第 1 章

TEX の基礎

ここでは LATEX でのマクロ・クラスの解読の前に理解すべきことを紹介します．LATEX は TEX を基盤としたマクロの集まりですから，まずは TEX の基礎を習得すべきでしょう．この章では TEX の基礎的な知識を学んでみようと思います．ランニングと筋力トレーニングのような基礎勉強になりますが，気楽に読んでください．

特殊な用語に関してですが，本冊子は独自の訳し方を使っています[*1]．

1.1 LATEX の基盤

LATEX はどのように構築されているのでしょうか．それらは全て `source2e.tex` に書かれていますが，これを理解するには何かが足りません．その足りない知識とは TEX についてです．LATEX は TEX を基盤としていますから TEX に関する知識，いわゆる**プリミティブ**と呼ばれる命令について理解する必要があります．プリミティブとは TEX のプログラム自体に直接組み込まれているコマンドのことで，数百個あります．これら全てを理解する必要はありませんが，LATEX のマクロを理解する上で必要なものが沢山あります．この章では LATEX の基盤である TEX のプリミティブとそれに関わる機構について少し紹介しようと思います．

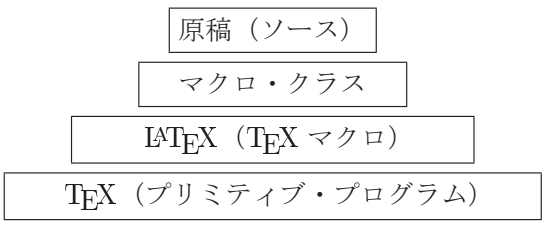


図 1.1　TEX と LATEX の関係

[*1] *TEXbook* の訳が変だということではなく，より一般的で分かりやすい表記を目指しただけです

1.2　TEX の処理状態

モードについて.

1.2.1　数式中でのスタイル

D, T, S, SS, 圧縮スタイル？

1.3　TEX の罫線

\hrule と\vrule の特性

水平線を文中に引こうと\verb|\hrule|を
使おうとしたら{\hrule}全然うまく逝きません.
逆 に\verb|\vrule|の 方 が{\vrule width10pt}良
い
です. なぜですか？

水平線を文中に引こうと\hrule を使おうとしたら
全然うまく逝きません. 逆に\vrule の方が良いです. なぜですか？

このことから LATEX の\fbox 命令は

```
\vbox{ \hrule
       \hbox{\vrule\relax{これ}\vrule}%
       \hrule}%
```

と同じようなことを行っていると考えられるでしょう.

{\fboxsep=0pt\fbox{これ}}は多分{\TeX}の
\verb|\hrule|, \verb|\vrule|, \verb|\vbox|,
\verb|\hbox|を次のようにしていると考えられます.

さっきのは
\vbox{\hrule\hbox{\vrule\relax こ　　れ
\vrule}\hrule}

これ は多分 TEX の\hrule, \vrule, \vbox, \hbox を
次のようにしていると考えられます. さっきのは これ

しかし少し考えてみれば LATEX では\fbox を作成するときに\fboxsep と\fboxrule によって罫線の太さと文字列との間隔を調整できます. もう少し工夫した定義が必要なのはお分かりだと思います.

第 2 章

LaTeX のマクロ用コマンド

2.1 変数の型

それほど多くはない．ブール型，カウンタ型，スキップ，...

数値ではなくトークンとして次の命令が定義されている．ltplain.dtx にて定義される．\@ne=1 \tw@=2 \thr@@=3 \sixt@@n=16 \@cclv=255 \@cclvi=256 \@m=1000 \@M=10000 \@MM=20000

\newcount \newdimen \newskip \newmuskip \newbox \newhelp \newtoks

2.2 クラスファイルの解剖

マクロ・クラス作成に関わる命令の数は膨大で，それらを一つずつ解説していては読者の皆さんも疲れるだろうから，ここは実際に奥村晴彦氏の jsarticle.cls を解剖してガリッと理解しましょう．順に見ていきましょう．最初は\NeedsTeXFormat と\ProvidesClass 命令です．

```
\NeedsTeXFormat{pLaTeX2e}
\ProvidesClass{jsarticle}[2004/02/25 okumura]
```

ここでは必要とされる TeX の種類（pLaTeX 2_ε）と提供するクラスファイルの名前（jsarticle）を指定します．

```
\newif\if@restonecol
\newif\if@titlepage
\newif\if@enablejfam \@enablejfamtrue
```

\newif 命令は新規にブール型の変数を定義します．クラスファイル（⟨*file*⟩.cls）やマクロ（⟨*file*⟩.sty）の中ではアット ‘@’ は通常の英字と同じように扱われますから，\makeatletter や\makeatother といった命令は使われていません．

\newif\if⟨ブール変数⟩

次は**クラスオプション**の宣言をします．

```
\DeclareOption{a4paper}{%
  \setlength\paperheight {297mm}%
```

```
    \setlength\paperwidth  {210mm}}
  \DeclareOption{a5paper}{%
    \setlength\paperheight {210mm}%
    \setlength\paperwidth  {148mm}}
```

\DeclareOption 命令でそのクラスファイルが提供するオプションを宣言できます.

\DeclareOption{⟨オプション⟩}{⟨内容⟩}

{⟨オプション⟩}を宣言するときの{⟨内容⟩}は何かの定義でも構いませんし，パラメータの設定でも構いません.

フォントサイズの設定をします.

```
\DeclareOption{9pt}{\renewcommand{\@ptsize}{-1}}
\DeclareOption{10pt}{\renewcommand{\@ptsize}{0}}
\DeclareOption{11pt}{\renewcommand{\@ptsize}{1}}
\DeclareOption{12pt}{\renewcommand{\@ptsize}{2}}
```

\@ptsize というフォントサイズの基準を決める命令を再定義しています.

用紙の縦と横の長さを入れ替えるための処理をしています.


```
\DeclareOption{landscape}{%
 \setlength\@tempdima   {\paperheight}%
 \setlength\paperheight {\paperwidth}%
 \setlength\paperwidth  {\@tempdima}}
```

2.3　寸法単位

何かの長さを示すときには，ある基準となる**単位**が必要になります．単位には絶対的な単位と相対的な単位の 2 種類があります．絶対的な単位にはメートル ‘m’ のように速度の変化しない（と言われている）光速が進むことのできる距離を基準にしているものもあります．

TeX（pTeX）には表 2.1 に示すような寸法が用意されています．例えば`\parindent`

表 2.1　寸法単位

単位	説明		実際の長さ
bp	ビッグポイント	72 bp=1 in	
cc	シセロ	1 cc=12 dd	
cm	センチメートル	2.54 cm=1 in	
dd	ディドーポイント	1157 dd=1238 pt	
in	インチ	1 in=72.27 pt	
mm	ミリメートル	10 mm=1 cm	
pc	パイカ	1 pc=12 pt	
pt	ポイント		
sp	スケールポイント	65536 sp=1 pt	
em	現在の ‘M’ の幅		
ex	現在の ‘x’ の高さ		
zh	現在の和文の高さ		
zw	現在の和文の幅		

の値を指定するには次のようにします．

```
\parindent=1zw
\parindent-10pt
\parindent 3 em
\parindent 1.5 zw
```

さて，次のような学位論文用の表紙を作りたいとしましょう．通常のクラスファイルが提供している表代用の命令には

```
\title \author \date \maketitle
```

などが用意されています．これらと同じような処理をすれば表紙をカスタマイズできそうです．では

```
\def\date#1{\gdef\@date{#1}}
\gdef\@date{\today}
```

hogehogehoge

\llap と\rlap の理解

```
\def\llap#1{\hbox to 0pt{\hss#1}}
\def\rlap#1{\hbox to 0pt{#1\hss}}
```

```
Y\llap =\par% 成功
\rlap = Y\par% ゴバーーーーー (o ∀ o) ーーーーー
ク!!!!
```

¥
=
Y

2.4　TEX のディレクトリ構造

TEX は大変便利なプログラムでしたから，多くの人が TEX に関わるプログラムやマクロなどを作りました．その結果もの凄い数のファイルがうまれ，どこに何を置けばよいのか分からない状態になりました．そこで，[TeX Users Group]TEX Users Group がどのファイルをどこに置くべきかを定めました．これが *A Directory Structure for TEXFiles* と呼ばれる文書に記されています．

TEX に関連するファイルを格納すべき一番上のディレクトリ（フォルダ）を`$TEXMF` と呼びます．この`$TEXMF` を探すには次のように入力します．

```
▌kpsewhich -expand-var='$TEXMF'
```

kpsewhich というプログラムを使って`$TEXMF` の変数を展開すると

```
/usr/local/share/texmf
```

などと表示されます．通常はこのディレクトリの下に関連するファイルが収納されています．

おおむね`$TEXMF` 以下のディレクトリ構成は次のようになります．スラッシュ '/' はディレクトリ（フォルダ）の切れ目を意味します．

`bibtex/`　欧文用の BIBTEX に関わるファイルが格納されています．`bibtex/bst/`以下には参考文献スタイルが，`bibtex/bib/`以下には参考文献データベースがあると思います．

`context/`

`cweb/`

`doc/`

`dvipdfm/`

`dvips/`

`fontname/`

`fonts/`

`jbibtex/`

`ls-R`　Unix 系 OS で標準的に使われているコマンド ls に`-R` オプションを付けた出力を保存したようなファイル．Cygwin などの stat（スタット）[*1] が相当遅いマシンではこの `ls-R` が必要になるでしょう．`ls-R` ファイルを作るには専用のプログラム mktexlsr を使います．端末から引数なしで実行すると自動的に`$TEXMF` を見つけ出し，適切に処理をしてくれます．この `ls-R` は`$TEMXF` 以下からファイルを検索するために使

[*1] 元々はラテン語なのですが，プログラムが検索されて実行されるまでの時間のことだと思っていただけると良いかと思います．

われます．もしも$TEXMF 以下のディレクトリ構成が変更された場合は mktexlsr を実行しなおさなければなりません．最近の Windows は十分な処理速度がありますから，この ls-R は不要と思われます．削除するときは端末から deltexlsr を実行するだけです．

```
makeindex/
metafont/
metapost/
mft/
omega/
pdftex/
ptex/
tex/
tex4ht/
web2c/
```

2.5　ハイフネーション

行を分割するにはうんぬんカンヌ Σ (・ん・ ;).

```
\hyphenation{⟨単語, ...⟩}
\fussy
\sloppy
```

```
{\fussy hoge hoge hoge hoge
 hogehogehogehogehogehoge hoge.}\par
{\sloppy hoge hoge hoge hoge hoge
 hogehogehogehogehoge hoge.}\par
hoge hogehogehogehogehoge hoge.
```

hoge hoge hoge hoge hogehogehogehogehogehoge hoge.
hoge hoge hoge hoge hoge hogehogehogehogehoge hoge.
hoge hogehogehogehogehoge hoge.

