LaTeX による文書作成（その 3）

プログラミング演習 1#13
平成 25 年 7 月 12 日

1　コマンドや環境を作ろう

LaTeX で文章を書いている時，同じフレーズを何度も書く必要に迫られることがあります．たとえば，emacs についての解説文を書いている時は，\texttt{emacs}という文字列を何度も LaTeX ソース中に入れたくなります．論文中で操作 α と操作 β を定義した上で，$\xrightarrow{\alpha}$ や $\xrightarrow{\beta}$ といった記号をふんだんに使って

$$\begin{aligned}(\lambda xy.xy)(\lambda x.x)x &\xrightarrow{\alpha} (\lambda ab.ab)(\lambda c.c)x\\ &\xrightarrow{\beta} (\lambda c.c)x\\ &\xrightarrow{\beta} x\end{aligned}$$

というような式を書きたくなることがあります．$\xrightarrow{\alpha}$ や $\xrightarrow{\beta}$ といった記号は\stackrel{\alpha}\rightarrow や\stackrel{\beta}\rightarrow と書くことで作ることができますが，これらの記号が現れるたびにこんな長い表現を書くのは面倒ですし，頑張って書いたとしても LaTeX ソースファイルがごちゃごちゃして見難くなります．

このように頻繁に使う表現がある場合には，\newcommand を用いて自分で新しくコマンドを作ると便利です．上記の例だと

```
\newcommand{\emacs}{\texttt{emacs}}
\newcommand{\aarrow}{\stackrel{\alpha}\rightarrow}
\newcommand{\barrow}{\stackrel{\beta}\rightarrow}
```

というのを文書の最初の方に書いておけばよろしい[1]．こうしておけば，\emacs やら\aarrow やら\barrow やら書くだけで，emacs やら $\xrightarrow{\alpha}$ やら $\xrightarrow{\beta}$ やらが作られます．

コマンドを自分で作って使用するようにすることで，楽になるばかりでなく，ソースが見やすくなります．また，たとえば後から「やっぱ emacs じゃなくて EMACS にしよう」と思い直した時にも，\newcommand の行を

```
\newcommand{\emacs}{\textsc{emacs}}
```

に直すだけで，すべての\emacs と書かれている箇所が一度に\textsc{emacs}という解釈に置き換わってくれます．さらに，1 箇所しか書き換わらないので，バージョン管理する上でも衝突（コンフリクト）が起きにくくなります．

コマンドは引数（パラメータ）を取ることができます．たとえば,「俺様専用の括弧 $(\!|\cdot|\!)$ を作ったぜ！」[2]という場合を考えてみます．数式モード中で (\!|x|\!) と書けば，$(\!|x|\!)$ と表示されます.（なお，\!は間隔を詰めるコマンド.）でも，毎回いちいちこんな面倒な文字列を書かなくても

```
\newcommand{\banana}[1]{(\!|#1|\!)}
```

[1] \emacs, \aarrow, \barrow というのは適当に決めた名前．ただしバックスラッシュ（環境によっては¥マーク）で始まる必要がある．

[2] ちなみにこの括弧は全然「俺様専用」ではありません．http://en.wikipedia.org/wiki/Catamorphism 参照.

と文書の最初の方に書いておけば，それ以降の場所（の数式モード中）に`\banana{x}`と書くことで $(|x|)$ が得られますし，もちろん`\banana{\sin\theta}`と書けば $(|\sin\theta|)$ が得られます．便利ですね．

`\newcommand`の一般的な書式は以下のようになります：

```
\newcommand{コマンド名}[引数の数]{コマンドの表す内容}
```

複数の引数がある場合は，それらを {} で括って与えていくと，「コマンドの表わす内容」の中の`#1`, `#2`, ... が順に置き換わっていきます．たとえば，

```
\newcommand{\dbanana}[2]{(\!|#1,#2|\!)}
```

としておくと，`\dbanana{\phi}{\psi}`と書くことで $(|\phi,\psi|)$ が得られます．

コマンド名としてすでにある名前を指定するとエラーになります．このような場合には，別の名前を指定しましょう．もし，すでにあるコマンドを変更したい場合は，`\renewcommand`を用います．書式は`\newcommand`と同じです．

本日のやってみよう#1
先々週から編集している文書で，`\Info`というコマンド（別の名前でもいいけど）を定義して，これを用いて $x\log_2 x$ という形式になっている部分をすべて書き換えてみよう．
ちなみに，第 1 引数の内容が複数箇所で参照される場合でも，その各箇所に`#1`を書いておけば，全て第 1 引数で置き換わります．（もちろんこれは第 2 引数以降についても言えることですが．）

