字列を簡単な文字列で表現したり、あるいは数値計算や繰返し制御のようなプログラミングも可能です。ページ番号や章、図表などの番号も、これらの機能によって自動的に計算されます。

こういった機能はちょっと使うぶんにはたいへん便利なのですが、マクロ言語の動作が分かりにくいために、多少変わったことをしようとするとすぐにいき詰まってしまうことが多いようです。東京大学の萩谷昌己先生は、TEX に対して辛辣な批判を加えています[1]。その当否の判断はひとまず措くとして、TEX が初心者にとって扱いにくいシステムであることは多くの人が認めるところでしょう。私自身にしても、複雑な機能を十分に使いこなしているとはいえません。

1 http://nicosia.is.s.u-tokyo.ac.jp/pub/essay/hagiya/h/tensai

それでは、初心者にも手軽に利用でき、ある程度以上の機能をもつ文書整形システムには、どのような条件が必要なのでしょうか。

オーサリングとプログラミング

さきほど述べたように、TEX のもっとも重要な機能は文書の整形ですが、マクロ言語を用いて自由に拡張できる点も大きな特徴です。

このような〝基本的な機能をプログラミング言語によって拡張する〟という仕組みは、TEX 独自のものではありません。事実、多くの対話的システムやオーサリング・システムでも同様の方法が用いられています。

たとえば、GNU Emacs では Emacs Lisp という言語を用いてさまざまな機能を付加できます。Windows や Mac OS 用のエディタにしても、マクロ機能をもっているものは少なくありません。ほかにも、アニメーション作成システムとしてひろく使われている Director は Lingo という言語でプログラミングが可能であり、Windows 上の MIDI 音楽作成ソフト Cakewalk では CAL というプログラミング言語が使えます。

このように、用途を拡張するためにプログラミング機能を追加する手法はさまざまなシステムで用いられています。しかし、この方法に問題がないわけではありません。思いつくままに挙げてみただけでも、下記のような短所があります。

- アプリケーションごとに独自の言語を習得しなければならない

  たいていの拡張言語はアプリケーション独自のものなので、文書の整形や編集、音楽の作曲、アニメーションの

生成など、アプリケーションごとに異なる言語を憶える必要があります。

- 拡張言語の仕様はとかく巨大化しやすい

ある程度以上のことを実現しようとすれば、拡張言語といえども一般的なプログラミング言語と同様な機能を備えているべきでしょう。条件分岐や繰返しなどの制御文は当然として、算術演算などの機能も必要になります。場合によっては、変数やサブルーチン呼出しのような機能が求められるかもしれません。

このように、アプリケーションの拡張言語といえどもかなり高い機能が必要となり、けっきょくは汎用言語との違いがなくなってしまいます。

用途が限定された小さな拡張言語ならまだしも、アプリケーションごとに異なる膨大な機能をもつ汎用言語を憶えるのはほとんど不可能です。

Perl で TEX

アプリケーションごとに独自の言語を用意するのが問題であるならば、**汎用プログラミング言語に専用機能を追加する**という方法が考えられます。

その一例が、1999 年 4 月号で紹介した〝ストトンシステム〟[2]です。ストトンシステムでは、特殊な音楽記述言語を用いて楽曲を記述するのではなく、C や Perl のプログラムに〝ドレミ〟のようなテキストで表現した演奏データを埋め込んで演奏機能を実現しています。したがって、たとえば〝ドレミを 100 回鳴らす〟といった場合にも C や Perl の制御文が利用でき、独自の音楽記述言語を憶える必要がありません。

TEX にも、これと同様な方式が応用できそうです。つまり、TEX のマクロ機能を使う代わりに Perl などの汎用言語を用いて文章を書き、そこから TEX の機能を呼び出すわけです。そうすれば、TEX の難解なマクロ機能を使いこなせなくても複雑な処理が実現できます。この方式では、ページ番号の計算など、TEX の内部処理についての知識を必要とする情報は扱えませんが、マクロの置換や関数の定義といった部分には汎用言語の機能が利用できます。