2.6　(o д o) ノ■ ビックリバコドゾー

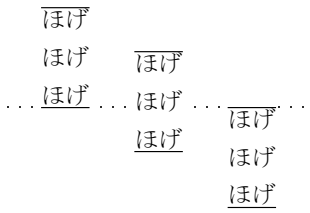

```
\hbox{⟨要素⟩}（水平モード）
\vbox{⟨要素⟩}（内部垂直モード）
```

\hbox は水平モードなので ⟨要素⟩ を水平に並べようとします．\vbox は垂直モードですから，⟨要素⟩ を垂直に並べようとします．

```
\hbox{\hbox{ほげ}\hbox{ほげ}\hbox{ほげ}}
\vbox{\hbox{ほげ}\hbox{ほげ}\hbox{ほげ}}
\vbox{\leavevmode\hbox{ほげ}\hbox{ほげ}%
 \hbox{ほげ}}
```

ほげほげほげ
ほげ
ほげ
ほげ
ほげほげほげ

```
\newcommand{\tete}{\hbox{ほげ}
 \hbox{ほげ}\hbox{ほげ}}
\ldots\vbox{\hrule\tete\hrule}
\ldots$\vcenter{\hrule\tete\hrule}$
\ldots\vtop{\hrule\tete\hrule}\ldots
```



2.7　マクロの定義

\def⟨綴り⟩{⟨定義内容⟩}

```
\def\are{sore}
\def\kore{dore}
\def\irekae#1#2{#2\space#1}
\are\ \kore\ \irekae \are \kore
```

sore dore dore sore

```
\def\irekae#1#2\par{#2、#1。\par}
\irekae {アレは}{良いものだぁ}\par
```

良いものだぁ、アレは。

2.8　四則演算

TEX での四則演算は結構面倒.

\advance⟨綴り⟩⟨値⟩
\multiply⟨綴り⟩⟨値⟩
\divide⟨綴り⟩⟨値⟩

calc を使いましょう.

2.9　マクロ作成

▷ **例題 2.1** 次の実行結果から\repeat と\fi の役割について考えてください.

```
\@tempcnta=\z@
\loop\ifnum\@tempcnta<10
  \the\@tempcnta\space
  \advance\@tempcnta\@ne
\repeat
\def\check@tempcnta{%
  \ifnum\@tempcnta<10
    \the\@tempcnta\space
    \advance\@tempcnta\@ne
    \check@tempcnta
  \fi}%
\@tempcnta=\z@
\check@tempcnta
```

▷ **例題 2.2** TEX では\TeX 命令は TEX と定義されているが，LATEX では TEX と定義されているのは何故でしょうか. は space factor である事を考慮してください.

▷ **例題 2.3** 次の実行結果から\undefined や\empty，\@empty などの意味をもう少し理解してください．　未定義何か苦 undefined

定義済み何か苦 macro:->hoge

定義済み空苦 macro:->

このことから LaTeX の\@ifundefined は何をしていると考えられるでしょうか．あぁ，話がまとまらない．

```
\expandafter\ifx\csname#1\endcsname\relax
  \expandafter\@firstoftwo
\else
\expandafter \@secondoftwo\fi
```

▷ **例題 2.4**

```
\def\check@meaning#1{\bgroup\ttfamily
  \expandafter\meaning\csname#1\endcsname\relax\egroup\par}
\check@meaning{LaTeX}
```

として\LaTeX がどのように定義されているのかを調べてください．結果は

```
\protect\LaTeX
```

となることから，定義時には\DeclareRobustCommand が使われていることを理解してください．結果から\DeclareRobustCommand は何をしていると考えられますか．次のように入力してその答えを確認してください．

```
\check@meaning{DeclareRobustCommand}
\check@meaning{declare@robustcommand}
```

さらに\@star@or@long は何をするマクロかを考えてください．

\showoutput[⟨整数⟩]
\tracingall
\tracingcommands[⟨整数⟩]
\tracingstats[⟨整数⟩]
\tracingpages[⟨整数⟩]
\tracinglostchars[⟨整数⟩]
\tracingmacros[⟨整数⟩]
\tracingparagraphs[⟨整数⟩]
\tracingrestores[⟨整数⟩]

▷ **例題 2.5** \arabic と\@arabic の違いを理解してください．

```
\def\check@meaning#1{\bgroup\ttfamily
  \expandafter\meaning\csname#1\endcsname\relax\egroup\par}
\check@meaning{arabic}
\check@meaning{@arabic}
```

上記の結果から\arabic と\@arabic の定義が

```
\def\arabic#1{\expandafter\@arabic\csname c@#1\endcsname}
\def\@arabic#1{\number#1}
```

であると分かります．そうすると\arabic{page}はできても{\arabic{\c@page}はできなし，\@arabic\c@page はできても\@arabic{page}はできないことは分かりますよね．

▷ **例題 2.6**
```
\check@meaning{fnsymbol}
\check@meaning{@fnsymbol}
```
から\@fnsymbol では最大いくつまで表示できますか．

▷ **例題 2.7** 次の結果から\@alph はどのような限界があると考えられますか．

```
\@tempcnta= 3 \@alph\@tempcnta\space
\@tempcnta=13 \@alph\@tempcnta\space
\@tempcnta=25 \@alph\@tempcnta\space
\@tempcnta=26 \@alph\@tempcnta\space
\@tempcnta=27 \@alph\@tempcnta\space
```


▷ **例題 2.8** 次の出力結果から\@roman はどのようなキャパシティを持っていると考えられますか．

```
\@tempcnta= 3 \@roman\@tempcnta\space
\@tempcnta=13 \@roman\@tempcnta\space
\@tempcnta=23 \@roman\@tempcnta\space
\@tempcnta=50 \@roman\@tempcnta\space
\@tempcnta=103 \@roman\@tempcnta\space
\@tempcnta=255 \@roman\@tempcnta\space
```

▷ **例題 2.9** \@roman の例を参考に\@alph を次のような動作がするように拡張してください．

a, b, c, . . . , z, aa, ab, ac, . . . , az, ba, bb, bc, . . .

▷ **例題 2.10** \usefont について理解してください．\usefont は 4 つの引数を取ります．その 4 つはそれぞれ何でしょうか．

1. エンコーディング
2. ファミリ
3. シリーズ
4. シェイプ

となります．

▷ **例題 2.11** \usefont の役割を考えると\normalfont は何をする命令でしょうか．NFSS の選択機構を考えて自分で同じような命令を定義してみてください．解はおおむね次のようになります．

```
\DeclareRobustCommand\normalfont{\usefont
  \encodingdefault
  \familydefault
  \sariesdefault
  \shapedefault
  \relax}
\let\reset@font\normalfont
```

\normalfont で最後に\relax が必要な理由を考えてください．\usefont で選択されたフォントは例えば次のように展開されます．

```
\usefont\encodingdefault\familydefault\sariesdefault\shapedefault
```

ならば\OT1/cmr/m/n/10 など．

▷ **例題 2.12** \pagenumbering と等価な\pagenokazu を作成してください．\@roman\c@page とすれば良いのでおおむね\c@page を 1 に初期化し，\thepage を\csname @#1\endcsname\c@page と定義すれば良いことになります．

```
\def\pagenokazu#1{%
  \c@page=\@ne
  \def\thepage{\csname @#1\endcsname\c@page}}
```

しかし ltpageno.dtx では\global が使われています．その理由を考えてください．

```
\def\pagenonumbering#1{%
  \global\c@page\@ne
  \gdef\thepage{\csname @#1\endcsname\c@page}}
```

▷ **例題 2.13** \begin と\end は何をする命令でしょうか．

```
\check@meaning{begin}
\check@meaning{end}
```

2.10　preload files

LaTeX 処理を実行した原稿のプリアンブルに

```
\listfiles
```

という\listfiles 命令を記述すれば，自分が処理している原稿に何のファイルが使用されているのかを端末と ⟨*filename*⟩.log に書き出します．例えば

```
\documentclass{jbook}
\listfiles
\begin{document} hoge \end{document}
```

のようなファイル ⟨*filename*⟩.tex が存在し，platex などで LaTeX 処理をしたならば

```
This is pTeX  Version 3.141592-p3.1.2 (sjis)(Web2C 7.5.2)
 *File List*
  pldefs.ltx    2000/07/13 v1.2 pLaTeX Kernel
    jy1mc.fd    1997/01/24 v1.3 KANJI font defines
    jy1gt.fd    1997/01/24 v1.3 KANJI font defines
 kinsoku.tex
 plpatch.ltx
    jbook.cls    2001/10/04 v1.3 Standard pLaTeX class
    jbk10.clo    2001/10/04 v1.3 Standard pLaTeX file
 ***********
Output written on filename.dvi (1 page, 216 bytes).
Transcript written on filename.log.
```

のような表示が出るでしょう．ここで少し疑問に思っていただきたいことは，「原稿には jbook を使うことしか宣言していないのに何か別のファイルも一緒に読み込まれている．」ということです．このように LaTeX プログラムを実行した段階ですでに読み込まれているファイルを *preload file* と呼びます．

