

L^AT_EX 入門

吉永徹美

2006年3月22日

(version 1.3.1)

“T_EX” は American Mathematical Society の商標です .

“POSTSCRIPT” は Adobe Systems Inc. の商標です .

その他 , 本稿に現れるシステム名・製品名などは一般に各社の商標です .

なお , 本稿では “TM” , “®” などの表示は行っていない .

目 次	i
目 次	
目 次	i
序（お断り）	1
1. L ^A T _E X による文書作成のしくみ	1
2. L ^A T _E X 文書の枠組	3
3. L ^A T _E X 文書の本文の書き方	5
3.1. ソースファイル中の改行	5
3.2. 改行位置の指定	6
3.3. 改段落	7
3.4. ソースファイル中の改行文字・空白文字の取り扱い	8
4. 文書クラス	9
4.1. 主な文書クラス	9
4.2. 文書クラスの主なオプション	10
5. 特殊文字の出力法	11
5.1. T _E X の特殊文字の出力	11
5.2. テキスト用記号類	12
5.3. テキストでも用いられることが多い数式記号	13
5.4. テキスト用アクセント記号	13
5.5. 丸や枠で囲んだ文字	13
6. 書体・文字サイズの変更	15
6.1. 文字サイズの変更	15
6.2. 書体の変更	15
7. 文書の論理構造（章・節など）	18
7.1. 文書の構造を指定するコマンド	18
7.2. 見出しの与え方に関する補足	20
8. 文書の視覚的構成（右寄せ・左寄せなど）	22
8.1. テキストの右寄せ・左寄せ	22
8.2. テキストを書いてあるとおりに出力する方法	24
8.3. “引用” 風の環境	25
8.4. 幅を指定した段落	26
9. 箇条書き	27
9.1. 番号のない箇条書き	27
9.2. 番号付き箇条書き	29

10. 表 (1) [tabular 環境の基本事項]	32
10.1. tabular 環境による表	32
10.2. 表の体裁の変更	35
11. 表 (2) [tabular 環境の応用と table 環境]	37
11.1. 複数の列をまとめること・部分的な罫線	37
11.2. 幅を指定した表	40
11.3. 表の配置	40
12. 図 (1) [picture 環境・figure 環境]	43
12.1. picture 環境の書式	43
12.2. 線分・矢印	44
12.3. 円・円板	45
12.4. 線の太さの変更	46
12.5. 図への文字列の書き込み	47
12.6. 同じものの繰り返し	48
12.7. ベジエ曲線	49
12.8. 枠囲み	50
12.9. 図の配置	52
13. 図 (2) [graphicx パッケージ]	53
13.1. 図およびテキストの回転・拡大	54
13.2. eps ファイルの取り込み	56
14. 脚注	58
15. 相互参照	60
15.1. 相互参照の方法	60
15.2. 相互参照のしくみ	61
16. 目次	63
17. 参考文献の作成	64
18. 著者名・タイトルの表示 / 概要の表示	65
19. ページスタイル・2 段組	67
19.1. ページスタイルの指定	67
19.2. 2 段組・多段組	68
20. 空白の調整	69
21. 改行・改ページの起こりやすさの調節	71
22. 数式 (1) [L ^A T _E X における数式]	72

目 次	iii
23. 数式 (2) [添字・分数]	74
23.1. 添字	74
23.2. 分数	75
24. 数式 (3) [根号・省略記号]	77
24.1. 根号	77
24.2. 省略記号	78
25. 数式 (4) [演算子]	79
25.1. 2 項演算子	79
25.2. 関係演算子	79
25.3. 大型演算子	82
25.4. 関数名	83
26. 数式 (5) [括弧]	85
27. 数式 (6) [数式用記号]	89
27.1. ギリシャ文字・ヘブライ文字	89
27.2. 数式用の雑多な文字	90
27.3. 矢印	91
27.4. 数式用アクセント	91
28. 数式 (7) [数式での書体変更]	92
29. 数式 (8) [数式のスタイル]	96
30. 数式 (9) [下線など]	98
31. 数式 (10) [数式番号付きのディスプレイ数式]	99
31.1. equation/eqnarray 環境の基本	99
31.2. eqnarray 環境の応用	101
32. 数式 (11) [行列]	106
33. 数式 (12) [数式での空白の調整]	108
34. 定理型の環境	109
35. 大規模な文書の作成	112
36. 索引の作成	114
36.1. 索引を作成する方法の概要	115
36.2. 索引項目と“読み方”の指定	116
36.3. 索引項目の階層化	117
36.4. 索引でのページ番号の書式の指定	118
36.5. mendex の特殊文字を含むような索引項目	119
36.6. 索引の体裁の (簡単な) カスタマイズ	120

補遺 A. amssymb パッケージが提供する記号	122
A.1. 2 項演算子	122
A.2. 関係演算子	122
A.3. 矢印類	123
A.4. その他の記号	124
補遺 B. テキストの色付け	124
B.1. 色指定の方法	125
B.2. 背景色の指定	126
参考文献	127

序（お断り）

本稿は、 \LaTeX ($\text{\LaTeX} 2_{\epsilon}$) を用いた文書作成の基本についての解説を試みた筆者のウェブページを \LaTeX 文書化したものです。ただし、本稿は、 \LaTeX システムがすでにインストールされていて、かつ、正常に使用できるように環境設定が行われていることを前提として、 \LaTeX 文書の作成方法の解説を行うものです。 \LaTeX システムのインストールに関しては、適当な (CD-ROM 付きの) 書籍あるいは“インストールガイド”的なウェブサイト¹⁾を参照してください。

あと、本稿では原則として横書きの文書を扱います。縦書きの際に本稿の記述とは異なる挙動を示すコマンド (例えば、`\cleardoublepage`) があるということをあらかじめご了承ください。なお、文字 ‘\’ は環境によっては ‘¥’ のように表示されます。本稿では特に ‘¥’ を用いる必要がある場合以外は ‘\’ を用いています。必要があれば、環境に応じて適宜読み替えてください²⁾。

1. \LaTeX による文書作成のしくみ

ここでは、 \LaTeX を用いて文書を作成・印刷（または画面表示）する一連の操作の概要を説明します。

\LaTeX の文書ファイルは、例えば、

```
\documentclass[12pt]{article}
\begin{document}
Hello \TeX!
\end{document}
```

のような形式（細かいことは後述します）のテキストファイルで、ファイル名には普通は “.tex” という拡張子をもつ名前をつけます。実際に、手元にあるテキストエディタで上記の内容をもつファイルを作成し、example.tex という名前で保存しましょう。なお、テキストエディタでなくても、文書をテキストファイルとして保存することができるワープロソフトを用いても構いません。

次に、今作成したファイル example.tex を \LaTeX で処理します。具体的な方法は、“シェルのコマンドプロンプトから platex example (または jlatex example) と入力する”、“ \TeX の統合環境のメニューから、‘ \TeX で処理’ に相当する項目を選択する”などさまざまでしょう。この操作で、example.aux, example.dvi, example.log というファイルが新しく作られます。（処理した example.tex の拡張子を “.aux”, “.dvi”, “.log” に変えたものです。）これらのうちの example.dvi に、処理結果が画面表示や印刷を行える形で記述されています。（log ファイルには、文書ファイルを処理した際に表示されるメッセージなどが記録されます。また、aux ファイルには、相互参照 (§15 参照) や目次の作成 (§16 参照) で用いる情報などが記入されます。）以後、この \TeX の文

¹⁾web 上の、実用に耐える (\LaTeX システムの) インストールガイドは数多く存在するので、探すのに困難はないはずです。

²⁾‘\’ と ‘¥’ の問題は \TeX に関するのではなく、むしろ“パソコン入門”の類で説明されていて然るべきことです。そこで、必要があれば“パソコン入門”の類の解説書を参照してください。

書ファイルから dvi ファイルなどを作成する操作をコンパイル³⁾ともいいます．最後に、適当なプレビューア⁴⁾を起動して、example.dvi を表示させてください．ここまでの操作がうまくいってれば、次のような文がページの上部に表示され、ページの下の方にはページ番号が表示されるはずです．

Hello \TeX !

また、印刷については、プレビューアによっては印刷を行うことができるもの（例えば dviout）もあります．一方、POSTSCRIPT プリンタを常用する環境では、dvips などを用いて POSTSCRIPT ファイルに変換した後、プリンタに出力することになります⁵⁾．なお、dvi ファイルを読み取って画面表示や印刷などを行うソフトウェアを dviware と総称します．

上記のことをまとめると、次のようになります．

テキストエディタを用いて (\LaTeX 形式の) 原稿ファイルを作成する



\LaTeX で処理する (→ dvi ファイルが作成される)



dviware で dvi ファイルを処理する



処理結果の表示または印刷

また、 \LaTeX で文書ファイル进行处理しているときに、“?” というメッセージが表示された状態で止まることがあります．これは、処理中に（文書ファイルの書き間違いなどの理由で）エラーが起こったということを意味します．そのときは、何回か単に [ENTER]（または [RETURN]）キーを押して、処理を続けさせてみてください．それでもしつこく “?” と表示されるようなら、“x” と入力してください．（間違えても “q” とは入力しないでください．“x” というのは “ただちに処理を終了しなさい” という指示で、“q” というのは “メッセージを全く表示せず、エラーが起こってもひたすら無視しなさい” という指示です．）

一方、 \LaTeX での処理が “*” というメッセージを表示した状態で止まってしまった場合には、とりあえず、次のようにしてください．

- `\end{verbatim}` と入力します．
- それでも “*” と表示されるようなら、`\end{verbatim*}` と入力します．

多くの場合、これで “?” というメッセージが表示されるので、そこで “x” と入力すれば処理を終了できます．それでもまだ “*” と表示されるようなら、さらに `\stop` と入力します．こうすると、たいていの場合ただちに処理が終了しますが、それでも “*”

³⁾プログラミングからの類推です．

⁴⁾dvi ファイルを読み込み、画面に表示するソフトウェアのことを（dvi ファイルの）プレビューアといいます．広く用いられているものとしては dviout, xdvi などがあります．

⁵⁾POSTSCRIPT プリンタへの出力の具体的なことについては、周囲にいる（読者が使用しているプリンタを実際に触れる）人のうちすでに \TeX を用いている人に聞いた方が確実です．

と表示される場合には, “ファイルの終端” を表す文字⁶⁾を (何回か) 入力します. それでも処理が止まらない場合には, ([CTRL]+C などを入力して) L^AT_EX を強制終了させてください. (なお, L^AT_EX の処理の際に “*” を表示したまま止まった状態になるというのは, 文書ファイルの終端まで読み込んでも “T_EX 文書の終わり” を表すコマンドが見つからなかった, ということを表します.)