1.1 コマンド中でのカウンタ

`\section`などの LaTeX コマンドも，`\newcommand`により作られています．このため，`\renewcommand`による再定義が可能です．また，上記の「やってみよう」も`\newcommand`により実現されています．

`\section`にはセクション番号があり，「やってみよう」には「やってみよう番号 (?)」があり，これらはコマンドを呼び出すごとに増えていきます．LaTeX において，このような増大していく数は「カウンタ」という仕組みで実現されています．

カウンタを使うには，まず

```
\newcounter{カウンタ名}
```

でカウンタを定義します．（この時，カウンタの値は 0 になります．）その上で，カウンタを増やしたい箇所で

```
\addtocounter{カウンタ名}{増やしたい値}
```

を実行します．また，そのカウンタをアラビア数字で表示したければ，表示する場所に

```
\arabic{カウンタ名}
```

と書いておきます．他にも，ローマ数字とか漢数字とかいろいろあったと思います．（漢数字はなかったかも？）

というわけで，「やってみよう」のための`\TryIt`コマンドは，下記のような感じで実現されています．

```
\newcounter{TryIt}
\newcommand{\TryIt}[1]{
  \vspace{5mm}
  \doublebox{
```

```
        \begin{minipage}{0.9\textwidth}
            \addtocounter{TryIt}{1}
            \textbf{本日のやってみよう\#\arabic{TryIt}}\\
            #1
        \end{minipage}
    }
}
```

注釈 1 \doublebox コマンド（*2* 重の四角で囲むコマンド）を使うには\usepackage{fancybox}を指定しておく必要があります.

注釈 2 カウンタ名の TryIt とコマンド名の\TryIt として，バックスラッシュ以外同じ名前を用いていますが，これは利便性からそうしているのであって，必ずしもそうする必要があるわけではありません．また，カウンタとコマンドは本来独立のものなので，類似した複数のコマンドで同一のカウンタを使うことによって通し番号にすることができます．たとえば，来週の資料において「本日のやってみよう*#2*」の次が「本日の（ヒマなら）やってみよう*#3*」になっていますが，これは同一のカウンタを使うことによるものです.

本日のやってみよう#2
最初に書くと「問題 1」と表示し, 次に書くと「問題 2」と表示し，...となるような\exercise コマンドを作ってみよう（カギカッコは不要）．これを使って，先々週から編集している例の文章に

問題 1　1 + 3 は？

ってのを入れてみよう.

1.2　環境の定義

上記の\TryIt のようなそこそこ長いコマンドを定義して使う場合，コマンドの始まりはコマンド名が来るので分かりやすいですが，コマンドの終わりには}一文字しかないため，特に入れ子になった時にどの始まりがどの終わりに対応しているのか分かりにくいことがあります.

このような場合には，コマンドではなく環境を定義すると，終端が\end{環境名}で終わるため，多少わかりやすくなります.（代わりに，たくさんタイプする必要が出てきますが，まあその辺はテキストエディタが多少は手伝ってくれるかもしれません.）環境は，以下の形式で定義します.

\newenvironment{環境名}[引数の数]{「\begin{環境名}」で実行する内容}{「\end{環境名}」で実行する内容」}

2　条件付きコンパイル

皆さんが 4 年生になると，卒業論文（1 段組・長め）とその概要（2 段組・4 ページ）を書かなければなりません．また，もし家庭教師のアルバイトでもやっていようものなら，通常の問題用紙と模範解答の書かれた問題用紙を作成する必要にかられることもしばしばあるでしょう．私の博士課程の時の同級生は，ゼミの配布資料をこっそり「通常版」と「先生に渡す版」でビミョーに変えたことがあります.

このように，内容は似ているものの細部や体裁を変えて複数の文書を作成する必要に迫られることは，大学生活の様々な場面でしばしば起こります．ここでは，いかにエレガントに複数の版を生成するかを説明します.