2.11　簡単なマクロの作成

自分でマクロの作成ができたほうがなんだか LaTeX を使いこなせると思い込めるので，ここでは簡単なマクロの作成方法を紹介します．

2.11.1　マクロ作成で役に立つコマンド

- \ifnum{⟨数値 1⟩}{⟨関係演算子⟩}{⟨数値 2⟩} 関係演算子は<, >, =のいずれか．
- \ifdim{⟨寸法 1⟩}{⟨関係演算子⟩}{⟨寸法 2⟩} 寸法の比較．
- \ifodd{⟨数値⟩}
- \ifx{⟨トークン 1⟩}{⟨トークン 2⟩} トークンが一致しているかどうか．
- \ifcase{⟨数値⟩} \or \or \else \fi
- \number{⟨数値⟩}
- \string{⟨トークン⟩}
- \jobname
- \csname{⟨トークン⟩}\endcsname
- \the へんすうの現在地を表すトークンのならびに展開される．\the\textwidth, \the\parindent,

数式中のコロンやセミコロン数式の中に使用されたカンマやセミコロンの後ろには細スペースを置く．これは\colon を用いて出力するコロン記号とまったく同じことを行う．そうでないとコロンは$x:=y$や$a:b::c:d$というように，関係演算子としてみなされる．これは $f\colon A \to B$
$L(a,b;c\colon x,y;z)$

2.11.2　条件判断

基本的な条件判断のためのコマンドは LaTeX tools の calc パッケージで定義されている．

\equal{⟨str1⟩}{⟨str2⟩}
\ifthenelse{⟨条件⟩}{⟨then⟩}{⟨else⟩}
\lengthtest{⟨len op len⟩}
\isodd{⟨数値⟩}
\boolean{⟨名前⟩}ブールレジスタの値を評価．
\and, \or, \not, \(, \) を括弧として使うと，複合条件半田もできる．
\whiledo{⟨判断⟩}{⟨処理⟩}

2.12　米国数学会の提供するマクロ

2.12.1　amsmath パッケージの中身

$\mathcal{A}\mathcal{M}\mathcal{S}$-LaTeX には数式記述に関するマクロパッケージ amsmath や，LaTeX では用意されていない AMSFonts などが含められています．amsmath パッケージの中では

amsbsy　数式を太字にする\boldsymbol を使うための命令が定義されている．
amstext　数式中で文章を出力する\text 命令が定義されている．
asmcd　ダイヤグラムを描くための CD 環境が定義されている．
amsopn　新規に演算子を定義するための\DeclareMathOperator 命令が定義されている．
amsxtra　その他のコマンドが定義されている．

の 5 つのパッケージが自動的に読み込まれます．そのため amsmath を読み込んでおけばこれらのパッケージを読み込む必要はありません．

amsmath パッケージのパッケージオプションとしては以下のオプションが使用できます．

centertags
tbtags
sumlimits
nosumlimits
intlimits
nointlimits
namelimits
nonamelimits
leqno
ceqno
fleqn

2.13　この章は未完成です

高度な数式の組版に関してはまだまだ調査が足りないのでこの章はまだ完成しそうにありません．そのため高度な数式の組版を行いたい方は以下の文献を参照してください．

[1] 小田忠雄. 数学の常識・非常識——由緒正しい TeX 入力法.
http://www.math.tohoku.ac.jp/oda_tex/oda_tex.pdf
ウェブ上で無料で入手できるマニュアルです．とても参考になりますので，ご一読ください．

[2] Michael Downes. *Short Math Guide for LaTeX*. http://www.ams.org/tex/short-math-guide.html
英語ですが amsmath などの解説を含むマニュアルです.

他にもいろいろと文献がありますので付録 ??などの書籍を参照してください.

2.13.1　自作の箇条書き型の環境

箇条書きとはいくつかの段落を改行や余白などを含めた一連の項目のことです．LaTeX ではすでにこの箇条書き型の環境が登場しています．主な環境として

- ラベル付きの itemize 環境.
- 番号付きの enumerate 環境.
- 説明ラベル付きの description 環境.
- 項目を中央揃えさせる center 環境.

などがあります．箇条書き型の環境では項目の始めにラベルを付けて字下げを挿入します．ラベルや字下げは必ずしも必要ではないので，中央揃えさせるだけの center 環境にも使えます．既存の箇条書き型環境で満足できないときは list 環境や，少し制限のある trivlist 環境のパラメータの変更を行います.

2.13.2　list 環境

list 環境は汎用性の高い箇条書き型環境です.

```
\begin{list}{⟨標準のラベル⟩}{⟨設定⟩}
⟨項目⟩
\end{list}
```

2.13.3　trivlist 環境

2.14　箇条書き環境の自作

```
\list{⟨ラベル⟩}{⟨命令⟩} ⟨項目⟩ \endlist
\item
```

```
\begin{trivlist} ⟨項目⟩ \end{endlist}
```

図 2.1 `list` 環境で設定できる値

図 2.2　リスト環境の形式

\leftmargin (0)
\labelwidth (0)
\itemindent (0)
\linewidth (\hsize)

\makelabel
\usecounter{⟨hoge⟩} \boxlabels

垂直空白 (スキップ)

\topsep (0)
\partopsep (0)
\itemsep (0)
\parsep (\parskip)

水平空白 (寸法)

\leftmargin ()
\rightmargin (0pt)
\listparindent (0pt)
\itemindent (0pt) \labelwidth (0) \labelsep (0)

\leftmargini \leftmarginii \leftmarginiii \leftmarginiv \leftmarginv
\leftmarginvi

\itemize と\enumerate カウンタ enumi，enumii，enumiii，enumiv，

2.15 索引

索引の選定基準方法は例えば次のようにします．

1. 人名，事実，概念，その文書特有の考え方などを把握する．
2. どのような語を索引に追加するのかを決める．JIS の業界別用語などを参考にする．
3. 読者が本当にその語を引くかどうかを検討する．
4. 索引に追加した語と同じ概念も追加する．例えば著者にとって「インストール」という概念で事項を検索しようと思ってもある読者は「導入」という語で引くかもしれない．

LaTeX で索引作成を実現する方法は次のようになります．

1. プリアンブルに\{usepackage}{makeidx}を書く．
2. プリアンブルに\makeindex 命令を書く．
3. 原稿の中で\index 命令を使い索引を追加する．
4. 索引を出力したい場所に\printindex 命令を書く．
5. タイプセットをして ⟨*file*⟩.idx に並べ替えられていない索引語を出力する．
6. MakeIndex プログラムを実行して，⟨*file*⟩.idx の索引語を並べ替えて ⟨*file*⟩.ind に出力する．
7. もう一度タイプセットを行い ⟨*file*⟩.ind を\printindex により読み込む．

索引の形態には次のようなものがあります．

- 語の階層的な概念．
- ページ範囲．
- 相互参照．
- 読みと出力の違い．

2.15.1 mendex のコマンドラインオプション

-l 索引語の並び替えを文字順に行う．標準は単語順．
-q 処理状況をあまり表示しない．
-c スペースやタブが連続しても 1 つとしてみなす．
-g 日本語の頭文字を『あかさ…わ』にします．標準は『あいうえ…わをん』です．

-s ⟨sty⟩　スタイルファイルを ⟨sty⟩ から読み込みます．何も指定しなければ標準のスタイルを使います．

-d ⟨dic⟩　⟨dic⟩ を辞書ファイルとして読み込みます．

-o ⟨ind⟩　成形後のファイルの出力先をです．

-t ⟨log⟩　処理結果を出力するファイル名を指定します．標準であれば ⟨*file*⟩.ilg に出力します．

2.15.2　複数の索引

index パッケージか multind パッケージを使うと簡単です．multind は少々古いのですが扱いが簡単ですから紹介します．プリアンブルで multind を\usepackage します．最初からある\printindex を少し変更します．

```
\renewcommand{\printindex}[2]{\chapter*{#2}\input{#1.ind}}
```

theindex 環境は multicol パッケージなどを使って定義すると良いでしょう．次に\makeindex で作りたいだけの索引ファイルを指定します．


```
\makeindex{nam} % 人名索引
\makeindex{pro} % プログラム索引
\makeindex{ind} % 概念索引
\makeindex{var} % 変数索引
```

あとは文章中でどの索引ファイルに加えるのかを指定して\index 命令を使います．

```
ところで\index{ind}{ほげ}ほげとはなんだろうか．
私はかねてよりほげに関する論文を書こうと思い，
ほげの考案者\index{nam}{ななしのごんべえ@名無しの権兵衛}
名無しの権兵衛氏に連絡を取ろうと試みた．しかし彼は
すでに不治の病にかかり，プログラム\index{pro}{ほげ}ほげは
消去されていた．ほげを知るうえでプログラムほげの消失は
我々にとって大きな存在となった．
```

索引を出力したい場所で\printindex を書きます．

```
\printindex{nam}{人名索引}
\printindex{pro}{プログラム索引}
\printindex{ind}{概念索引}
\printindex{var}{変数索引}
```

全ての記述が終わったらタイプセットします．次に端末などから複数の索引ファイル分 mendex などで処理します．

```
mendex -s style.ist nam
mendex -s style.ist pro
```

```
mendex -s style.ist ind
mendex -s style.ist var
```

するとディレクトリには pro.ind，ind.ind，varind，namind が作成されますのでもう 1 度タイプセットします．これで複数の索引を作成できました．

もう 1 つの方法に index パッケージを使う方法も紹介します．まずプリアンブルで \usepackage で index を読みこみます．次に\newindex 命令で作成したい索引を定義します．

\newindex{⟨ラベル⟩}{⟨処理前⟩}{⟨処理後⟩}{⟨見出し⟩}

⟨ラベル⟩ には複数の索引を区別するためのラベルを書きます．一般的な概念索引の場合はプリアンブルに

```
\newindex{id}{idx}{ind}{索引}
```