2. L^AT_EX 文書の枠組

前節では, 次の例を考えました.

```
\documentclass[12pt]{article}
\begin{document}
Hello \TeX!
\end{document}
```

最初の `\documentclass[12pt]{article}` というのは, 文書クラス (“章” の有無といった文書構造の規定と, 各種の見出しの出力形式といった体裁の指定をあわせたもの) として “article” を用い, (地の文における) 文字の大きさを 12 ポイント (1 ポイントは約 0.351 mm) にする, という意味です. 文書クラスについては §4 で扱います. ここでは, すでに用意されているスタイルと文字サイズを指定している, と理解しておけば充分です. その次の行の `\begin{document}` は, “ここから本文が始まる” という意味の記述です. その 2 行下の `\end{document}` は “ここで文書は終わり” という意味の記述です.

今説明した, `\documentclass` の指定, `\begin{document}` と `\end{document}` はすべての L^AT_EX 文書に必要です. また, `\begin{document}` と `\end{document}` がそれぞれ本文の始まりと終わりを表すので, この二つの記述の間に文章を書くわけです. 例えば,

```
\documentclass[12pt]{jarticle}
\begin{document}
\LaTeX{}ではここに本文を書きます.
\end{document}
```

という内容のファイル进行处理すると,

L^AT_EX ではここに本文を書きます.

のような結果が得られます. この例では, `\documentclass` の指定が `jarticle`⁷⁾ になっていますが, これも, すでに用意されている文書クラスのうちの一つです.

L^AT_EX の文書ファイル (これ以降, ソースファイルということもあります) の枠組をもう一度書くと, 次のとおりです.

```
\documentclass[<options>]{<classname>}
```

⁶⁾EOF コード. [CTRL]+Z や [CTRL]+D など, OS によって異なります.

⁷⁾JL^AT_EX を用いる場合には `j-article` にしてください.

```
\begin{document}
((本文))
\end{document}
```

ここで、 $\langle classname \rangle$ は “article” , “jarticle” などのことで、 $\langle options \rangle$ は “12pt” の部分のことです。($\langle classname \rangle$, $\langle options \rangle$ としてどのようなものが使えるかについては、やはり §4 で扱います。) なお、`\begin{document}` よりも前の部分をプリアンブル (preamble) といいます。(ただし、文脈によっては、`\documentclass` コマンドと `\begin{document}` の間をプリアンブルということもあります。)

また、L^AT_EX では各種の拡張機能は “パッケージ” という形で提供されます。例えば、補遺 B で扱うような色付けは “color パッケージ” によって提供されます。そのようなパッケージを利用する際には、プリアンブルで `\usepackage` というコマンドを次の形式で用います。ただし、 $\langle package \rangle$ は使用するパッケージの名称で、 $\langle options \rangle$ は $\langle package \rangle$ パッケージのオプション指定です ($[\langle options \rangle]$ は省略できます)。

```
\usepackage[ $\langle options \rangle$ ]{ $\langle package \rangle$ }
```

複数のパッケージを用いる場合には、

```
\usepackage{amsmath,amssymb}
```

のようにパッケージ名のコンマ区切りリストを `\usepackage` に与えても構いません。その場合にオプション指定 $\langle options \rangle$ も用いていると、指定したオプションは列挙したパッケージのすべてに用いられます。例えば、

```
\usepackage[dvips]{graphicx,color}
```

の場合には、オプション `dvips` は `graphicx` , `color` の両パッケージに共通に用いられます。オプション指定が共通ではないようなパッケージは、別々の `\usepackage` を用いて読み込みます (`\usepackage` は何回用いても構いません)。

注意 2.1

講義用のレジュメや学生のレポートなどの “著者自身が最終的な体裁の面倒を見るような文書” を作成する際には、とりあえず “意図通り” の出力が得られるような記述を行って構いません⁸⁾。一方、一般の出版物の場合には最終的な体裁は著者自身ではなく出版社が決定します。また、組版⁹⁾を行うのも (通常は) 著者自身ではありません。したがって、出版物用の原稿においては、著者自身がいくら念入りに体裁を整えてもそれは無駄な作業でしかありません¹⁰⁾。そこで、出版を前提とした原稿を作成する場合には次のことを心がけてください。

- 原稿の体裁にはこだわらないようにします。

⁸⁾ 本当は、どのような文書を作成するときにも出版用の原稿を作成するときと同じ心構えで作成した方がよいのですが、そうするためには体裁のカスタマイズ方法がある程度学ぶ必要が生じます。

⁹⁾ ここでは、原稿を実際に出版される体裁に整形する処理、と理解して構いません。

¹⁰⁾ 学会等の講演の予稿集等では、著者が出力した原稿をそのまま集めて印刷に回すようなこともあります。しかし、商業的な出版物に用いる原稿をそれと同レベルで考えることはできません。

- (\LaTeX を用いる場合は) 原稿の意味を反映したマークアップ¹¹⁾を行います .

言い換えると, \LaTeX による (出版物用の) 原稿作成の際に重要なことは, 原稿の意味と構造が正しく伝わるように記述することで, 見た目のよい原稿を作成することではありません¹²⁾.

3. \LaTeX 文書の本文の書き方

3.1. ソースファイル中の改行

前節では `\begin{document}` と `\end{document}` との間に本文を書く, と述べました . ここでは, その本文の書き方の基本的なことを説明します .

まず, 本文は (日本語の文章であるか否かによらず) 普通に書いて構いません . ただし, 出力結果での改行¹³⁾位置は, \TeX が (1 行の長さにうまく収まるように) 自動的に決定します . したがって, ソースファイル (§2 以前では文書ファイルと呼んでいました) では, 出力結果での改行位置とは無関係に改行¹⁴⁾して構いません . (決して WYSIWYG¹⁵⁾ではない, というわけです .)

例えば,

```
\documentclass[12pt]{jarticle}
\begin{document}
\LaTeX{}では改行位置は自動的に決定されます。
\end{document}
```

というファイルと

```
\documentclass[12pt]{jarticle}
\begin{document}
\LaTeX{}では
改行位置は
自動的に
決定されます。
\end{document}
```

というファイルを処理すると, どちらも同じ結果

\LaTeX では改行位置は自動的に決定されます。

¹¹⁾後述するように, \LaTeX では種々のコマンドを用いて文書の構造 (例えば, “章” や “節” の見出し) を記述します .

¹²⁾ \LaTeX の場合, 原稿の意味と構造が適切に指定してあれば, 然るべき体裁に整形することは容易です . 逆に, いくらきれいな原稿であっても文書の構造を反映していないマークアップしかなされていない場合には, 組版時に原稿全体にわたって再マークアップしなければならないこともあります .

¹³⁾出力結果における “改行” とは, 文字通り “行が改まること” (一つの行が終わり, 次の行が始まること) です . “新しい段落を始めること” は改行ではなく, “改段落” といいます .

¹⁴⁾テキストファイルにおける “改行” とは, 改行コードを入力すること (あるいは改行コードの位置) を指します . また, テキストファイルにおける “行” とはある改行コード (またはファイルの先頭) からその次の改行コード (またはファイルの終端) までのことです .

¹⁵⁾“What You See Is What You Get” の意です .

が得られます．そこで，ソースファイルにおいては改行位置を気にせずに，読みやすい位置で（文末あるいは文節の切れ目で）改行しましょう¹⁶⁾．

3.2. 改行位置の指定

改行位置は，T_EX が自動的に決定しますが，改行する位置を（ユーザ自身が）指定する場合には，改行する位置に `\\` と記入します．例えば，

```
\documentclass[12pt]{jarticle}
\begin{document}
ここで  \\ 改行します。
\end{document}
```

と入力すると，

```
ここで
改行します。
```

のようになります．

Note: `\\` の直後に，文字 “*”， “[” で始まるテキストを書きたい場合には，それらの文字の直前に `{}` を補ってください．（これは，`\\` の処理の都合によります．）

注意 3.1

改行のコマンド `\\` の直後に “`[<space>]`”（`<space>` は単位付きの長さ）という記述をつけると，改行を行った直後に `<space>` の大きさの垂直方向の空白を入れます．（`<space>` が負の場合には，`<space>` の絶対値だけ行間隔を小さくします．）例えば，

```
patapata\\[-1.5ex]
oyoyo\\
arere\\[1.5ex]
ottotto
```

という入力からは

```
patapata
oyoyo
arere

ottotto
```

という出力が得られます（`1ex` はほぼ “文字 `x` の高さ” です）¹⁷⁾．なお，`tabular` 環境や `eqnarray` 環境（これらについては後述します）での `\\` においても “`[<space>]`”（`<space>` は単位付きの長さ）というオプションによる行間隔の調整を行うことができます．また，`\\` の直後に文字 “*” をつけて用いると，その `\\` による改行位置での改ページを禁止します．

¹⁶⁾書籍の類の原稿を作成する場合には，“1 段落 = 1 行” の形式を要求されることもあります．ただし，T_EX を用いる業者が組版を担当することがわかっている場合には適宜改行を入れた方がよいでしょう．