以下では、いくつかの例を用いて Perl から TEX を呼び出す仕組みの利点をみていくことにします。

2 http://www.csl.sony.co.jp/person/masui/Sutoton/

図 1　報告書を作成するプログラム

```
#!/usr/local/bin/perl
# 報告書

require 'report.pl';    # スタイルファイルの呼出し

$loc = '東京ビッグサイト'; # マクロ定義
$name = 'PC Expo 2000';

&title("$name 報告");
&author('増井俊之');
&date('2000/10/20');
&place($loc)

&text(<<TEXTEND
${loc}で開催された${name}に参加した...
TEXTEND
);

&out;
```

簡単な例

まず、実行すると TEX ファイルが出力される Perl プログラムを作成します。これによって、TEX のマクロ機能を利用するのとほぼ同じ要領で文章を書くことができます。

たとえば、図 1 の Perl プログラムでは、簡単な報告書の TEX ファイルが作れます。これは Perl のプログラムですが、〝スタイルファイル〟の指定や〝マクロ定義〟のあとで本文を記述するという点では、一般的な TEX/LATEX の記法と大差ありません。

ここで呼び出している report.pl は、図 2 のような単純なもので十分です。

制御文の利用

Perl の if 文や while 文を使えば、条件分岐や繰返しを簡単に記述できます。条件分岐や繰返しは TEX のマクロで記述することも可能ですが、読みやすい記法とはいえませんし、変数などの使用に制限があり手軽には使えません。

Perl を利用すると、たとえば以下のプログラムで漢字コード表を簡単に生成できます。

```
print "\\begin{tabular}{|l|l|}\n\\hline\n";
for($h=0xa1;$h<=0xa1;$h++){
  for($l=0xa1;$l<=0xfe;$l++){
    printf("%02X%02X & %c%c \\\\\n",
```

図 2　report.pl

```
# report.pl

sub title  { $title  = $_[0]; }
sub author { $author = $_[0]; }
sub place  { $place  = $_[0]; }
sub date   { $date   = $_[0]; }
sub text   { $text  .= $_[0]; }

sub out {
  print <<EOF;
\\documentstyle{jarticle}
\\author{$author}
\\title{$title}
\\begin{document}
\\maketitle

\\begin{description}
\\item[日時] $date
\\item[場所] $place
\\item[報告] $text
\\end{description}

\\end{document}
EOF
}
1;
```

```
            $h,$l,$h,$l);
  }
}
print "\\hline\n\\end{tabular}\n";
```

これを実行して得られた TEX テキストにより、以下のような表が作成できます。

A1A1	
A1A2	、
A1A3	。
A1A4	,
.....	.....
A1FD	◎
A1FE	◇

計算機能の利用

TEX にも簡単な計算機能はありますが、使いやすいとはいえません。汎用的なプログラム言語にもとづいていれば、計算が必要になった場合も、普通の数式で計算できるので便利です。

たとえば、0 から 10 までの総和を計算する式を、

```
\[ \sum_{i=0}^{10} i \]
```

と TEX で記述すると、

$$\sum_{i=0}^{10} i$$

のような数式の出力が得られます。

しかし、この計算そのものを TEX で処理して解答を得るのは容易ではありません。そこで、Perl と併用して以下のようなプログラムを作成すれば、計算結果を含む数式を出力することができます。

```
$n = 10;

$sigma = 0;
for($i=0;$i<=$n;$i++){
  $sigma += $i;
}

print <<EOF;
\\[ \\sum_{i=0}^{$n} i = $sigma \\]
EOF
```

このプログラムを実行すると、以下の数式出力が得られます。

$$\sum_{i=0}^{10} i = 55$$

外部プログラムの利用

UNIX には、TEX 以外にも文書作成を支援するさまざまなツールがあります。しかし、通常の TEX では外部プログラムを呼び出すことはできないので、外部プログラムで生成されたテキストをあらかじめ TEX 形式のファイルなどに格納しておく必要があります。