のようにします．同じように他の索引も次のように定義します．

```
\newindex{nam}{ndx}{nnd}{人名索引} % 人名索引
\newindex{pro}{pdx}{pnd}{プログラム索引} % プログラム索引
\newindex{var}{vdx}{vnd}{変数索引} % 変数索引
```


以上の作業ができたならば\index 命令に任意引数にラベルを指定して索引を追加します．

```
ところで\index[id]{ほげ}ほげとはなんだろうか．
私はかねてよりほげに関する論文を書こうと思い，
ほげの考案者\index[nam]{ななしのごんべえ@名無しの権兵衛}
名無しの権兵衛氏に連絡を取ろうと試みた．しかし彼は
すでに不治の病にかかり，プログラム\index[pro]{ほげ}ほげは
消去されていた．ほげを知るうえでプログラムほげの消失は
我々にとって大きな存在となった．
```

最後に索引を出力したい場所に\printindex 命令を書きます．

```
\printindex[nam]
\printindex[pro]
\printindex[var]
\printindex[id]
```

以上のような記述をしたファイル file.tex あるならばタイプセットします．これもここの索引ファイルを mendex などで処理します．

```
mendex -s style.ist -o file.ind file.idx
mendex -s style.ist -o file.nnd file.ndx
mendex -s style.ist -o file.pnd file.pdx
mendex -s style.ist -o file.vnd file.vdx
```

索引の数だけ mendex を実行したならば再びタイプセットします．

2.15.3　用語集

用語集も基本的に索引と同じように mendex で作成すればよいのですが，まず始めに注意すべき点は以下の命令が索引の場合とは違います．

\index → \glossary
\indexentry → \glossaryentry
theindex 環境 → theglossary 環境

大抵のクラスファイルにはあらかじめ theindex 環境が定義されているのですが theglossary 環境は定義されていません．とりあえず

```
\newenvironment{theglossary}{\begin{theindex}}{\end{theindex}}
```

のようにしておきます．

2.16　本作り

紙に印刷され本屋さんで販売されるような本を LaTeX で作るならば相当慎重になってください．そもそも LaTeX を使うと出版社側の規定と合わない部分が数多く発生すると思われます．本を出す大雑把な流れとして

原稿作成　本に書かれる文章を書きあげます．
校正　その文章に間違いや誤植などがないかを調べます．
再校　校正の結果，修正すべき箇所を書き直します．場合によっては複数回行います．
製版　紙に印刷するために版を作成します．最近はオンデマンド印刷といって版を作成しない場合もあります．小部数の場合は版を作成せずにオンデマンド印刷したほうが安上がりです．
印刷　版を元に紙に印刷します．いろいろな印刷方式があります．オフセット印刷などが主流です．
装丁　本の顔である表紙や製本方法を決めて，それを行います．
製本　原稿が印刷された紙を人が読める本にします．
販売　お店に並べます．内部資料の場合は直接本が読者に渡されるでしょう．
読者が読む　本として出来上がったならば読者がその本を読みます．

となりますが，執筆者が LaTeX で原稿を入稿する場合は出版社側でその原稿を編集できるかどうかが鍵となります．もし出版社側でレイアウト，体裁，そのほか必要と思われる作業ができないとなると，全て執筆者が LaTeX で行うことになります．クラスファイルの設計が十分にできる人でなければこれは難しいのではないかと思います．出版社によっては LaTeX のクラスファイルが提供されています．LaTeX での文書作成に慣れていなけ

ればべた書きのテキストファイルで渡したほうが安全だと思います．特に縦組みの本を作るならば自分の LaTeX での技術に頼らずに出版社さんに丸投げしたほうがまともな本ができるかもしれません．

本を出すためには多くの方々の協力が必要です．出版社さんは校正，製版，印刷，製本，販売という一連の作業を各企業と連携を取って本を作ってくれるのです．出版社さんはまず本を出すために**企画**を行います．原稿が出来上がったらそれを組版します．執筆者が LaTeX で原稿を書いている場合はこの作業は必要ないでしょう．ただし，あまりにも滅茶苦茶な原稿だと打ち直しされるかも知れません．原稿が出来上がったら**製版所**で製版をします．オンデマンド印刷ではこの作業は省略されます．製版が終了したら**印刷所**で印刷を行います．印刷が終了したら**製本所**で製本作業をします．印刷所が製本を掛け持つ場合もあります．このときに製版所と印刷所が別とか，印刷所と製本所が別というケースも存在します．LaTeX を PDF に変換した PDF を出版社さんに入稿し，上記のような過程があると，思わぬ場面で問題が起きる可能性もあります．出版社さんと綿密な企画・打ち合わせを行ってください．中小の出版社は自社で製版システム，印刷機，製本システム，販売システムを確保していない場合があります．このため複数の業者さんが常備しているシステムの確認を行うのも良いでしょう．

2.17　〇〇流の定義

複数の引数を持つ命令の定義

```
%\usepackage{color}
\newcommand\hoge[2]{{\color{red}#1 is red.} {\color{blue}#2 is blue.}}
```

赤 is red. 青 is blue.

TeX 流の定義

```
\def\hoge#1#2{{\color{red}#1 is red.} {\color{blue}#2 is blue.}}
```

赤 is red. 青 is blue.

より再帰的な環境の定義


```
%\usepackage{color}
\def\hoge{\bgroup\color{red} \@hoge}
\def\@hoge#1{#1 is red. \egroup\bgroup\color{blue}\@@hoge}
\def\@@hoge#1{#2 is blue.\egroup}
```

赤 is red. 青 is blue.

2.18　段落の余白の制御

```
\bgroup\rightskip=3zw \leftskip=\rightskip
段落が終了した時点で、値が反映される。
段落の終了を明示的に示さないとだめ。\par\egroup
\newenvironment{myquote}[2]%
  {\ifvmode\par\fi\bgroup\leftskip#1\relax\rightskip#2}%
  {\par\egroup}
\begin{myquote}
段落が終了した時点で、値が反映される。
段落の終了を明示的に示さないとだめ。
\end{myquote}
```

✂ それはちょっとやめたほうが良いんじゃないか？だってねぇ、これから逝くんでしょう？やっぱりやめたほうがいいよなぁ。あぁぁ。それはちょっとやめたほうが良いんじゃないか？だってねぇ、これから逝くんでしょう？やっぱりやめたほうがいいよなぁ。あぁぁ。それはちょっとやめたほうが良いんじゃないか？だってねぇ、これから逝くんでしょう？やっぱりやめたほうがいいよなぁ。あぁぁ。

2.19　自動語尾生成命令

```
\def\nyo#1{\item[で〇こ] #1 にょ}
\def\nyu#1{\item[ぷ〇こ] #1 にゅ}
\def\piyo#1{\item[ぴ〇こ]#1 ぴよ}
\def\gema#1{\item[げ　〇]#1 げま}
```

で〇こ　「あぁ、ひまだにょ」
ぷ〇こ　「そうかにゅ」
で〇こ　「なんか、面白いことはおこらないかにょ」
ぷ〇こ　「昨日は暴れん坊が来たにゅ」
で〇こ　「そうだったにょ」
げ　〇　「もう、こりごりげま」
ぷ〇こ　「だれかきたにゅ」
げ　〇　「だれげま」
ぴ〇こ　「ぴよー、ぴよーぴよ」
で〇こ　「いきなり、だれにょ」
ぷ〇こ　「だれにゅ」
げ　〇　「だれげま」
ぴ〇こ　「ぴよぴよぴよ、悪参上ですぴよ」

2.20　ブール型変数の簡易な使用方法

つぎのような定義をしておくと、TeX のブール値を扱いやすくできます。

```
\newcommand*\newbool[1]{%
  \expandafter\newif\csname if#1\endcsname\relax}
\newcommand{\TorF}[3]{\csname if#1\endcsname\relax #2\else#3\fi}
\newcommand{\ifTrue}[2]{\csname if#1\endcsname\relax#2\fi}
\newcommand{\ifFalse}[2]{\csname if#1\endcsname\else#2\fi}
\newcommand*{\True}[2]{\csname #1true\endcsname\relax}
\newcommand*{\False}[2]{\csname #1false\endcsname\relax}
%
\newbool{hoge}
\TorF{hoge}{真です、}{偽です。}
\True{hoge} \ifTrue{hoge}{真ですね。}
\False{hoge} \TorF{真だ。}{偽だ。}
```

さてさて、結果はどうなるか。

偽です。 真ですね。 偽だ。

しかし、このままでは未定義のブール値も構わず処理します。

```
 \newbool{geho}
\ifFalse{gege}{あれれ}{およよ}
```

そこで、定義済みかどうかを判定する処理を付け加え、もし未定義ならば処理を中断するようにしましょう。

2.21　ある命令が定義済みかどうかの判定

色々方法があるのですが、まずはすでに LaTeX で定義されている命令を見てみましょう。

```
\def\@ifundefined#1{\expandafter\ifx\csname #1\endcsname\relax
  \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\long\def\@firstoftwo#1#2{#1}
\long\def\@secondoftwo#1#2{#2}
```


まぁ、ちょっとした命令を作ってみますか。

```
\long\def\CheckDefine#1{%
  \expandafter\ifx\csname #1\endcsname\relax
  未定義 \else 定義済み\fi}
\CheckDefine{hoge} \CheckDefine{document}
```

結果はいかほど。 定義済み 定義済み

ここで\ifx の一つ目の引数は、\CheckDefine で与えられた一つ目の引数となります。しかし、二つ目の引数はなぜか\relax になっています。\relax は一体どのような定義になっているのでしょうか。これは調べる必要がありそうです。

2.22　モードのあれこれ

それぞれのモードを図解する、そのうち...

垂直モード
内部垂直モード
水平モード
限定水平モード
数式モード
ディスプレイ数式モード

2.22.1　モードの遷移

法則を以下にまとめる。

1. 垂直モードのとき、単なる文字列を見つけると水平モードに移行する。
2. 垂直モードのとき、特殊な命令によって水平モードに移行する（たとえば\indent や\leavevmode など）。
3. 垂直モードのとき、\hbox を見つけると限定水平モードに移行する。
4. ...