¹⁷⁾古い版の pL^AT_EX を用いた場合，`[-1.5ex]`，`[1.5ex]` の後の改行文字（に由来する空白）を無視させるために `[-1.5ex]` などの直後に文字 `%` を置く必要がある場合があります．

3.3. 改段落

新しい段落を始めたい場合には，

```
\documentclass[12pt]{jarticle}
\begin{document}
  1 行目 \\  
  2 行目
```

2 番目の段落です。

```
\end{document}
```

のように，新しい段落の前に空白行（改行文字だけの行）を入れます。（空白行の代わりに `\par` というコマンドを用いても構いません。）実際，上の例からは次の出力が得られます。

```
  1 行目  
  2 行目  
  2 番目の段落です。
```

なお，この出力からわかるように，段落の先頭での字下げも自動的に行われるので，ソースファイルでは，段落の先頭に空白を入れる必要はありません。（段落の先頭に和文文字の空白（いわゆる全角空白）を入れると，その和文文字の空白の分も余分に字下げされてしまう¹⁸⁾）ので，注意してください（ASCII character の）空白文字は，行頭に入っても問題はありません）。また，段落の先頭での字下げを行わせないようにするには，段落の先頭に `\noindent` と書きます。例えば，次のようになります。

— INPUT —

```
\documentclass[12pt]{jarticle}
\begin{document}
  1 行目 \\  
  2 行目  
  
  \noindent 2 番目の段落ですが字下げは行われません。  
  \end{document}
```

— OUTPUT —

```
  1 行目  
  2 行目  
  2 番目の段落ですが字下げは行われません。
```

なお，コマンドの後には原則として空白文字を入れてください¹⁹⁾。例えば，“`\LaTeX user`”と書くべきところを，空白を入れずに“`\LaTeXuser`”と書くと，この

¹⁸⁾というよりも，字下げした後に和文文字の空白が書き込まれる，ということなのですが。

¹⁹⁾この“`\LaTeX 入門`”においては“空白文字”というのは ASCII character の空白文字（いわゆる半角空白）のことです。

“\LaTeXuser”の全体が一つのコマンドとして扱われて不都合が生じます。ただし、\\のような、“文字\ + ((記号 1 文字))”という形のコマンドの直後には空白文字を入れる必要はありません。

注意 3.2

“和欧文間スペース”についても（段落の先頭の字下げのスペースと同様に）自動的に挿入されます。例えば，“localな”のように“local”と“な”の間に空白を入れずに記述しても“local な”のように“local”と“な”の間に和欧文間スペースが補われます。

3.4. ソースファイル中の改行文字・空白文字の取り扱い

次に，ソースファイルにおける改行について，少し補足します。この節の最初の例からわかるように，和文文字の直後の改行は全く無視されます。一方，欧文文字の直後の改行は空白文字として扱われます。実際，次のようになります。

— INPUT —

```
\documentclass[12pt]{article}
\begin{document}
The new-line codes following an ASCII character
are considered as space characters.
\end{document}
```

— OUTPUT —

The new-line codes following an ASCII character are considered as space characters.

このことは，欧文のテキストファイルにおいては，単語の途中で改行するのは不自然である（逆に言えば，改行位置は単語どうしの間とみなして構わない）ということに注意すれば，自然なことです。

一方，ソースファイル中の連続した空白文字は，何個連続していてもただ 1 個の空白文字として扱われます。実際，

```
\documentclass[12pt]{article}
\begin{document}
空白文字 1 個 空白文字 2 個 空白文字 3 個 おしまい
\end{document}
```

という入力からは

空白文字 1 個 空白文字 2 個 空白文字 3 個 おしまい

のような出力が得られます。

また，“コメント”すなわち，“文書ファイル中には注釈として書き込みたいが出力結果には表示したくないような記述”を入れたい場合には，文字“%”を用います。コメントになる範囲は，文字“%”からその後で現れる最初の改行文字までです。例えば，

```

\documentclass[12pt]{jarticle}
\begin{document}
%% これは，改行文字の処理についての例です．
\LaTeX{}では
改行位置は
自動的に
決定されます。
\end{document}

```

という内容のファイル进行处理すると，

```
%% これは，改行文字の処理についての例です．
```

という行の全体がコメントとして扱われます．したがって，この例は，この節の最初の例と同じ結果

\LaTeX では改行位置は自動的に決定されます。

を与えます．

以上のことをまとめると，次のようになります．

- (1) ソースファイルの改行位置は出力結果の改行位置とは関係ありません．
- (2) 段落を改めたいときには，空白行を入れます．
- (3) 段落の先頭での字下げは自動的に行われます．字下げを抑制するには `\noindent` コマンドを用います．
- (4) 和文文字の直後の改行は無視されます．欧文文字の直後の改行は空白文字として扱われます．
- (5) 連続する空白文字はただ 1 個の空白文字として扱われます．
- (6) 文字 “%” から，その後で現れる最初の改行文字までの範囲はコメントとして扱われます．

なお，(ASCII character の) “%”，“&” といったいくつかの文字は， \TeX において特別な意味をもちます．これらの文字を出力したいときには，和文文字を用いるか，§5 で説明する特殊文字の出力法を用いてください．

4. 文書クラス

4.1. 主な文書クラス

§3 ですでに述べたように， \LaTeX の文書ファイルは(基本的には)文書クラスの指定

```
\documentclass[\langle options \rangle]{\langle classname \rangle}
```

で始まります．ここで，*\langle options \rangle* は文書クラスのオプションで，*\langle classname \rangle* は文書クラス名です([*\langle option \rangle*] は省略できます)．日常的に用いられる文書クラスには，次のようなものがあります．

- (1) `article`, `jarticle`: 最も一般的に用いられる文書クラスです²⁰⁾. “章”, “節” のような文書の論理構造としては, `\part` (部), `\section` (節), `\subsection` (小節, 項), `\subsubsection` (小小節, 目), `\paragraph` (段落, 細目), `\subparagraph` (小段落, 細細目) が利用できます.
- (2) `report`, `jreport`: これらの文書クラスを用いると, 文書の論理構造として, `\chapter` (章, `\part` と `\section` の中間の構造) も利用できます.
- (3) `book`, `jbook`: これらの文書クラスでは, `report`, `jreport` と同様に `\chapter` を用いることができるのに加え, 両面印刷 (あるいは製本すること) を前提として, 奇数ページと偶数ページのレイアウトを変えてあります.

ここで, 名前が “j” で始まる文書クラスは, 日本語の文書のための文書クラスです. (`\parindent` (段落の先頭での字下げの幅) などの, 文書の体裁に関係する寸法の値が, 欧文用の文書クラスにおける値と異なっています. なお, \LaTeX では日本語の文書用の文書クラス名は “j-” で始まります.) ただし, 日本語の文書に `article`, `report` などの欧文用の文書クラスを用いても (数式中で和文文字を用いていなければ) たいいていの場合エラーにはなりません. 逆に, 日本語を用いていない文書に日本語の文書用の文書クラスを用いてもエラーが生じることはありません. また, \LaTeX 2_ϵ には, `tarticle`, `treport` のような, 文字 “t” で始まる名前をもつ文書クラスがあります. それらは, “縦書きがデフォルト” の文書クラスです.

4.2. 文書クラスの主なオプション

文書クラスのオプションには次のようなものがあります. もちろん, 文書クラスによってはここに述べたもの以外のオプションもあります.

- (1) `10pt`, `11pt`, `12pt`: 本文の文字の大きさの指定
これらのオプションを用いると, 本文の文字サイズがオプションの値になります²¹⁾. `1pt` は約 0.351 mm ですが, 具体的な長さについて考えるよりも, これらのオプションを適当に指定してみて, 読みやすい文字サイズにすればよいでしょう²²⁾. なお, これらのオプションを指定しないときには, “`10pt`” が用いられます.
- (2) `onecolumn`, `twocolumn`: 1 段組, 2 段組の指定
“`onecolumn`” を指定すると, 本文は 1 段組で組版されます. “`twocolumn`” を指定すると 2 段組になります. なお, これらのオプションを用いないときには, “`onecolumn`” が用いられます.
- (3) `oneside`, `twoside`: 偶数ページと奇数ページの体裁を変えるか否かの指定
“`oneside`” を用いると, すべてのページが同じ体裁で出力されます. “`twoside`” を用いると (両面印刷を前提として), 偶数ページと奇数ページが異なる体裁で出力されます. これらのオプションを指定しない場合, `book`, `jbook` クラスでは “`twoside`”

²⁰⁾和文の場合には, 奥村晴彦氏による `jsarticle` クラスを用いるのもよいでしょう.

²¹⁾“`11pt`” を指定した場合には, 正確には `10.95pt` が用いられます.

²²⁾欧文の書籍で比較的一般的な文字サイズは `10pt` です. 和文主体の出版物では文字サイズを `9pt` 程度にすることが多いのですが, “`9pt`” というオプションは (`jarticle` などのクラスファイルでは) 用意されていないので, とりあえず “`10pt`” を指定するとよいでしょう.