2.1 \newif による分岐

TEX には，\if で始まるコマンドがいくつか定義されています．(そのものずばり\if というコマンドもあります．) これらは，

```
\if 条件
...
\fi
```

または

```
\if 条件
...
\else
...
\fi
```

という構造をしています．前者の (\else がない) 場合は，\if 条件と\fi の間の部分は「条件」が真の時のみコンパイルされます．(「条件」が偽の時はなかったことになります．) 後者の場合は，\if 条件と\else の間の部分は「条件」が真の時のみコンパイルされ，\else と\fi の間の部分は「条件」が偽の時のみコンパイルされます．

\if で始まるコマンドは，\newif コマンドで作ることができます．具体的には，

```
\newif\if 条件
```

とすると，\if 条件というコマンドと，条件を真に設定するための\条件 true というコマンドと，条件を偽に設定するための\条件 false というコマンドが作られます．そこで，たとえば

```
\newif\ifAnswer
%\Answertrue
%\Answerfalse

\newcounter{exercise}
\newcommand{\exercise}{\addtocounter{exercise}{1}問題\arabic{exercise}\quad}

(中略)

\exercise $1+3$は？
\ifAnswer

... $4$!
\fi
```

という風にすれば，\Answertrue の行の%を外せば答え付き版が作成され，\Answerfalse の行の%を外せば答えなし版が作成されることになります．またたとえば

```
\newif\ifJapanese
%\Japanesetrue
```

```
%\Japanesefalse

\ifJapanese
\documentclass{jarticle}
\else
\documentclass{article}
\fi
\begin{document}
\ifJapanese
以下の式を満たす$(a,b,c)$の組をピタゴラスの三つ組といいます.
\else
Triples $(a,b,c)$ satisfying the following equation
are called {\em Pytagorian Triples}:
\fi
\[
a^2 + b^2 = c^2
\]
\ifJapanese
この名称は有名なピタゴラスの定理から来ています.
\else
This name is taken from the famous Pytagorian Theorem.
\fi
\end{document}
```

という風にすれば，`\Japanesetrue` の行の`%`を外せば日本語版が作成され，`\Japanesefalse` の行の`%`を外せば英語版が作成されることになります.

2.2 `\input` によるファイルの挿入

以上でもそこそこ便利ではあるのですが，このやり方だと，`cp` や `mv` でファイル操作しない限り，同じファイル名に上書きしてコンパイルすることになります．これには

1. コンパイルする前に`.tex` ファイルをいじる必要があるので，`.tex` ファイルの最終更新日が更新され，正確な日にちを反映しなくなる.
2. コンパイルする前に`.tex` ファイルをいじる必要があるので，面倒.
3. 参照などをゼロからコンパイルすることになるため，余計な時間がかかる.
4. 異なるバージョンを同時に持てないので不便．異なる版が欲しい場合，いちいち再コンパイルする必要がある.
5. `\date` コマンドを使用せずに`\maketitle` を実行するソースの場合，タイトルの日付欄が再コンパイルした日付になってしまう.

といった問題があります．もちろん，上書きされないように別のファイル名に書き換えておけば複数の版を同時に持てますが，それはそれで面倒です.

このような問題を解決するには,

- 最初の 3 行以外の部分を含むファイル（仮に body.tex とする）
- \newif\if 条件 \条件 true を実行した後，この body.tex を読み込むファイル（仮に true.tex とする）
- \newif\if 条件 \条件 false を実行した後，この body.tex を読み込むファイル（仮に false.tex とする）

の 3 つを作成すればよいです．true.tex をコンパイルすれば true.dvi などのファイルが作成されますし，false.tex をコンパイルすれば false.dvi などのファイルが作成されます．

ファイルの読み込みには，\input コマンドを用います．前出の例だと，

body.tex:

```
\ifJapanese
\documentclass{jarticle}
\else
\documentclass{article}
\fi
\begin{document}
\ifJapanese
以下の式を満たす$(a,b,c)$の組をピタゴラスの三つ組といいます.
\else
Triples $(a,b,c)$ satisfying the following equation
are called {\em Pytagorian Triples}:
\fi
\[
a^2 + b^2 = c^2
\]
\ifJapanese
この名称は有名なピタゴラスの定理から来ています.
\else
This name is taken from the famous Pytagorian Theorem.
\fi
\end{document}
```

true.tex:

```
\newif\ifJapanese
\Japanesetrue
\input body.tex
```

false.tex:

```
\newif\ifJapanese
\Japanesefalse
\input body.tex
```

という感じになります.

本日のやってみよう#3
先々週から編集している例の文章を更にいじくって，この資料の置いてあるサイトにある `long.pdf` と `short.pdf` が作成できるようにしてみよう！ちなみに，この 2 つの版の違いは，

- 1 段組か 2 段組かの違い
- 例 1 及び図表が入っているかどうかの違い

だけです．なお，`long.tex`, `short.tex` の 2 つの短いファイルを作ることになりますが，これらのファイルも忘れずに `darcs add` しておくとよいでしょう．あと，`short.pdf` が 1 ページに収まらなくても気にしないでくれたまえ.