これを状態遷移図として、そのうち...

2.23　段落の制御あれこれ

2.23.1　段落の左右余白

\rightskip と\leftskip を使うと、ごにょごにょできる。ただし、**段落が終了した時点で設定が反映される**ので、段落の終了を明示的に示す必要もあります。


```
{\ifvmode\leavevmode\fi
 \leftskip=3zw \rightskip=3zw
 あいうえお、かきくけこ\par
}
```

波括弧は\leftskip や\rightskip などの大域変数の設定が外側の部分にも影響しないように、有効範囲（スコープ）を決めています。

2.23.2　字下げ量

インデントを操作する命令として\hangindent と\hangafter の二つがあります。

\hangindent (字下げの幅)
\hangafter (どの行から字下げするか決める)

2.23.3　もっとへんなの

\parshape $= n\, i_1\, l_1\, i_2\, l_2 \cdots i_n\, l_n$

2.24　日本の活版印刷の歴史

2.24.1　和文他書体への道

朗文堂でいくつか書籍を出しておられる片塩二朗氏の『活字に憑かれた男たち』 [?] は日本の活版印刷について詳しく調査・研究されている良書です。

```
@book{JK1999,
  author = {{\ruby{片塩}{かたしお}} {\ruby{二朗}{じろう}}},
  yomi = {Jiro Katashio},
  title = {活字に憑かれた男たち},
  ISBN = 4947613483,
  year = {1999},
  publisher = {朗文堂},
}
```


悲しい戦争の雰囲気が滲む変體活字廢棄運動についてや、**個人印刷所運動**、出版における言論の自由に関することなど、実に重要な資料が提供されています。日本の活版印刷の話のとどまらず、工芸家であるウィリアムス・モリス (William Morris 氏) の製本魂にも触れられています。書体のバランスの良さを決定する**オプティカルスケーリング**（視覚調整）や**リニアスケーリング**（単純幾何学変形）についての歴史は興味深いものです。

本題に入りますと、そこにはなぜ日本では「明朝体」と「ゴシック体」の 2 書体しかないかという問題が書かれています。これが先の**変体活字廃棄運動**のせいであることは、文献を参照していただければ良いので、とにかく日本では戦後からこの 2 書体しか使われていませんでした。しかし、DTP が広く普及するとそれらの制約もほとんどなくなり、また DTP 屋さんの方からも『2 書体ではデザインできない』という要望もあり、最近では様々な書体が使われるようになってきています。「様々な書体」といっても、実は戦前日本で使われていた書体でもあります。戦前には細明朝、太明朝、丸ゴシック、角ゴシックなどのユニークなものが存在していました。ですから、これらの書体を DTP の世界に持ってくることはなんら不思議はありません。これらの書体がどのように使われるべきなのかを、少し考えてみましょう。

まず、当時どのようにそれらの書体が使われていたのかを研究しましょう。現在、どのように使われているのかでも参考になるでしょう。いずれにしても書体は一貫性を持たせて使用することが前提となります。

ごにょごにょ...

2.25 書体

TEX は組版処理を行っているときに、使われてる書体の解像度やグリフ（字形）を一切気にかけていません*2。このような処理方法にすると、解像度などに依存しない、いわゆるデバイス非依存の仮想的なページを技術することが可能になります。これがいつも目にしている DVI 形式のファイルとなるわけです。さて、では TEX が書体のどの情報を使いページを組んでいるのでしょうか。それは文字配置（font metrics）という情報が記載された TFM（TEX font metric）ファイルと呼ばれるものです。和文書体の場合は JFM（Japanese Font metric）となります。この TFM/JFM ファイルには以下の情報がおおむね記載されています。

- その書体に含まれる各文字の幅（width）。
- その書体に含まれる各文字の高さ（height）。
- その書体に含まれる各文字の深さ（depth）。
- その書体の中での文字間のカーニング値（kerning）。
- 合字情報（）。
- イタリック補正値（）。

和文の場合は、これに若干色がつきますが、今は考慮する必要はないでしょう。

2.25.1 DVI ドライバの仕事

さて、そうするお DVI ファイルには使用されている書体の実際のグリフ情報等は反映されておらず、その書体の各文字の配置に関する情報しかないことになります。そうすると実際のグリフが記載された情報が必要になります。これは GF（generic font）ファイル、PXL（pixel font）ファイル、PK（packed font）ファイルなどに記載されています。DVI ドライバは都合に合わせてこれらの内からいずれかの形式のファイルを参照します。通常は圧縮された PK ファイルが参照されます。これらのファイルは Donald Knuth 氏が作成したフォント作成用プログラム METAFONT によってその書体がデザインされていることになります。

最近では解像度に依存しない True Type フォントも使われてきていますので、以上三つのファイルが参照されることも少なくなっています。

2.25.2 歴史背景

昔はフォントを選択するのも結構大変だった。今は楽になりました。

1989 年に Frank Mittelback 氏と Rainer Schöpf 氏らによって $\mathcal{AMS}$-LaTeX 用の

*2 最近の PDFTEX などはこの限りではないかもしれません。

NFSS1（New Font Selection Scheme ver. 1）がリリースされました。そののち１９９２年に LaTeX 2_ε 用に LaTeX3 Project Team のメンバーが NFSS2 （New Font Selection Scheme ver. 2）を発表しました。現在私たちがメインに使用しているのはこの NFSS2 となります。

2.25.3　フォント属性

エンコード　うーん、
ファミリー　ほげほげ。
シリーズ　ほげほげ。さらにウェイト（weight）とウィドゥス（width）に分類される。
シェイプ　ほげほげ。
サイズ　ほげほげ。

`\symbol` コマンドで全てにアクセスできる（欧文なら）。
Computer Modern: CM フォントのファミリー。

ローマン　cmr
サンセリフ　cmss
タイプライタ　cmtt

CM 数式フォントのファミリー。

math italic　cmm
数学記号　cmsy
拡張数学記号　cmex

ウェイト

ultra light	extra light	light	semi light	medium	semi bold	bold	extra bold	ultra bold
ul	el	l	sl	m	sb	b	eb	ub

ウィドゥス

uc, ec, c, sc, m, sx, x, ex, ux

シェイプ

n, it, sl, sc, ul

サイズ

pt, Q
実際には字送り（ベースラインスキップ）の幅も指定しなければならない。

\encodedefault (OT1)
\familydefault (\rmdefault)
\seriesdefault (m)
\shapedefault (n)

ファミリでは

\cmr cmr
\cmss cmss
\cmtt cmtt

Adobe の PS フォントではタイムズ、ヘルベチカ、クーリエの書体がそれぞれ ptm, pnv, pcr だが、次のように定義するとどうなるか。

```
\renewcommand\rmdefault{ptm}
\renewcommand\sfdefault{pnv}
\renewcommand\ttdefault{pcr}
```


実は times パッケージと同じ。

```
\renewcommand\familydefault{\sfdefault}
```

とするとどうなるか。

\fontencoding
\fontfamily
\fontseries
\fontshape
\fontsize

これらがさらに

\f@encoding
\f@family
\f@series
\f@shape
\f@size
\f@baseilneskip

実際には\familyencoding や\fontfamily を変更しても書体が選択されない。その雰囲気を\selectfont の定義から考えてください。

```
\DeclareRobustCommand\selectfont{%
...
```

さて、それでは一度書体を標準の設定に戻す\normalfont コマンドはどのような定義であるべきでしょうか。

```
\DeclareRobustCommand\normalfont{%
  \usefont\encodingdefault\familydefault\seriesdefault\shapedefault\relax}
\let\reset@font\normalfont
```

さて、最後の\relax が必要な理由と、頑丈に定義している理由も考えてください。展開された状態で、その次に普通の文字列がきたら？

2.26　問題集の作成

さて、theorem パッケージを使って、「例題型環境」や「問題型環境」を作成してみましょう。

例えば次のような仕様のマクロを作成しようとしましょう。

> 問題と解答を同じ場所に執筆し、何らかの切り替えで解答側の文章を別のファイル kaito.tex に書き出せるようにし、巻末で解答を表示する。

ここで使用例をまず示します。

```
\begin{Toi}
\begin{mondai}
問題を作成することは簡単かどうか、１０文字以内で答えなさい。
\end{mondai}
\begin{kaito}
非常に簡単ではない。
\end{kaito}
\end{Toi}
```

▸ **問題 2.14** 問題を作成することは簡単かどうか、10 文字以内で答えなさい。

ここで、別ファイルに解答を書き出すには TeX のファイル出力機能を使います。各問題における解答を一つの別ファイルに書き出し、原稿ファイルの末尾、いわゆる巻末にすべての解答を読み込みます。それをどのように実装するか、丁寧に解説します。

まず、「問題」は章カウンタ（chapter）と連動して問題番号を表示してほしいものです。これには theorem パッケージが流用できそうですから、今回はこのパッケージを使うことにします。今回は「問題」部分に特別な処理をしなくても良いと思いますが、今後のことも考えて環境だけは定義しておきます（このようにすると簡単に将来の拡張に対して対応できます）。

```
%\theorembodyfont{\normalfont}% プリアンブルで
%\newtheorem{myToi}{問}[chapter]% プリアンブルで
\newenvironment{Toi}{\begin{myToi}}{\end{myToi}%
  \label{toi:\c@myToi}}
\newenvironment{mondai}{}{}
```

さて、次は肝心の kaito 環境を定義します。TEX でファイル出力を使用するためには、まず\newwrite 命令で宣言します。

\newwrite{⟨ファイル名用の制御綴り⟩}

\newwrite には直接ファイル名をしません。あくまで制御綴りでその存在を確保するだけにします。

```
\newwrite\MyKaito
\immediate\openout\MyKaito\jobname.kai
\immediate\write\MyKaito{解答だよだけど、改行とかは出力されないよ。}
\immediate\write\MyKaito{解答だよだけど、改行とかは出力されないよ。}
\immediate\closeout\MyKaito\newpage
\input{\jobname.kai}
```