が用いられます．その他のほとんどすべての文書クラスでは，“oneside” が用いられます．

- (4) final, draft: “overfull \hbox” が起こった箇所を表示するか否かの指定
 “overfull \hbox” というのは、適切な改行位置がなくて、一つの行が行の右端から右にはみ出した状態のことです²³⁾．オプション “draft” を指定すると、その “overfull \hbox” が起こったところに（目印として）黒い長方形が出力されます．“final” を用いると、その目印は出力されません．これらを指定しない場合には、“final” が用いられます．なお、“draft” を指定した場合には、少しでも “overfull \hbox” が起こると（はみ出した部分の長さが 1 pt 未満であったとしても）目印が出力されます．
- (5) a4paper, b5paper: 印刷用紙のサイズの指定²⁴⁾
 “a4paper”, “b5paper” はそれぞれ、A4 サイズ、B5 サイズの用紙を用いるという指定です．用紙サイズを指定しない場合、日本語の文書用の多くの文書クラスでは “a4paper” が用いられます．一方、欧文用の文書クラスでは、“letterpaper” (letter size (横 8.5 インチ, 縦 11 インチ) の用紙を用いるという指定、日本語用の文書クラスでは使えないことが多いようです) がデフォルトであることが多いようです．

なお、オプションは二つ以上同時に用いることができますが、その場合には、

```
\documentclass[11pt,twocolumn,twoside,a4paper]{article}
```

のようにオプションをコンマで区切って並べます．

文書クラスとそのオプションについては、概ね次のように決めるとよいでしょう．

- (1) 文書クラスは、普通の文書を書く場合には、article または jarticle で充分です．特に大規模な文書を書く場合には、(j)report, (j)book の利用を検討しましょう．
- (2) 文字サイズについては、A4 サイズの紙に印刷する場合には、(1 段組で、版面幅を大きくとる場合) 11 pt または 12 pt にすることを検討しましょう．1 行の長さにもよりますが、10 pt にすると、1 行あたりの文字（または単語）数が多くなって、読みづらくなることがあるようです．
- (3) その他のオプションについては、必要に応じて用いてください．

なお、文書クラスやオプションを変化させたときの出力の様子については、読者自身がいろいろと実験してみることをお勧めします．

5. 特殊文字の出力法

5.1. \TeX の特殊文字の出力

\TeX では、次に示す文字は特別な意味をもつため、そのままでは出力できません²⁵⁾．

²³⁾それ以外の状態もありますが、“ \LaTeX 入門” の段階では考えなくても構いません．

²⁴⁾版面（本文部分が占める領域）の寸法の設定とも連動します．

²⁵⁾不等号 <, > には特別な意味はないのですが、これらの文字を数式ではないところで用いるとそれぞれ “ \langle ”, “ \rangle ” を出力します．

`\`, `{`, `}`, `#`, `%`, `$`, `^`, `_`, `<`, `>`

なお、バックスラッシュ文字 “`\`” は環境によっては円記号 “¥” で表示されます。

これらの文字を出力するには、次のように記述します。(書体をタイプライタ体にして出力しても構わないのであれば、§8 で紹介する `\verb` コマンドを用いることもできます。)

```

\backslash$      → \ , \{ → { , \} → }
\textasciitilde → ~ , $<$ → < , $>$ → >
\textasciicircum → ^ , \_ → _ , \& → &
\#              → # , \% → % , \$ → $

```

5.2. テキスト用記号類

次のような文字も用意されています。

```

\pounds        → £      , \S   → §    , \P → ¶ , \ae → æ , \AE → Æ
\copyright     → ©      , \o   → ø    , \O → Ø , \oe → œ , \OE → Œ
\textregistered → ®      , \l   → l    , \L → L , \aa → å , \AA → Å
\LaTeX         → LATEX   , \TeX → TEX , \i → i , \j → j , \ss → ß
\LaTeXe        → LATEX 2ε

```

ここで、`\TeX`、`\LaTeX`、`\LaTeXe` はただ一つの文字ではなく、`TEX`、`LATEX`、`LATEX 2ε` というロゴを出力しますが、ついでに入れておきました。また、`\i`、`\j` は文字 `i`、`j` にアクセント記号をつけるときに使う“点のない `i`、`j`”を出力するためのコマンドです。

また、ハイフンやダッシュ(短い横線)は文字 `-` を用いて出力できます。ただし、`-` は数式(文字 `$` で挟まれたところなど)の中では単にマイナス記号になります。

```

- → - , -- → - , --- → —
$-$ → - , $--$ → -- , $---$ → ---

```

これらのハイフンとダッシュは、概ね次のように使い分けます。

- ハイフン “`-`”: 複合語の類の区切り (“built-in”, “*n*-dimensional”), 郵便・電話番号などの区切り (“123-4567”)
- en-dash “`-`”: 数値の範囲の類 (“pp. 123–456”)
- em-dash “`—`”: 欧文でのダッシュ (“He stared at her ears — those of a cat!”)

注意 5.1

`TEX` では、英単語などの途中で行分割を行う場合のハイフネーションは自動的に行われます。したがって、原則的としてソースファイルではハイフネーションの指定を行いません。ただし、複合語・英語以外の言語に由来する単語などについて、ハイフネーションが正しく行われていないのを修正する場合には、コマンド `\-` をハイフンを挿入しても構わない位置に入れます。例えば、“Gauss`\-`ian” と記述すると、ハイフネーションの必要がある場合には “Gauss-ian” となり、ハイフネーションの必要がない場合には (`\-` が無視されて) “Gaussian” となります。あるいは

```
\hyphenation{Gauss-ian}
```

のように `\hyphenation` を (プリアンブルで) 用いて “ハイフネーションの例外” を登録するのもよいでしょう。 `\hyphenation` では、単語を “ハイフンを入れてもよい箇所のすべてにハイフンを入れた状態で” 列挙します。(複数の単語を登録する場合は、単語を空白文字で区切ります。)

なお、ハイフンは “必ずハイフンが入る” 箇所に限り用います。行分割位置に関するハイフネーションの調整の場合にはハイフンが入るとは限らない²⁶⁾ので、ハイフンを直接書き込むのではなく `\-` コマンドを用います。

欧文用の二重引用符は、(本文用の欧文フォントが用いられている箇所では) 左二重引用符に対しては ‘ ‘ , 右二重引用符に対しては ’ ’ のように単引用符を 2 個並べて書けば出力できます。

5.3. テキストでも用いられることが多い数式記号

次の表は \$ で挟んで用いるコマンド²⁷⁾で出力される文字の表です。

<code>\$\heartsuit\$</code>	→ ♡	<code>\$\dagger\$</code>	→ †	<code>\$\flat\$</code>	→ ♭
<code>\$\spadesuit\$</code>	→ ♠	<code>\$\ddagger\$</code>	→ ‡	<code>\$\natural\$</code>	→ ♮
<code>\$\diamondsuit\$</code>	→ ♦	<code>\$\diamond\$</code>	→ ◇	<code>\$\sharp\$</code>	→ ‡
<code>\$\clubsuit\$</code>	→ ♣	<code>\$\triangle\$</code>	→ △	<code>\$\star\$</code>	→ ★
<code>\$\circ\$</code>	→ ○	<code>\$\bullet\$</code>	→ ●	<code>\$\ast\$</code>	→ *
<code>\$\smile\$</code>	→ ☺	<code>\$\frown\$</code>	→ ☹		

これら以外にも多くの文字が用意されていますが、それらの多くは数式以外で用いられることはあまりないものなので、数式用の記号について説明する §27 で紹介します。

5.4. テキスト用アクセント記号

ここでは、テキスト用のアクセント記号を挙げます。

<code>\`o</code>	→ ò	<code>\'o</code>	→ ó	<code>\~o</code>	→ ô	<code>\~o</code>	→ õ
<code>\"o</code>	→ ö	<code>\H o</code>	→ ő	<code>\v o</code>	→ ǒ	<code>\u o</code>	→ ǔ
<code>\.o</code>	→ ȧ	<code>\d o</code>	→ ȡ	<code>\t{oo}</code>	→ Ōō	<code>\c o</code>	→ Ȯȯ
<code>\b o</code>	→ Ȣȣ	<code>\=o</code>	→ Ȫȫ	<code>\r o</code>	→ Ȥȥ		

このようなアクセント付きの文字を数式中で用いたいときには、数式用のアクセントをつけるコマンドを用います。それらについても、§27 で扱うことにします。

5.5. 丸や枠で囲んだ文字

本節の最後に、“丸で囲んだ文字” を出力するためのコマンド `\textcircled` と、テキストを枠で囲むためのコマンド `\fbox` を紹介します。まず、`\textcircled` は

```
\textcircled{\textit{letter}}
```

²⁶⁾原稿の修正などにより行分割位置が変化した場合にはハイフネーションの必要がなくなることもある、という点に注意してください。

²⁷⁾正しくは、“数式用の記号として定義されている” コマンドです。

($\langle letter \rangle$ は丸で囲む文字(列)) のように用います．例えば，

```
\textcircled{\small 1}--\textcircled{\small 9},
\textcircled{\scriptsize 10}--\textcircled{\scriptsize 99},
\textcircled{\tiny 100}--\textcircled{\tiny 999}
```

という入力からは，

①–⑨, ⑩–⑨⑨, ⑩⑩–⑨⑨⑨

という出力が得られます．ここで用いた，`\small`，`\scriptsize`，`\tiny` は，文字サイズを変更するコマンドです (§6 を参照してください)．

一方，`\fbox` は

```
\fbox{\langle text \rangle}
```

($\langle text \rangle$ は枠で囲む文字列) のように用います．また，枠の線の太さは `\fboxrule` という寸法で指定され，枠の中身と枠との間隔は `\fboxsep` という寸法で指定されます．(`\fboxrule`，`\fboxsep` のデフォルトの値は個々のクラスファイルの中で設定されますが，一般的には `\fboxrule` は 0.4pt，`\fboxsep` は 3pt となっています．)

例えば，

```
\fbox{patapata}
{\setlength{\fboxsep}{0pt}\fbox{patapata}}
{\setlength{\fboxrule}{1pt}\fbox{patapata}}
```

という記述からは

patapata patapata patapata

という出力が得られます．

Note: (一部の環境で和文文字として用意されている) 丸で囲んだ数字やローマ数字などの環境依存文字は，(\TeX のソースファイルに限らず) いかなるテキストファイルにおいても用いるべきではありません²⁸⁾．環境依存文字を使いたくなったときには，

- (1) 他の表現で代用できないか (例えば，“丸で囲んだ数字” は多くの場合，“(1)”，“(2)”のような記述に代えても意味は通じます)
- (2) \TeX のコマンドを使用して出力できないか (例えば，上記の `\textcircled` を使用した記述を用います)