ASCII 文字を用いて描いた図表を TEX 形式のテキストファイルに変換する plain2 というツールがあります[3]。plain2 は、図 3-a のようなプレーンテキストを TEX 形式のファイル（図 3-b）に変換してくれます。

plain2 を用いてこのような TEX 形式の表を作成するには、罫線で描いたテキストファイルを用意しなければなりません。当然のことながら、TEX の文書でこれを利用するためには、plain2 の出力結果をファイルとして保存しておく必要もあります。複数のファイルを扱う場合など、

3 日本電気の内田昭宏さんが開発したプログラムで、各地の anonymous FTP サーバーから入手できます。

図 3　plain2 によるテキストから TEX 形式への変換

(a) 入力するテキスト

```
+-----+--+
|りんご|10|
+-----+--+
|みかん|20|
+-----+--+
```

(b)plain2 の出力

```
~\\
\begin{center}
\begin{tabular}{|c|c|}
\hline
りんご & 10\\
\hline
みかん & 20\\
\hline
\end{tabular}\\
\end{center}
```

(c) 整形結果

りんご	10
みかん	20

手順が複雑になるときは Makefile を用意しなければならないこともあるでしょう。

しかし、これも Perl を利用すれば以下のようなプログラムとして実現できます。

```
sub plain2 {
  local($text) = @_;
  open(plain2,
       "| plain2 -tex -nopre > plain2.tex");
  print plain2 $text;
  close(plain2);

  `cat plain2.tex`;
}

$t = &plain2(<<EOF);
+-----+--+
|りんご|10|
+-----+--+
|みかん|20|
+-----+--+
EOF

print $t;
```

図版の自動生成

グラデーションで塗り潰した矩形を文書の区切りなどに使うときには、あらかじめ EPS (Encapsulated PostScript) 形式などのファイルを作成しておかなければなりません。

この場合にも、図 4 の Perl プログラムで以下のような EPS の図形を生成し、同時に TEX で参照可能な形式のファイルを出力することもできます。

図 4　グラデーション・ファイル生成プログラム

```
open(gradation,"> gradation.eps");
print gradation <<EOF;
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 0 0 100 10
0 1 1000 {
    /i exch def
    i 1000 div setgray
    /x1 i 10 div def
    /y1 0 def
    /x2 x1 0.1 add def
    /y2 10 def
    newpath
    x1 y1 moveto x2 y1 lineto
    x2 y2 lineto x1 y2 lineto
    closepath
    fill
} for
EOF
close(gradation);

print <<'EOF';
\epsfile{file=gradation.eps, ⇒
width=\columnwidth}
EOF
```

（誌面の都合上、⇒ で折り返しています）

ビットマップ・データの利用

TEX にビットマップ・データを取り込む場合、通常はデータを EPS ファイルに変換したものを使用します。一般に写真などの画像データはサイズが大きく、テキストとして扱うのは非現実的かもしれません。しかし、小さなビットマップ・データであれば、テキストとして記述しておくと扱いが楽になるでしょう。

また、TEX に標準で用意されていないフォントを使いたいときは、あらかじめ TEX 用のフォントをシステムにインストールしておく必要があります。しかし、せいぜい数回しか利用しないフォントをインストールするのは時間と労力の無駄になりかねません。これも、Perl プログラムで生成するようにしておけば、いくらか簡単になります。

たとえば のような特殊な記号を 1 回だけ使うといった場合には、フォントをインストールするのではなく、ビットマップ・データを利用するほうが手間がかかりません。図 5 の Perl プログラムを実行すると、カレント・ディレ

図 5　カーソルデータの定義と参照

```
open(cursoreps,"| pnmtops > cursor.eps");
print cursoreps <<EOF;
P1
10 16
1100000000
1010000000
1001000000
1000100000
1000010000
1000001000
1000000100
1000000010
1000000001
1000001110
1001001000
1010100100
1100100100
0000010010
0000010010
0000001100
EOF
$cursor =
  '\epsfile{file=cursor.eps,height=10pt}';

print <<EOF;
定義したフォントは
$cursor
のようにして参照できます。
EOF
```