上記が簡易な例となります。実際にタイプセットして\jobname.kai の中身を確認してください。このままでは入力原稿の改行などが反映されませんので、改行も解答ファイル\jobname.kai に出力するようにします。

theorem パッケージ等の力を借りずにきちんと自分でカウンタを使って環境を定義しましょう。

```
\documentclass{jarticle}
\makeatletter
\def\NewProblem{%%%%
  \newwrite\kaitofile\relax
  \immediate\openout\kaitofile\jobname.kai%
  \ifx\chapter\undefined\newcounter{monmon}\else
     \newcounter{monmon}[chapter]\def\themonmon{\thechapter.\c@monmon}\fi}%
\newenvironment{mondai}{%
  \refstepcounter{monmon}%
  \label{mondai:\themonmon}
  \ifvmode\par\fi {\gtfamily 問~\themonmon}\quad}{\par}
\newcommand\kaito[1]{\immediate\write\kaitofile{%
  \string\par{\string\gtfamily\space 解答\space\string\ref{mondai:\themonmon}}%
  \string\quad\relax#1}}
```

```
\newcommand\KAITO{\immediate\closeout\kaitofile\newpage
  \ifx\chapter\undefined \section*{解答}\else
    \chapter*{解答}\fi\input{\jobname.kai}}
\makeatother
\begin{document}
\NewProblem
\begin{mondai}
この問題を解け。
\kaito{解答だよだけど、改行とかは出力されないよ。}
\end{mondai}
\begin{mondai}
この問題をとけ。
\kaito{解答だよだけど、改行とかは出力されないよ。}
\end{mondai}
\KAITO
\end{document}
```


うーん、これだけでは少しさびしいと思われますので、もう少し色をつけます。これでも動くのですが、「解答ファイル」には verbatim 環境のように、記述した内容がそのまま出力されてほしいものです。そこで、今回はこの改造を施します。

2.27　ハイフン

ハイフンは通常和文のみには使われることのない記号です（ダッシュと混同しないようにしましょう）。ハイフンは連字符の働きをし複合語や、行頭・行末の揃えにも使われています。欧文のハイフンは次のようにエックス・ハイト (ex) の中央に来るように設計されています。

x-x, Future-University, A-B-C, p-q-g, H-H

ただし、大文字の間に混入する場合は天地（ボディ + アセンダー）の中央に移動するように調整しておきましょう。さらに左右にシンスペース\, 程度を挿入すると良いそうです。

```
NKH-Hakodate $\neq$ NKH\lower-.2ex\hbox{-}Hakodate
$\neq$ NKH\,\lower-.2ex\hbox{-}\,Hakodate
```

NKH-Hakodate ≠ NKH-Hakodate ≠ NKH - Hakodate

文字 'H' のみを考慮するならば次のようなマクロも書けるでしょう。

```
\makeatletter
\newcommand*{\hh}[1][H]{%
```

```
  \setlength\@tempdima{#1}%
  \@tempdima=0.5\@tempdima
  \advance\@tempdima-0.5ex
  \advance\@tempdima-0.2pt
  \lower-\@tempdima\hbox{-}}
\makeatother
```

全てのアルファベット大文字と全ての数字は、アッセンダいっぱいの高さがあることになっています。そのため、このマクロはアルファベット大文字と数字の組み合わせならば、なんにでも使えるはずです。他のアルファベット小文字と大文字が混同する場合などは、小文字を優先し、調整しないほうが無難です。ハイフンの両側の文字がアッセンダいっぱいの時のみにしましょう（アルファベットは都合の良いことに、小文字の場合は字形の重心がボディの中心に、大文字の場合は高さの中心になるようにデザインされています。いや、エックスハイトの 1/2 かもしれません）。

ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
01234567890

しかし、実際には使用している書体がどのようにデザインされているのかを把握することも重要であり、その書体に合わせたハイフンの適切な位置を調整する必要があります（書体によってそれぞれのグリフが異なるため）。

2.27.1 ハイフネーション

TeX のハイフネーションを実現するハイフネーションアルゴリズムは

そもそも欧文のハイフネーションはその単語の品詞によって違うことが問題となります。名詞では「贈り物」を意味する ‘present’ は ‘pres-ent’ になりますが、これが「贈呈する」の動詞になると、‘pre-sent’ になり、アクセントも変わります。もし、誤ったハイフネーションを組版してしまった場合は、文の構造までもが変わってしまうので、ハイフネーション一つをとっても難しい問題です。このような問題に対しては、最低でも構文解析等が必要になり、より完成度を高める場合は意味解析なども要求されます。しかし、それらの処理を実装することは非常に難しいことで、いわゆる自然言語処理の領域に手を染めなければなりません。TeX ではもう少し簡易なアルゴリズムでハイフネーションを行っています。簡易といてもそれは Frank Liang 氏によって発見された優秀なアルゴリズムで、精度が 89.3 % と極めて高くなっています。これは実際の単語の使用頻度を考えると、正確性は 95 % を超えるとも TeX の作成者の著書で述べられています。

TeX はハイフネーションをその辞書の中から検索し、パターンが一致したものを採用します。そのため、新語などにもある程度対応します。もし TeX に登録されていない単語でも\hyphenation コマンドで大域的に指定することができます。

```
\hyphenation{ho-ge pi-yo ge-ma a-rya}
```

2.28　空白とグレー

2.28.1　字間

字間、文字間、ワードスペース、ワードスペーシング。

いわゆる最近多いベタ組み、最悪です、あれは。欧米ではレタースペーシングと呼ばれています。

決定要因は非常に多く、可読性と視認性の両方を満足しなければいけません。例えば

書体、フォントサイズ、判型、版面、文章量、文章の特性

などが考えられますが、ほかにも多くの考慮すべき項目があります。詳しくは *Grid* を読んでください。

2.28.2　語間

語間、単語間、ワードスペース、ワードスペーシング。

2.28.3　行間

いわゆる「グレー」と呼ばれる、全体の雰囲気を決定する空白です。これが狭ければ版面は濃いグレーになりますし、広ければ薄いグレーになり、印象は軽くなります。

この範囲の話は前編『好き好き LaTeX 2_ε 初級編』と重複する部分もあるので、もう少しまとめておこうと思います。

2.28.4　グレー

さて、ここでページの印象を決めるグレーについて定義しましょう。人間は本を読むときにそのページ全体を長い時間見つめます。1 行あたりの行の長さを l_w、1 ページ辺りの高さ（1 行の高さと余白を含む）を l_h とします。横書きの場合のグレー率 G は、1 行辺りに使用されている漢字の存在確率を p_k、1 行辺りに使用されている仮名文字の存在確率を p_j、ベースライン以上の文字の高さを $\bar{h}$、文字の幅（アルファベットなどの英数字を除く）を $\bar{w}$、字送りを s_c、行送りを s_b とすると式 2.1 となります（かなり適当ですから当てにしないでください）。

$$G = l_w l_h - \left\{ l_w (s_b - \bar{h})(\frac{l_h}{s_b} - 1) - \left[s_c \bar{h} \left(\frac{l_w + s_c}{\bar{w} + s_c} - s_c \right) - \frac{1}{2} \bar{w} \bar{h} (\alpha p_k + \beta p_j) \frac{l_w + s_c}{\bar{w} + s_c} \right] \frac{l_h}{s_b} \right\} \tag{2.1}$$

バーがついている値はそれぞれ平均値を表し、変数 α は平均的に漢字が紙面を塗りつぶす領域、β はその平仮名の場合です。例えば（創造したくもありませんが）ゴシック体を本文に使っている場合は α と β の値が非常に高くなります（ウェイトの大きい書体の場合も同様）。

もちろん句読点やその他の記号、また欧文などの多言語が混入した場合はさらにパラメータが変化します。

もし、漢字と平仮名の間で幅や高さが違う場合は、文字の幅と高さがパラメータ $\bar{w}$ と $\bar{h}$ だけではなくなります。

2.29　書体あれこれ

文字は何千年も用いられている媒体（メディア）であり、他のメディアに比べて非常に洗練されており、正確性が高いものとなっています。それぞれのメディアには各々の特徴があります。電話というメディアには同期や双方向性などの特徴があります。文字には双方向性や同期はありませんが、きちんとした手順を踏めば情報の正確性が向上します。

「洗練されている」といっても、それは一つ一つの字形が洗練されているだけで、文字を一つの文書に**どのように**配置するかについては、編集者のセンスに任されているのが現状です。

それもそのはずで、文書に存在する全ての文字をどのように配置するかという問題は次の考慮すべき項目を含んでいるためです。

書体　一連の字形に用いられている文字の様式。その言語の文化や、その文書の歴史を反映するものでもある。デバイス依存の部分で TeX が直接的に関与することが難しい。歴史的文書の場合はその当時使われていた手書きの書体に似たものを使うこともある。

フォントサイズ　書体の大きさ。

判型　用紙の大きさ。経済的問題、技術的問題も関与している。

マージン　用紙の左右上下の端に存在する文字が何も書かれていない領域。絵画で言えば額縁にあたる重要な部分。用紙と文章の境界を表す。

版面　判型から左右上下のマージンを差し引いた実際の文章領域 (type area)。

文章量　一つの文書に含まれる文字の数。

文章の特性　平仮名の多い文章か、漢字の多い文章か、図と文章が密接に絡み合っているかなどの特性。

ほかにも紙媒体として作成するときの考慮すべき点が沢山あります。

印刷方式　凸版、凹版、平板など。今はオフセット印刷による平板が主流。

インク　インクの色、粘り強さ、品質など。

紙　書籍の場合はこの紙が重要で、耐久性があるかどうか、光沢があるかどうかも問題

となる。

装丁　実際に人間が手にとって読むときに読みやすいかどうか。

2.30　現代のアウトラインフォントが抱える問題点

2.30.1　リニアスケーリング

そもそも各フォントサイズに合わせてデザインされるべきフォントが、昨今の流れでは、あるサイズ（たとえば 10 pt) 用にデザインした字形を、全てのサイズに適用しています。Knuth はこのようなリニアスケーリング（色々呼び方がありますが、ここでは単純幾何学変換と呼ぶことにしましょう）の手法ではなく、きちんとサイズごとのフォントをデザインしています（これをオプティカルスケーリング、視覚的調整と呼ぶことにしましょう。LaTeX でも同様に、あるフォントを何倍して文章中で用いる、という方針ではなく、現在のフォントのある固定倍で拡大したサイズ（`\large` とか`\tiny` など）が用いられているのは、このような理由もあるからです（処理の都合上、DIN 規格の A 配列は数字が一つ増えるごとに $(\sqrt{2})^{-1}$ 倍されるので、フォントサイズもおおよそその倍数のサイズがあると非常に都合がよいのです)。