といったことを考えてください．なお，ローマ数字は “I”，“II”，“III” または “i”，“ii”，“iii” のように単にアルファベットを並べて書けばそれで充分です²⁹⁾．

²⁸⁾まあ，“Unicode 化” が充分に進んで丸数字の類も “普通の” 文字になると話は別ですが，テキストファイルで用いられる文字コードの大半が Unicode となる，という日はすぐには来ないでしょう．

²⁹⁾単にアルファベットを並べて表記するのが本来の記法です．

6. 書体・文字サイズの変更

6.1. 文字サイズの変更

ここでは、文字の書体やサイズの変更について扱います。まず、サイズの変更は、次のコマンドで行います。

```
\tiny ≤ \scriptsize ≤ \footnotesize ≤ \small ≤ \normalsize
≤ \large ≤ \Large ≤ \LARGE ≤ \huge ≤ \Huge
```

ここで “\tiny ≤ \scriptsize” というのは “\tiny によって指定されるサイズは \scriptsize によって指定されるサイズ以下である” ということを表します（他のコマンドの関係についても同様です）。ただし、(\documentclass のオプションでの) 文字サイズの指定によっては、\huge サイズと \Huge サイズが同じになる、といったこともあります。次の例を参照してください³⁰⁾。

<code>{\tiny tiny}</code>	→ tiny
<code>{\scriptsize scriptsize}</code>	→ scriptsize
<code>{\footnotesize footnotesize}</code>	→ footnotesize
<code>{\small small}</code>	→ small
<code>{\normalsize normalsize}</code>	→ normalsize
<code>{\large large}</code>	→ large
<code>{\Large Large}</code>	→ Large
<code>{\LARGE LARGE}</code>	→ LARGE
<code>{\huge huge}</code>	→ huge
<code>{\Huge Huge}</code>	→ Huge

この例のように、サイズを変更する範囲は “{” と “}” で囲んでください。

6.2. 書体の変更

数式以外のところでの書体の変更は次のコマンドで行います。

- `\rmfamily`, `\textrm`: 欧文フォントをローマン体（最も普通の欧文書体）にします。
 - `\sffamily`, `\textsf`: 欧文フォントをサンセリフ体にします。
 - `\ttfamily`, `\texttt`: 欧文フォントをタイプライタ体にします。
 - `\mcfamily`, `\textmc`: 和文フォントを明朝体にします。
 - `\gtfamily`, `\textgt`: 和文フォントをゴシック体にします。
-
- `\mdseries`, `\textmd`: 普通の太さの文字にします。
 - `\bfseries`, `\textbf`: ボールド体（太字）にします。
-

³⁰⁾もちろん、これは本稿の設定での出力例です。使用するクラスファイルおよびオプション指定により実際に用いられる文字サイズは変わります。

- `\upshape`, `\textup`: 直立体 (傾いていない, 普通の書体) にします.
- `\itshape`, `\textit`: イタリアック体にします.
- `\scshape`, `\textsc`: スモールキャピタル³¹⁾にします.
- `\slshape`, `\textsl`: スラント体³²⁾にします.

これらのコマンドのうち, 左側に書いた `\rmfamily` などは, サイズ変更コマンドと同様に,

```
{\rmfamily 書体を変更する範囲}
```

という形式で用います. 一方, 右側の `\textrm` などは

```
\textrm{書体を変更する範囲}
```

という形式で用います. 実際, 次のようになります³³⁾.

```
{\rmfamily rmfamily}, \textrm{rmfamily} → rmfamily
{\sffamily sffamily}, \textsf{sffamily} → sffamily
{\ttfamily ttfamily}, \texttt{ttfamily} → ttfamily
{\mdseries mdseries}, \textmd{mdseries} → mdseries
{\bfseries bfseries}, \textbf{bfseries} → bfseries
{\upshape upshape}, \textup{upshape} → upshape
{\itshape itshape}, \textit{itshape} → itshape
{\scshape scshape}, \textsc{scshape} → SCSHAPE
{\slshape slshape}, \textsl{slshape} → slshape
```

また, 上記のコマンドは横線によって三つのグループに分かれていますが, 異なるグループに属するコマンドは同時に使用できます. (同じグループに属するものどうしは, 最後に用いたものが有効です.) 例えば, `\bfseries` と `\itshape` を同時に指定してボールド・イタリアック体を使うこともできます.

— INPUT —

```
{\bfseries boldface}, {\itshape italic}\
{\bfseries\itshape bold italic}
```

— OUTPUT —

```
boldface, italic
bold italic
```

³¹⁾小文字の代わりにサイズを小さくした大文字を使う書体です.

³²⁾直立体の文字をただ傾けただけの書体です.

³³⁾和文フォントについては例示するまでもないでしょう. 明朝体は本稿の本文で用いられていますし, ゴシック体は見出し類で用いられています.

ここでは、`\bfseries` の方を先に書きましたが、もちろん `\itshape` の方を先に書いても構いません。なお、上記の出力を得るのに

```
\textbf{boldface}, \textit{italic}\\
\textbf{\textit{bold italic}}
```

のように記述することもできます。また、書体変更コマンドは、最初に述べたサイズ変更コマンドと同時に用いても構いません。ただし、一般的でない書体（例えば、サンセリフのスモールキャピタル）を用いる指定をした場合、エラーにはなりませんが、指定したものと異なる書体で代用されることがあります。

実際、“`\textsf{\textsc{aaa}}`” のような記述を行うと、文字列 “aaa” の部分は単に（ローマンの）スモールキャピタルで出力され、

```
LaTeX Font Warning: Font shape 'OT1/cmss/m/sc' in size <10> not available
(Font) Font shape 'OT1/cmr/m/sc' tried instead on input line 7.
```

のような警告が表示されます。この警告の意味は、次のとおりです。

OT1 エンコーディング³⁴⁾で、ファミリー³⁵⁾、シリーズ³⁶⁾、シェイプ³⁷⁾がそれぞれ `cmss` (computer modern sans-serif), `m` (medium), `sc` (small caps) である書体は文字サイズ 10pt では利用できないので、その代わりに 7 行目において OT1 エンコーディングで、ファミリー、シリーズ、シェイプがそれぞれ `cmr` (computer modern roman), `m` (medium), `sc` (small caps) の書体を用いることを試みました。

つまり、単に“使えない書体があったので代用した”と言っているにすぎません。この警告に現れた “OT1/cmss/m/sc” といった文字列の意味はわからなくても構いませんが、メッセージを読めば大意をとることができることでしょう³⁸⁾。また、和文フォントについてはデフォルトでは明朝体とゴシック体のみを用いるようになっているため、単にイタリック体を用いただけでも和文フォントに関する（上記の警告に似た）警告が生じます³⁹⁾。いずれにせよ、警告 (warning) が生じた場合には警告メッセージを落ち着いて読んでください。

なお、 \LaTeX 2.09⁴⁰⁾との互換性のために用意されている書体変更コマンドには次のようなものがあります。

```
\rm = \normalfont\rmfamily
\sff = \normalfont\sffamily
```

³⁴⁾フォントのエンコーディングとは、“文字コード”と“文字の字形”との対応関係のことです。

³⁵⁾フォントのファミリーとは、フォントのデザイン上の系統のことです。

³⁶⁾フォントのシリーズとは、個々の文字の幅や線の太さに着目した分類です。

³⁷⁾フォントのシェイプとは、個々の文字の傾き具合など、シリーズでは扱わない特徴に関する分類です。

³⁸⁾ただ、警告メッセージはきちんとした文になっていないものが多いので、その分読みにくいかもしれません。

³⁹⁾そもそも、和文フォントには“イタリック体”という概念はありません。(斜体、すなわち“スラント体”はときどき見かけますが。)

⁴⁰⁾古い版の \LaTeX です。現在ではサポートされていないので、新しく作成する文書では \LaTeX 2.09 形式の記述は行わない方がよいでしょう。


```

\tt = \normalfont\ttfamily
\bf = \normalfont\bfseries
\it = \normalfont\itshape
\sc = \normalfont\scshape
\sl = \normalfont\slshape

```

ここで、等号の右側の記述は、等号の左側のコマンドを数式ではないところで用いたときの意味です。また、`\normalfont` は、書体をデフォルトの書体 (普通は `\rmfamily` の `\mdseries` の `\upshape`) に戻すという働きのコマンドです。したがって、これらの \LaTeX 2.09 互換の書体変更コマンドを組み合わせ用いた場合、最後に用いたコマンドだけが有効になります。(まず `\normalfont` によって書体変更をリセットしてからさらに変更していることに注意してください。)

数式での書体の変更は `\mathrm` のような、`\text...` の `text` の部分を `math` に変えた名前のコマンドを用いて行いますが、数式専用の書体もあるので詳しいことは §28 で説明します。

7. 文書の論理構造 (章・節など)

7.1. 文書の構造を指定するコマンド

ある程度長い文章には“章”・“節”といった構造があります。 \LaTeX では、“章や節の見出しを、その見出しを用いる位置に適当な (見出し用の) 書体・サイズで直接書き込む”ようなことはしません。その代わり、“ここから新しい章が始まる”といった文書の構造に関する指定をソースファイルに書き込み、見出しの出力形式については“見出しを出力するコマンド”に任せる、という方法をとります。

この \LaTeX 流の方法の長所・短所のいくつかを挙げると次のようになります。まず、長所としては、

- 文書の構造がわかりやすい: “節”の見出しを出力するコマンドを検索させることで、文書の編集時に“節”単位の移動を行う、といったことも容易です⁴¹⁾。
- “章”、“節”などの番号付けを気にする必要がない: \LaTeX は自動的に番号をつけますし、文書の変更などで番号が変わるようなときには自動的に正しい番号に変更します。