クトリに cursor.eps というビットマップ画像が作成され、

定義したフォントは
\epsfile{file=cursor.eps,height=10pt}
のようにして参照できます。

という TEX 形式の出力が得られます。

動的なデータの使用

文書を整形するときにプログラムを起動したり計算したりすることができれば、動的に変化するデータも扱えます。

たとえば、下記のプログラムを実行すると日付が出力されます。

```
require 'ctime.pl';
$date = &ctime(time);
print "今日の日付は${date}です。\n";
```

また、以下の Perl プログラムは、自分自身のサイズを計算して出力します。

1/2AD スペース
(ノンブル段階で小口寄りに)

```
$file = "mytext.tex";
@stat = stat($file);
$size = $stat[7];
print "このファイルのサイズは${size}バイトです。\n";
```

コマンド引数の使用

■ プリプロセッサのように働く、ということでしょうか?

TEX にはコマンド行オプションがありませんが、Perl プログラムであれば、コマンド行で文書整形オプションを指定することもできるでしょう。

各種のチェック

さきほど例に挙げた図 2 の report.pl は、out 関数の部分で最終的な TEX 形式のテキストを出力するようになっています。これに手を加えて、ファイルとして出力する直前にスペルチェックをおこなったり、文字列を置換したりすることも可能です。

TEX 以外のテキストの生成

ここまでに挙げた例はすべて TEX 形式のファイル出力を前提としていましたが、出力する部分を変更すればほかの形式のファイルにも簡単に対応できます。まったく同じテキストファイルから TEX と HTML の文書を生成することもそれほど難しくありません。

TEX 形式のファイルを HTML 形式に変換する場合には、latex2html というプログラムがよく使われているようです。しかし、最初から TEX にも HTML にも変換可能な形式で文書を作成しておけば、そのような外部プログラムを使わずにすみます。

おわりに

以上に紹介した手法は、〝複数の言語の得意な部分を組み合わせて目的を達成する〟一例です。今回の例は、汎用言語としての Perl と、文書整形言語としての TEX を組み合わせたことになります。

これまでにも、複数の言語を組み合わせてプログラムなどを効率的に作成するさまざまな手法が考案されてきました。NEXTSTEP では C と PostScript が組み合わせて使われていましたし、C と SQL のような組合せも珍しくありません。ユーザー・インターフェイスの状態遷移、並列実行、画面表示などを簡単に記述するための**ユーザー・インターフェイス記述言語**を使い、C などのアプリケーション言語と併用する方法も提唱されています。異なるモデルにもとづく複数の言語を組み合わせると、いろいろな問題が出てきます。しかし、それぞれの守備範囲で十分実用的なものであれば、この種の組合せが有効に機能することも多いと思います。

汎用言語によるプログラムと文書を同一視するのは、なんとなく抵抗があるかもしれません。また、今回紹介した Perl プログラムは、Perl、TEX、PostScript などで記述された部分が混在しているため、TEX 形式のファイルより読みやすいとはいえません。

これらの手法を現実に使う場合には、Makefile を利用するのが一般的でしょう。たとえば、plain2 で生成した TEX ファイルを別の TEX ファイルで使いたいのであれば、下記のような Makefile を用意しておくといいかもしれません。

```
document.tex: graph.tex
graph.tex: graph.plain2
  plain2 -tex -nopre graph.plain2 > graph.tex
```

ただし、上の例では文書ファイル本体 (document.tex) 以外に挿入するファイル (graph.tex) や、実行を制御するための Makefile も必要になり、結果としてファイルやディレクトリ構成が複雑になってしまうという問題もあります。

今回紹介した手法は、現実にある程度有効な場合もあると思いますが、プログラムを文書の作成に利用する 1 つの例として考えていただければさいわいです。

プログラム本体も解説文書もマニュアルも 1 つのプログラムから生成できれば、1999 年 12 月号で紹介した WEB システムを超える世界が開けてくるかもしれません。

（ますい・としゆき　ソニー CSL）