そのフォントの種類に関わらずある文字を単純に一次変換しただけでは、つまり過ぎたり、あきすぎたりすることは自明のことです（サイズごとにフォントを変えたのは可読性を追求した Knuth の最適な選択だと思われます)。

まず、図 2.3 に Knuth がデザインしたフォントを示します。 図 2.3 ではそれぞれの

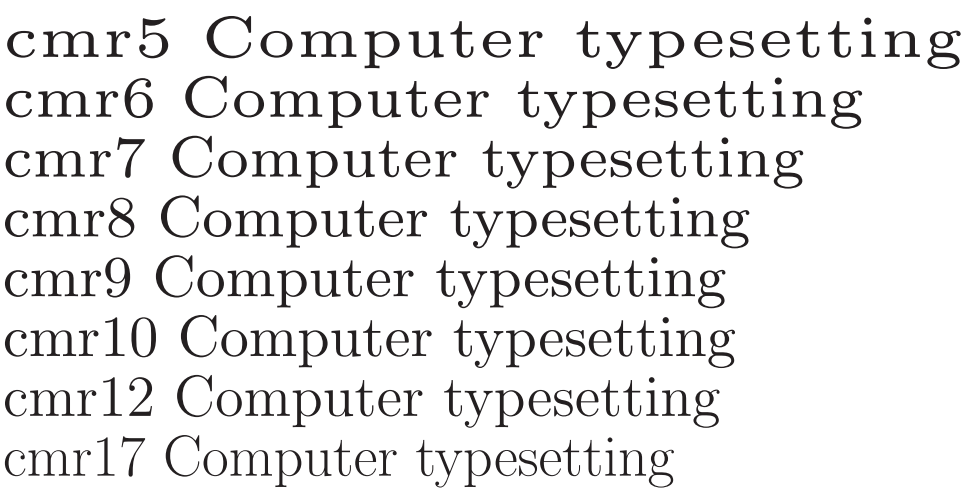


図 2.3　各サイズ用にデザインされた Computer Modern フォント

フォントを分かりやすいように 20 pt に拡大しています。

相当手間隙をかけて作成された Young Ryu 氏の `TXfonts` ではすべてのサイズに同じフォントが使われます（これは `TXfonts` に限った話ではありませんが、どうも Palatino や Times を好き好んで使う人たちがいるので)。これは `ot1txr.fd`（または `t1txr.fd`）

などをみると次のようにすべてのサイズに同じフォントを使うようになっています。

```
\DeclareFontShape{OT1}{txr}{m}{n}{<->txr}{}
```

標準の LaTeX の場合は次のようになります（ot1cmr.fd によります）。

```
\DeclareFontShape{OT1}{cmr}{m}{n}%
    {<5><6><7><8><9><10><12>gen*cmr%
     <10.95>cmr10%
     <14.4>cmr12%
     <17.28><20.74><24.88>cmr17}{}
```

この場合はなぜか、5–10 pt と 12 pt に cmr.*が、10.95 pt に cmr10.*が、14.4 pt に cmr12.*などが使われるようになります。個人的にこの定義が好きではないので、次のようにすることもあります。ただし、この場合は PS ファイルや PDF に埋め込むべきフォントが増えます（本当の本作りをしている場合は、下の定義のようなことも配慮すべきでしょう）。


```
\DeclareFontShape{OT1}{cmr}{m}{n}{%
        <-6>    cmr5
        <6-7>   cmr6
        <7-8>   cmr7
        <8-9>   cmr8
        <9-10>  cmr9
        <10-12> cmr10
        <12-17> cmr12
        <17->   cmr17}{}
```

これと同じことが type1cm パッケージで行われています。この場合、システムにそれぞれのサイズの True Type フォント（Postscript Binary 形式フォント）が導入されている必要があるでしょう（BlueSky/Y&Y が配布しているものを使うと良いでしょう）。

ためしに視覚的調整された Computer Modern Roman フォントと単純に幾何学一次変換を施した Times Roman を見比べてください。 フォントサイズが小さくなると当然のように太くなります。さらに幅が狭く感じます。CM の方ではサイズが小さくなるにつれて幅広で細く (light) になります。大きくなると感覚的には「引き締まった感じ」になります、要するに広がり過ぎないように、太り過ぎないようになっています（例では分かりませんが、6 pt 以下？のフォントには合字が使われていません、小さいフォントや太いフォントで合字を使うと視認性が悪くなるので）。Times Roman を単純幾何学変換しただけではフォントサイズが大きくなると太りすぎですし、引き締まった感じもありません

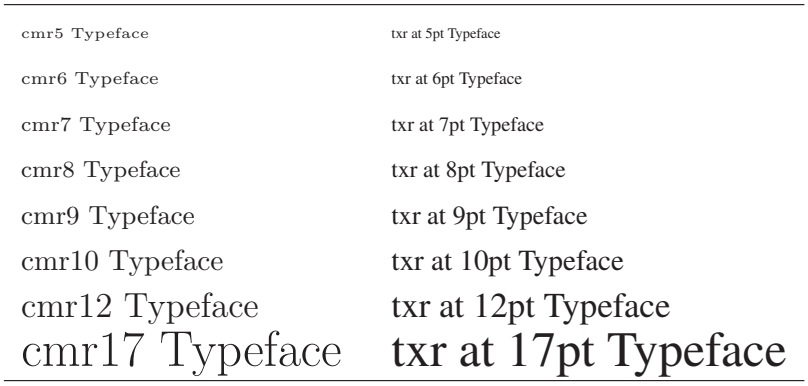


図 2.4　Computer Modern Roman と Times Roman

(Times Roman は良いフォントなのですが、このようにリニアスケーリングにすると無様に見えてしまいます)。

(ベースがきちんとしているので) 実は個人的には Latin Modern がお気に入りだったりします (本当は Minion が一番です、とか言ってみたり)。

まぁ、特に不都合がなければプリアンブルに


```
\usepackage{type1cm}
```

という記述か、好みに応じて

```
\usepackage{lmodern}
```

と書くと、視覚的調整されたフォントがサイズごとに用いられるようになります。記述を省略したときの結果を見比べてください (LaTeX では多くの人間を対象とするので標準的な設定になっているので、ローカルでの最適な設定になっていない部分もあります、特にデバイス依存な部分)。

2.30.2　画像ファイルの読み込みの工夫

別に画像がないときはそのファイルを読み込んでくれなくても良いので、\IfFileExists 命令を使って、何か命令を定義しましょう。

```
\newcommand*\Image[1]{\IfFileExists{#1}{あるよ}{ないよ}}
\Image{hoge.eps}
\renewcommand*\Image[2][width=.8\linewidth]{\IfFileExists{#2}{%
  \includegraphics[#1]{#2}}{図版張り込み予定}}
\begin{figure}[htbp]
\begin{center}
\Image[width=.3\linewidth]{hoge.pes}
```

```
\caption{ほげほげ}
\end{center}
\end{figure}
```

まぁ、色々応用できるでしょうから、例題程度に。

2.31 クラスファイルの設計

2.31.1 最小構成

いきなり大規模なクラスファイルを設計仕様と思っても無理なので、まずは最小構成のクラスファイルを作ります。そこで次のクラスファイル hoge.cls を作成します。

```
\ProvidesClass{hoge}[2004/08/09 LaTeX2e class file]
\renewcommand\normalsize{\fontsize{10pt}{12pt}\selectfont}
\endinput
```

実はこの 2 行があればうまく動きます。多分そのうち日本語 TeX にも依存したクラスになるだろうと思いますから\NeedsTeXFormat を宣言しておきます。


```
\NeedsTeXFormat{pLaTeX2e}
```

これであなたもクラス設計者になれました。

しかし、これだけではあまりにもさびしいので、今まで体裁を調整する命令を活用します。何か目標があったほうが良いので、今回は用紙が A4 で縦位置、フォントサイズが 10 ポイント、1 行 44 文字、1 ページ 36 行のレポート用のクラスファイルにします。ページ番号はページ下部中央にアラビア数字で出力します．すると先ほどのクラスファイルは次のようになります。

```
\NeedsTeXFormat{pLaTeX2e}
\ProvidesClass{hoge}[2004/08/09 LaTeX2e class file]
\renewcommand\normalsize{\fontsize{10pt}{18pt}\selectfont}
\normalsize
\setlength\textwidth{44zw}
\setlength\textheight{36\baselineskip}
\parindent=1zw
\pagenumbering{arabic}
\pagestyle{plain}
```

まぁ、何ができるか試してくださいな。フォントサイズを変更するコマンドや見出しの命令なども使えないことでしょう。ただ、通常のクラスファイルとちがって、このままでは

```
Underfull \vbox (badness 10000) has occurred while \output is active [1]
```

みたいなエラーが出ることになります。

2.32　LaTeX 標準パッケージ概説

2.32.1　indentfirst

LaTeX は見出しの後の字下げを\if@afterindent なるブール変数で制御しています。欧文のクラスファイルでは\chapter の後や\section 後は時下げしないスタイルもあります。LaTeX の article, report, book などのクラスファイルは\chapter や\section の見出し直後に字下げしないようになっています。これを簡単に字下げするようにするには「字下げしない」という命令に対して「字下げする」という命令を代入すれば良いことになりますので、次の 2 行を書くだけになります。

```
\let\@afterindentfalse\@afterindenttrue
\@afterindenttrue
```