といった点が挙げられます。一方、短所としては、

- 見出しの出力形式を変更しにくい: 見出しの出力形式を変更しようとする場合には、対応するコマンドの定義を変更する必要があります。

といった点があります。ただし、この短所は、“見出しの出力形式を覚えていなくても、同じコマンドが出力する見出しは必ず同じ形式で出力される”ということと表裏一体です。また、適当な手段を用いれば、見出しの出力形式のカスタマイズはある程度可能です。

⁴¹⁾ワープロソフトの類にもいわゆる“アウトラインプロセッサ”の機能があることも多いようですが、目次の作成や相互参照の問題を考えるとやはり \LaTeX 流の方法に軍配が上がるようです。

さて、 \LaTeX の“流儀”について述べたので、次に文書の構造を指定するコマンドを挙げます。 \LaTeX では、次のコマンドを用意しています。

```
\part
\chapter (一部の文書クラス (book など) でのみ使用可)
\section
\subsection
\subsubsection
\paragraph
\subparagraph
```

ここで、 $\backslash\text{paragraph}$ 、 $\backslash\text{subparagraph}$ に対しては、番号が出力されないことが普通です（その場合でも番号は数えられています）。一方、番号をつけない $\backslash\text{section}$ などには、 $\backslash\text{section*}$ のように “*” をつけてください。なお、見出しとして用いる文字列は、それらのコマンドの直後に {, } に入れて書いてください。例えば、本節 (§7) の見出しは

```
\section{文書の論理構造 (章・節など)}
```

という記述によって出力しています。また、本項 (7.1 項) の見出しは

```
\subsection{文書の構造を指定するコマンド}
```

という記述を用いて出力しています。

これらの例からわかるように、例えば、 $\backslash\text{subsection}$ のところでは番号は “7.1” のように $\backslash\text{section}$ の番号を添えた形で出力されています。（番号 n の $\backslash\text{section}$ の m 番目の $\backslash\text{subsection}$ では、番号は $n.m$ と出力されるわけです。）同様に、 $\backslash\text{subsubsection}$ に対しては $\backslash\text{subsection}$ の番号が添えられます。また、本稿では番号 “5.4” の $\backslash\text{subsection}$ の次の $\backslash\text{subsection}$ には “6.1” と番号がついていますが、このように $\backslash\text{section}$ が変わると、 $\backslash\text{subsection}$ の番号は 1 から始まるようにリセットされます。同様に、先程のリストにおいて、1 行上にあるコマンドが現れると、自分自身に対する番号はリセットされます（これは、自然な番号の付け方です）。ただし、 $\backslash\text{part}$ コマンドは例外で、このコマンドは $\backslash\text{chapter}$ の番号をリセットしませんし、 $\backslash\text{chapter}$ などの番号にも $\backslash\text{part}$ の番号は添えられません。

注意 7.1

$\backslash\text{section}$ などの文書の構造を表すコマンドを用いずに

```
{\bfseries 1 Preliminaries}
```

```
{\bfseries 1.1 Definitions and Fundamental Results}
```

のような記述をする人がいますが、これでは “第 1 節および 1.1 項を開始している” のか、“単にボールド体で表記した文字列を並べている” のか、の区別⁴²⁾は書いた本人以外にはできません。“第 1 節および 1.1 項を開始している” のであれば、そのことを意味する記述

⁴²⁾実際にはこれらのどちらでもなく、“第 1 章および 1.1 節の開始” かもしれません。

```
\section{Preliminaries}
\subsection{Definitions and Fundamental Results}
```

を (然るべき位置で⁴³⁾) 用いて, 個々の記述の意味と構造を明示します.

7.2. 見出しの与え方に関する補足

`\section` などの見出しにする文字列の中にコマンドを入れるときには, そのコマンドの直前に `\protect` をつける必要がある場合があります. また, 見出しが長すぎる場合や, 見出しに脚注が含まれている場合には, “目次用の見出し” を用意した方がよいでしょう. ここで, “目次用の見出し” は,

```
\section[ $\langle$ heading-for-toc $\rangle$ ]{ $\langle$ heading $\rangle$ }
```

(\langle heading-for-toc \rangle は目次用の見出し, \langle heading \rangle は本文用の見出し) というように, `\section` などの直後で “[”, “]” で囲んで与えます. (目次の作成については §16 を参照してください.)

ところで, 長い文章の一部を書いているときなどには, `\section` などの番号をユーザ自身で指定したい場合があります. 例えば, `\section` の番号を N から始めたい場合には

```
\setcounter{section}{ $N - 1$ }
```

という記述 (“ $N - 1$ ” についてはユーザ自身で計算して計算後の数値を書き込みます) を, 番号 N にする `\section` の直前に書き込んでください. この記述は, 現在の `\section` の番号を $N - 1$ にする, という意味です. (したがって, その次の `\section` の番号は意図通りに N になります.) 同様に, `\setcounter{subsection}{...}` などによって, `\subsection` などの番号も変更できます.

注意 7.2

`\section` などの見出しの書式を変更するには, `\section` などの定義を変更する必要があります. 適当なクラスファイル (例えば, `jarticle.cls`) を読んでみると,

```
\newcommand{\section}{\@startsection{section}{1}{\z@}%
  {1.5\Cvs \@plus.5\Cdp \@minus.2\Cdp}%
  {.5\Cvs \@plus.3\Cdp}%
  {\reset@font\Large\bfseries}}
\newcommand{\subsection}{\@startsection{subsection}{2}{\z@}%
  {1.5\Cvs \@plus.5\Cdp \@minus.2\Cdp}%
  {.5\Cvs \@plus.3\Cdp}%
  {\reset@font\large\bfseries}}
```

のような定義が見つかります (`\Cvs`, `\Cdp` は寸法です). ここで, これらの定義の中の `\@startsection` コマンドの最後のパラメータ (例えば, `\section` の定義では, `\normalfont\Large\bfseries` のところ) が, 見出しの書体の指定です. した

⁴³⁾これらの `\section`, `\subsection` の番号が “1”, “1.1” となるような場所で, ということです.

がって、例えば、プリアンブルにおいて

```
\makeatletter
\def\section{\@startsection{section}{1}{\z@}%
  {1.5\Cvs \@plus.5\Cdp \@minus.2\Cdp}%
  {.5\Cvs \@plus.3\Cdp}%
  {\normalfont\large\bfseries}}
\makeatother
```

のように `\section` コマンドを再定義すると、`\section` の見出しが `\large` サイズのボールド体 (`\bfseries`) で出力されるようになります。(ここで現れた `\bfseries` などの書体変更コマンドについては §6 を参照してください。)

なお、`\@startsection` の書式とパラメータの意味は次のとおりです。(詳しいことについては $\text{T}_{\text{E}}\text{X}/\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ でのマクロ作成について扱った文献を参照してください。)

```
\@startsection{<name>}{<level>}{<indent>}
  {<beforeskip>}{<afterskip>}{<style>}
```

- `<name>`: 見出し項目の名称です。見出しの番号付けに用いられるカウンタの名称としても用いられます。
- `<level>`: 見出し項目の“レベル”です。レベルの値がカウンタ `secnumdepth` の値を超えると、見出しには番号がつかなくなります。(カウンタ `secnumdepth` の値はクラスファイルの中などで設定されます。)
- `<indent>`: 見出し項目の字下げ量です。
- `<beforeskip>`: 見出しの前に追加される空白の大きさは、`<beforeskip>` の値の絶対値になります。`<beforeskip>` の値が負である場合には、見出しの直後の段落の先頭の字下げが抑制されます。
- `<afterskip>`: `<afterskip>` の値が正である場合には、見出しの下側に `<afterskip>` の大きさの空白が追加されます。`<afterskip>` の値が負または 0 である場合には、見出しの後に後続の文章を追い込み、見出しと後続の文章の間に `-\<afterskip>` の大きさの空白を入れます。
- `<style>`: 見出し部分の文字サイズや書体の指定です。

また、見出しの番号の書式を変更するには、`\@secntformat` というコマンドの定義を変更します。これは、

```
\def\@secntformat#1{\csname the#1\endcsname\quad}
```

のように定義されていて、`\csname the#1\endcsname` の部分が `\section` などの番号になります。したがって、プリアンブルで、

```
\makeatletter
\def\@secntformat#1{\csname the#1\endcsname.\quad}
\makeatother
```

のように `\@secntformat` を再定義すると、`\section` の番号が、“1.”、“2.” のよう

にピリオドがついた形になります。(ただし、`\subsection` の番号も “1.1.”, “1.2.” のようになります。(`\subsubsection` 以下についても同様です。))

注意 7.3

`\section`, `\subsection`, `\subsubsection`, `\paragraph`, `\subparagraph` の番号はそれぞれ, `section`, `subsection`, `subsubsection`, `paragraph`, `suparagraph` というカウンタ (ここでは, カウンタの名称については \LaTeX での呼称を用いています) を用いて数えられます。したがって, それらの値を出力するコマンド `\thesection`, `\thesubsection` などの定義を変更すれば, `\section` などの番号の形式を変更できます。例えば,

```
\def\thesection{\Roman{section}}
```

という記述をプリアンプルに入れると, `\section` の番号が大文字のローマ数字で出力されるようになります。なお, カウンタの出力形式は次のようなコマンドで指定できます。

- `\arabic`: 普通に算用数字を用いて 10 進表記します。
- `\roman`: 小文字のローマ数字で出力します。
- `\Roman`: 大文字のローマ数字で出力します。
- `\alph`: “a”, “b”, “c” のように小文字のアルファベットを用います。
- `\Alph`: “A”, “B”, “C” のように大文字のアルファベットを用います。

これらは, `\arabic{subsection}` のように, 出力させるカウンタの名称を与えて用います。

一方, `\subsection` の番号の形式を, “1-1”, “1-2” のように “`\section` の番号と `\section` 内での `\subsection` の通し番号をハイフンで結んだ形式” にするには, `\thesubsection` を

```
\def\thesubsection{\thesection-\arabic{subsection}}
```

のように定義します。

8. 文書の視覚的構成 (右寄せ・左寄せなど)

8.1. テキストの右寄せ・左寄せ

前節では文書の構造を指定するためのコマンド (の一部) を紹介しましたが, ここでは, 文章を右寄せにしたり, 入力ファイルに書いたとおりに出力する方法を説明します。まず, 右寄せなどを行うための記述を挙げます。

- `\begin{flushright} <text> \end{flushright}`:
`<text>` の部分の文章を右寄せにして出力します。
- `\begin{flushleft} <text> \end{flushleft}`:
`<text>` の部分の文章を左寄せにして出力します。
- `\begin{center} <text> \end{center}`:

`<text>` の部分の文章を中央寄せにして出力します。

なお、ここで現れた `\begin{...}` と `\end{...}` で囲まれた部分を“環境”といいます。また、特に環境の種類を明記したいときには `\begin{...}` の... のところの文字列を用いて“... 環境”のようにいいます。例えば、`\begin{flushright}` から `\end{flushright}` までの部分は“flushright 環境”といいます（他の環境についても同様です）。

上記の環境を用いて、

```
\begin{flushright}
  2002 年 3 月 6 日
\end{flushright}
\begin{center}
  例会のお知らせ
\end{center}
\begin{flushleft}
  今年度最後の例会を次の日時・場所にて執り行います。\\
  \textbf{日時} : 3 月 24 日 (日) \\
  \textbf{会場} : 談話室 (A123 室)
\end{flushleft}
```

という入力を与えると、

2002 年 3 月 6 日

例会のお知らせ

今年度最後の例会を次の日時・場所にて執り行います。

日時：3 月 24 日（日）

会場：談話室（A123 室）

のような出力が得られます。ここで、flushleft 環境は何も特別な効果がないように見えますが、次の例のように flushright 環境の中で flushleft 環境を用いてみると、flushleft 環境はテキストを左寄せにしていることがわかります。

— INPUT —

```
\begin{flushright}
  右寄せです
\begin{flushleft}
  左寄せです
\end{flushleft}
  再び右寄せです
\end{flushright}
```

— OUTPUT —

右寄せです

左寄せです

再び右寄せです

なお、今の例は“環境の中で環境を用いても構わない”ということも示しています。

8.2. テキストを書いてあるとおりに出力する方法

テキストをソースファイル中の記述のとおりに出力するには、`verbatim` 環境が使えます。例えば、

```
\begin{verbatim}
#include<stdio.h>

int main(int argc, char *argv[]){
    printf("Hello, World!\n");
    return 0;
}
\end{verbatim}
```

のような入力（空白はすべて空白文字で、タブ文字は用いていないものとします）に対しては、

```
#include<stdio.h>

int main(int argc, char *argv[]){
    printf("Hello, World!\n");
    return 0;
}
```

という出力が得られます。また、`verbatim*` 環境は `verbatim` 環境と同様ですが、空白文字を「`␣`」という記号に変えて出力します。

— INPUT —

```
\begin{verbatim*}
#include<stdio.h>

int main(int argc, char *argv[]){
    printf("Hello, World!\n");
    return 0;
}
\end{verbatim*}
```

— OUTPUT —

```
#include<stdio.h>

int main(int argc, char*argv[]){
    printf("Hello, World!\n");
    return 0;
}
```

`verbatim(*)` 環境は、今の例のように複数行のテキストをそのまま出力するのに使いますが、ほんの数文字（例えば、`(^o^)` のようなもの）を文字通りに出力するには `\verb` コマンドが使えます。これは、出力したい文字列を、その文字列に含まれないような別の文字で挟んで、

```
\verb/(^o^)/
```

のように使います（`\verb|^(^o^)|`、`\verb+(^o^)+` のようにしても構いません）。実際、この入力では“`(^o^)`”という出力を与えます。なお、“`\verb*/.../`”のように*をつけて用いると、空白文字を記号 `_` に変えて出力します。

Note: `\verb` コマンドと `verbatim` 環境は、他のコマンドのパラメータの中では使えません。特に、`\section` などの見出しの中または脚注の中では使えない、ということに注意してください（脚注は §14 で扱います）。

8.3. “引用” 風の環境

また、短い文章を字下げして記述し、“引用” しているかのように出力するには、次の例のように `quote` 環境を用いることができます。

— INPUT —

```
少し文章を引用してみましょう。
\begin{quote}
ここでは、\LaTeX を用いて文書を作成・印刷（または
画面表示）する一連の操作の概要を説明します。
\end{quote}
これは §1 の冒頭です。
```

— OUTPUT —

少し文章を引用してみましょう。

ここでは、`\LaTeX` を用いて文書を作成・印刷（または画面表示）する一連の操作の概要を説明します。

これは §1 の冒頭です。

複数段落にわたるような長い“引用” 箇所には、`quotation` 環境を用います。`quotation` 環境の使用法は `quote` 環境と同様（環境名を `quotation` に変えるだけ）です。

8.4. 幅を指定した段落

この節の最後に，“幅を指定した段落”を作成するコマンド `\parbox` を紹介します．これは，

```
\parbox{<width>}{<text>}
```

（`<width>` は段落の幅，`<text>` は段落のテキスト）という書式で用います．例えば，

```
\parbox{15zw}{%
  幅 15zw のテキストを作ります．
  ここで用いている \texttt{\symbol{'134}\parbox}の中では，
  \texttt{\symbol{'134}\verb}コマンドは使えません．}
```

という入力からは，次のような出力が得られます．（ただし，（タイプライタ体を適用している箇所での）`\symbol{'134}` というのはバックスラッシュ文字のことです．）

```
幅 15zw のテキストを作ります．
ここで用いている \parbox の中で
は，\verb コマンドは使えません．
```

また，`\parbox[b]{15zw}` のようにオプションをつけると“`\parbox` で作成したテキストの，`\parbox` の外部の行に対する位置関係”を指定することができます．オプションと位置関係との対応は大まかに言えば次の通りです（デフォルトは“c”です．）

- b: `\parbox` 中の段落の最後の行を外部の行に揃えます．
- c: `\parbox` の（縦方向の）中央を外部の行に揃えます．
- t: `\parbox` 中の段落の最初の行を外部の行に揃えます．

ここで，`\parbox` にオプションをつけた例を挙げます．

— INPUT —

```
これと
\parbox[b]{5zw}{幅を決めた段落（小さい箱）の例です．}%
これと
\parbox[c]{5zw}{幅を決めた段落（小さい箱）の例です．}%
次の箱は
\parbox[t]{5zw}{幅を決めた段落（小さい箱）の例です．}%
\verb/\parbox/の例です．
```

— OUTPUT —

幅を決めた		幅を決めた	
段落（小さい箱）の例		段落（小さい箱）の例	
これとです．	これと	です．	次の箱は幅を決めた\parbox の例です．
			段落（小さい箱）の例
			です．

この `\parbox` には、その中で `\verb` コマンドや `verbatim` 環境を用いることはできない、という制約があります。幅を決めた段落の中で `\verb` コマンドや `verbatim` 環境を用いたい場合には、

```
\parbox{<width>}{<text>}
```

のように記述する代わりに

```
\begin{minipage}{<width>}
<text>
\end{minipage}
```

と記述することができます。また、`minipage` 環境にも `\begin{minipage}[b]{10cm}` のようにオプションをつけることができます。オプションの意味は `\parbox` の場合と同じです。なお、`minipage` 環境の中では、その `minipage` 環境の中での“脚注”をつけるといったこともできますが、ここでは深入りしません。

9. 箇条書き

9.1. 番号のない箇条書き

ここでは、箇条書きを行う環境について説明します。まず、

(番号のない) 箇条書きの例です。

- 箇条書きは `itemize`、`enumerate` という環境でできます。
- ここでは `itemize` 環境を用いています。

のように、箇条書きの各項目に印をつける箇条書きは `itemize` 環境で出力できます。実際、上記の出力は次のような入力で得られたものです。

(番号のない) 箇条書きの例です。

```
\begin{itemize}
\item 箇条書きは itemize, enumerate という環境でできます。
\item ここでは itemize 環境を用いています。
\end{itemize}
```

この入力テキストからわかるように、箇条書きの各項目の前には `\item` をつけます。また、次の例のように箇条書きの中でさらに箇条書きを行うことや見出しを一時的に変更することもできます。

— INPUT —

```
\begin{itemize}%%% レベル 1
\item 箇条書きの中で箇条書きを使ってみます。
\item 箇条書きの中の箇条書きでは見出しの記号が変わります。
\begin{itemize}%%% レベル 2
\item ここでは見出しは‘‘\textbf{--}’’ですね。
\item[ ] 見出しを に変えてみました。
```

```

\begin{itemize}%%% レベル 3
  \item さらに内側の itemize 環境です .
  \item 字下げの幅が変わっていることにも注意してください .
  \begin{itemize}%%% レベル 4
    \item ここまで itemize 環境を入れ子にできます .
  \end{itemize}
\end{itemize}
\end{itemize}
\item ここは一番外側の itemize 環境です .
\begin{itemize}%%% レベル 2
  \item 再び見出しは‘‘\textbf{--}’’ になりました .
\end{itemize}
\end{itemize}

```

— OUTPUT —

- 箇条書きの中で箇条書きを使ってみます .
- 箇条書きの中の箇条書きでは見出しの記号が変わります .
 - ここでは見出しは “—” です .
 - 見出しを — に変えてみました .
 - * さらに内側の itemize 環境です .
 - * 字下げの幅が変わっていることにも注意してください .
 - ・ ここまで itemize 環境を入れ子にできます .
- ここは一番外側の itemize 環境です .
 - 再び見出しは “—” になりました .