どうです? 簡単ですよね? David Carlisle 氏による indentfirst も全く同じ記述になっています。

2.32.2　alltt

べた書き環境の verbatim 環境がありますが、タブ記号や改行記号などを出力したいときにはちょっと不便です。この場合は Johannes Braams 氏の alltt パッケージが使えます。これは、'\'、'{'、 '}' の三つの命令だけがそのまま使えるという alltt 環境を提供します。これと同じような環境を自分でも定義してみましょう（ただし、様々な状況を全く無視した簡易版です)。まず '\'、'{'、 '}' 以外の文字はカテゴリーコードを変更するために\mysanitize 命令を定義します（\@sanitize 命令はバックスラッシュのカテゴリコードも変更するので、ここでは使いません)。

```
\makeatletter
\newcommand*\mysanitize{%
  \@makeother\$ \@makeother\& \@makeother\# \@makeother\^%
  \@makeother\_ \@makeother\% \@makeother\~}
\makeatother
```

次に対象となる myTT 環境を定義します。安全のため trivlist 環境の中に入れます。さらに、定義の変更を局所的にするために\begingroup と\endgroup でサンドイッチします。さらに先ほどの\mysanitize 命令と\ttfamily 命令を使用します。ここで問題になるのは改行文字^^M ですが、これはすでに\obeylines 命令を使うことで改行文字を改段

落\par 命令を用います。ただし、改段落にするので\parindent には 0 pt を代入しておきます。

```
\newenvironment{myTT}{%
  \trivlist\item\relax\begingroup%
    \ttfamily\mysanitize%
    \obeylines\parindent=0pt}%
  {\endgroup\endtrivlist}%
```

これで myTT 環境の定義に終了です。

```
\begin{myTT}
$ & # ^ _ % ~ \( x\sp{2}\neq x\sb{2} \)
This line will be in new line.
\end{myTT}
```

さて、実際には alltt パッケージではもう少し丁寧なことをやっているので、興味のある人は見ておいてください。

2.32.3　afterpage

LaTeX の浮動体 (floats) 制御では満足の行く場所に、図表を配置することができないことがしばしばあります。このような場合\clearpage を使うことが考えられますが、ページに対する影響が多いためお得とは言いがたいでしょう。この場合とりあえず、次のページから図表を配置してほしいという命令があれば助かります。これを可能にするのが David Carlisle 氏の afterpage パッケージで、

```
\afterpage{\clearpage}
```

のように使います。さらにある図を現在のページが組み終わった後に出力したいというならば

```
\afterpage{\clearpage\begin{figure}[!h] ... \end{figure}}
```

のように使うことができます。

実際、この afterpage パッケージは TeX の\output などと関わっているため、この冊子では深く取り扱いません。

2.32.4　xspace

用語などを統一するという用途で次のような命令を定義することがあります。

```
\newcommand*\pdftex{PDF\TeX}
```

これを次のように用いてしまうと、意図しない出力になります。

```
\pdftex is a very large system.
```

PDFTEXis a very large system.

このような場合は次の文字がスペースの場合は自動的に空白を挿入し、句読点や約物の場合はスペースを入れないようにしてほしいものです。実際に David Carlisle 氏の xspace パッケージでは次のような処理をしてします。

```
\DeclareRobustCommand\xspace{\futurelet\@let@token\@xspace}
\def\@xspace{%
  \ifx\@let@token\bgroup\else
  \ifx\@let@token\egroup\else
  ...
  \ifx\@let@token\@sptoken\else
 \space
 \fi\fi ... \fi}
```


この中に登録されていないスペースを入れてほしくない文字は、自分で追加することができます。

先ほどの記述は次のようになります。

```
%\usepackage{xspace}
\renewcommand*\pdftex{PDF\TeX\xspace}
\pdftex is a very large system.
He may be a \pdftex{}nician.
```

PDFTEX is a very large system. He may be a PDFTEXnician.

2.32.5　版面を構成するパラメータ

さて、前回は次のようにしてクラスファイルを作成しました。

```
\NeedsTeXFormat{pLaTeX2e}
\ProvidesClass{hoge}[2004/08/08 pLaTeX2e class file]
\renewcommand\normalsize{\fontsize{10pt}{18pt}\selectfont}
\normalsize
\setlength{\textwidth}{40zw}
\setlength{\textheight}{36\baselineskip}
```

この段階で

```
Underfull \vbox (badness 10000) has occurred ....
```

なるエラーになります。まぁ、原因は版面を構成するパラメータを認識できていないことになります。とりあえず、エラーを回避するだけならば\textheight を次のように

$$\textheight = n\baselineskip + \topskip$$

とするとうまく行きます。n は適当に 1 ページ何行にするかを決めます。\baselineskip は行送りのスキップです。\topskip はページ上部に挿入するスキップです。とりあえず、この二つを\textheight に代入するとなんとかなるでしょう。

よってエラーを回避するためのクラスファイルは次のように記述します。

```
\NeedsTeXFormat{pLaTeX2e}
\ProvidesClass{hoge}[2004/08/11 pLaTeX2e class file]
\renewcommand\normalsize{\fontsize{10pt}{18pt}\selectfont}
\normalsize
\setlength{\textwidth}{40zw}
\setlength{\textheight}{35\baselineskip}
\setlength{\topskip}{1\baselineskip}
\addtolength{\textheight}{\topskip}
```


このようにすると 1 ページ辺り 36 行文の行送りが取られることになります。

版面を構成するパラメータは次のようになります...

2.32.6　普通の文字の大きさの定義

LaTeX のクラスファイルのなかで、文字の大きさを変更するための命令を作ります。基本として用意すべきなのは\magstep の整数倍の大きさに変更するような命令です[*3]。

クラスオプション　通常ドキュメントクラスに渡すオプションは、そのクラスファイルのみに渡されるのかというと実は違います。渡されたオプションは\usepackage で読み込まれているマクロパッケージに照合されます。そのため、次のような宣言もできることになります。

```
\documentclass[titlepage,dvipdfm,T1]{jarticle}
\usepackage[usenames]{color}
\usepackage{graphicx}
```

[*3] ただし、当該フォントのファミリーがどのように宣言されているかで対応も変わります。

文中の上付き添え字　例えば‘\texttrademark’で™を出力するように、添え字は何かと文中でも使うものです。この場合は\textsuperscript を使うとうまく行きます。

```
\let\tsp\textsuperscript
詳細は参考文献\tsp{(1,3}を参照してください。
```

詳細は参考文献(1,3を参照してください。

合字　何らかの原因で合字（エンコーディングの問題などで）が使えないときは、それぞれ次のコマンドで代替することができます。

通常の命令	代替命令	合字
---	\textemdash	—
--	\textendash	–
!‘	\textexclamdown	¡
?‘	\textquestiondown	¿
‘‘	\textquotedblleft	“
’’	\textquotedblright	”
‘	\textquoteleft	‘
’	\textquoteright	’

たまにドイツ語などで合字を殺したいときは、\textcompwordmark 命令を使うこともできます。これはハイフネーションされません。ハイフネーションされても良い場合は\-を使うと良いでしょう。

```
shelfful, shelf\textcompwordmark ful, shelf\-ful\par
```

丸付き文字　丸付き文字は色々な出力方法があります。LaTeX にあらかじめ用意されているのは\textcircled のみです。

```
\let\MA\textcircled
\MA{\scshape r}, \MA{c}, \MA{A}, \MA{8},
\copyright, \textregistered
```

Ⓡ, ⓒ, Ⓐ, ⑧, ©, ®

なんだかちょっと騙された感じがあるので、他の命令を探したくなります。既存のものとしては奥村晴彦氏の okumacro に含まれている\MARU 命令などを使うことでしょう。これは囲みたい文字の大きさに応じて丸を graphicx で拡大するという方法を取っています。

```
%\usepackage{okumacro}
\MARU{R}, \MARU{1}, \MARU{あ}
```

Ⓡ, ①, ㊥

標準パッケージの紹介　alltt, doc, exscale, fontenc, graphpap, ifthen, inputenc, latexsym, makeidx, newlfont, oldlfont, showidx, syntonly, tracefnt

tools パッケージの紹介　array, calc, dcolumn, delarray, hhline, longtable, tabularx, bm, enumerate, fontsmpl, ftnrigth, indentfirst, layout, multicol, rawfonts, somedefs, showkeys, theorem, varioref, verbatim, xr, xspace

ファイル出力　一つの配布ファイル hoge.tex から、別のファイルに出力したいときは\filecontents 命令を使います。コメントをつけてほしくないときは\filecontents*を使います。

```
\begin{filecontents}{hoge.txt}
私はこれから山に芝刈りに行きます。
\end{filecontents}
\documentclass{jsarticle}
\begin{document}
\section{芝刈り}
\input{hoge}
\section{2 度目の芝刈り}
\input{hoge}
\end{document}
```


エラー出力　プリアンブルに

```
\setcounter{errorcontextlines}{1}
```

によってエラーの出力される量を調整することができます。標準は −1 です。

読み込まれているファイルを確認する　現在タイプセットしているファイルで使用されているファイル（プリロードも含む）を調べるには\listfiles 命令を使います。

宣言型命令は環境にできる　それ以降、何らかの属性を変更する宣言型命令は環境にすることができます。

```
\begin{ttfamily}
ここは Typewriter 体になるんでしょうか?‘
\end{ttfamily}
```

ここは Typewriter 体になるんでしょうか¿

版面の高さ調整　ある 1 ページだけ版面の調整をしたいときは`\enlargethispage` 命令を使うと、そのページの版面の高さを変えることができます。`\enlargethispage*`は垂直方向の空白をなるべく縮めるようにします。