ここで、各 itemize 環境について、その itemize 環境の始まりの `\begin{itemize}` 以前にある `\begin{itemize}` と `\end{itemize}` の個数を数えて、

$$((\text{\code{\begin{itemize}} の個数})) - ((\text{\code{\end{itemize}} の個数}))$$

をその itemize 環境の “レベル” ということにします。(上記の入力例においては、各 `\begin{itemize}` のところにその itemize 環境のレベルをコメントとして書いています。)すると、レベル 1, 2, 3, 4 の itemize 環境の見出しはそれぞれ、

●, —, *, ·

のようになっています⁴⁴⁾。また、itemize 環境のレベルは 4 が最大です。(レベルが 5 以上の itemize 環境を作ろうとするとエラーになります。)

なお、見出しを一時的に変えるには、

```
\item[⟨label⟩]
```

(⟨label⟩ は一時的に用いる見出し)のようになります。ここで、`\item` の後の “[” と “]” は、このような一時的な見出しの指定を表すので、箇条書きの項目として文字 “[” で

⁴⁴⁾ただし、これらは用いている文書クラスに依存します。

始まるような文章を書くときには文字 “[” の直前に “{” を補ってください。また、一時的に用いる見出しに文字 “]” が含まれる場合には、

```
\item[{{1}}]
```

(これは “[1]” という見出しをつける場合です) のように見出し部分を “{”, “}” で囲んでください (L^AT_EX が誤解しないようにするわけです)。

9.2. 番号付き箇条書き

一方、番号付きの箇条書きは `enumerate` 環境を用いて出力されます。実際、次の例のようになります。

— INPUT —

```
\begin{enumerate}
\item enumerate 環境です .
\item enumerate 環境も入れ子にすることができます .
\begin{enumerate}
\item 見出しは ‘(a)’ になりましたね .
\item[({a'$})] 見出しを変更してみました .
\begin{enumerate}
\item レベル 3 の enumerate 環境です .
\begin{enumerate}
\item やはりレベル 4 が限度です .
\end{enumerate}
\end{enumerate}
\end{enumerate}
\end{enumerate}
```

— OUTPUT —

- (1) enumerate 環境です .
- (2) enumerate 環境も入れ子にすることができます .
 - (a) 見出しは “(a)” になりましたね .
 - (a') 見出しを変更してみました .
 - i. レベル 3 の enumerate 環境です .
 - A. やはりレベル 4 が限度です .

ここで、`enumerate` 環境についてもレベルを `itemize` 環境と同様に (ただし、今度は `\begin{enumerate}` と `\end{enumerate}` の個数を用いて) 定義します。すると、レベル 1, 2, 3, 4 の `enumerate` 環境の見出しはそれぞれ、

1., 2., 3., ...; (a), (b), (c), ...; i., ii., iii., ...; A., B., C., ...

のようになります⁴⁵⁾。それ以外の点は `itemize` 環境と同様です。

⁴⁵⁾ これらも、用いている文書クラスに依存します。

注意 9.1

適当なクラスファイル（例えば，article.cls）を読んでもと，

```
\renewcommand\theenumi{\@arabic\c@enumi}
\renewcommand\theenumii{\@alph\c@enumii}
\renewcommand\theenumiii{\@roman\c@enumiii}
\renewcommand\theenumiv{\@Alph\c@enumiv}
\newcommand\labelenumi{\theenumi.}
\newcommand\labelenumii{\theenumii}
\newcommand\labelenumiii{\theenumiii}
\newcommand\labelenumiv{\theenumiv.}
\newcommand\labelitemi{\textbullet}
\newcommand\labelitemii{\normalfont\bfseries \textendash}
\newcommand\labelitemiii{\textasteriskcentered}
\newcommand\labelitemiv{\textperiodcentered}
```

のように...enum... , ...item... という形の名前のコマンドを定義しているところが見つかります．ここで定義されているコマンドは次のようなものです．

- `\labelitem* (* = i, ii, iii, iv)` itemize 環境での各項目の見出し
- `\theenum* (* = i, ii, iii, iv)` enumerate 環境での各項目の見出しの番号の部分
- `\labelenum* (* = i, ii, iii, iv)` enumerate 環境での各項目の見出し
- `\p@enum* (* = i, ii, iii, iv)` enumerate 環境の番号を相互参照 (§15 参照) するときに `\theenum*` に前置される文字列

ただし，itemize 環境，enumerate 環境は最大で 4 段階まで重ねることができますが，上記の * にあてまはる文字列は “itemize 環境または enumerate 環境のレベルを小文字のローマ数字で表記した文字列”（この “レベル” という言葉は先程と同じ意味で用いています）です．そこで，見出しを変更するときにはこれらのコマンドを再定義します．例えば，プリアンブルに

```
\def\labelitemi{\$ \circ \$}
```

という定義を書き込むと，レベル 1 の itemize 環境の見出しは小さな丸になります．一方，上記の `\theenumi`，`\labelenumi` の定義は，

- レベル 1 の enumerate 環境の番号は算用数字で出力します．
- レベル 1 の enumerate 環境の見出しは，番号にピリオドをつけたものにします．

ということです．ここで，`\labelenumi` は

```
\newcommand\labelenumi{\theenumi.}
```

のように，番号だけの部分の `\theenumi` と飾りのピリオドを組み合わせていることに注意してください．(`\labelenumii` などともこれと同様に，`\theenumii` などに飾

りを追加するように定義します.)

例えば, レベル 1 の `enumerate` 環境の見出しについて, (番号は算用数字のままにしておいて) 番号の後にピリオドをつけるのではなく番号を括弧“(”, “)”で囲むようにするにはプリアンブルに

```
\def\labelenumi{(\theenumi)}
```

という定義を書き込むとよいわけです.

さらに, レベル 1 の `enumerate` 環境の番号を小文字のローマ数字で出力したい場合には, 番号部分を表す `\theenumi` の定義を

```
\def\theenumi{\roman{enumi}}
```

のように変更すればよいでしょう. (`enumerate` 環境の見出しの番号は `enumi` `enumii` といった名前のカウンタによって数えられています.)

注意 9.2

箇条書きに関してしばしば耳にする質問に, “箇条書きの項目間に空白が入るのを取り止めたい” というものがあります⁴⁶⁾. 箇条書きの体裁のカスタマイズについては丁寧に説明すると長くなるので, ここではごく手短な対処法だけを紹介します. (興味があれば \LaTeX でのマクロ作成について解説した文献(例えば [6] の第 4 章と第 11 章)を参照してください⁴⁷⁾.)

プリアンブルに下記の記述を入れると, 次のように設定されます.

- 項目間には空白が入らなくなります.
- 最上位の箇条書き全体の前後に限り, 1/2 行分の空白を入れます. 最上位のもの以外の箇条書き全体の前後には空白を追加しません.

```
\makeatletter
\parsep = 0pt \labelsep = .5zw
\def\@listi{%
  \leftmargin = 2zw \rightmargin = 0pt
  \labelwidth\leftmargin \advance\labelwidth-\labelsep
  \topsep = .5\baselineskip
  \partopsep = 0pt \itemsep = 0pt
  \itemindent = 0pt \listparindent = 1zw}
\let\@listI\@listi
\@listi
\def\@listii{%
  \leftmargin = 2zw \rightmargin = 0pt
```

⁴⁶⁾確かに, 1 項目が 1 行に収まるような箇条書きでは項目間の空白が目立ちますが, 1 項目が複数行にわたる場合 (特に欧文の場合) には項目間に多少空白があった方が読みやすいようです. したがって, (箇条書きの) デフォルトの体裁をどうすべきか, というのは一概には言えないところがあります.

⁴⁷⁾実際, ここで紹介した記述は手短に済ますことを最優先にしていますが, 本来はかなりきめ細かに設定できるようになっています.

```

\labelwidth\leftmargin \advance\labelwidth-\labelsep
\topsep      = 0pt \partopsep      = 0pt \itemsep    = 0pt
\itemindent = 0pt \listparindent = 1zw}
\let\@listiii\@listii
\let\@listiv\@listii
\let\@listv\@listii
\let\@listvi\@listii
\makeatother

```

なお，ここで設定しているパラメータは次のような意味をもちます．

- `\parsep`: 箇条書きの中の段落間に追加される空白量
- `\labelsep`: 箇条書きの見出し部分と項目部分との間の間隔
- `\leftmargin`: 箇条書きの左余白 (の増加量)
- `\rightmargin`: 箇条書きの右余白 (の増加量)
- `\labelwidth`: 見出し部分の幅
- `\topsep`: 箇条書き全体の前後の空白量
- `\partopsep`: 箇条書きを段落間で用いたときの `\topsep` の補正量
- `\itemsep`: 項目間に追加される空白量
- `\itemindent`: 箇条書きの各項目の字下げ量
- `\listparindent`: 箇条書きの各項目が複数段落にわたるときの第 2 段落目以降の先頭の字下げ量

必要があれば，`\topsep` の値などをさらに調整するのもよいでしょう．

10. 表 (1) [tabular 環境の基本事項]

10.1. tabular 環境による表

ここでは，“表”を作成する `tabular` 環境について説明します⁴⁸⁾．まず，例を挙げます．

表の例です．

年度 \ 月	4 月	7 月	10 月	1 月
平成 8	55	20	10	7
平成 9	40	7	5	3

これは，次のような入力で得られたものです．

表の例です．

```

\begin{center}
\begin{tabular}{|l||c|c|c|c|}\hline
  年度 \ 月 & 4月 & 7月 & 10月 & 1月 \\ \hline
  平成8     & 55  & 20  & 10   & 7   \\ \hline
\end{tabular}
\end{center}

```

⁴⁸⁾環境名が “`table`” でないことに注意してください． \LaTeX には `table` 環境もありますが，それは “表” そのものを作成する環境ではありません (`table` 環境は次節で扱います)．

```

平成9    & 40 & 7 & 5 & 3 \\ \hline
\end{tabular}
\end{center}

```

この例からわかるように、表の配列要素の区切りには&を使い、1行の終わりは\\で示します。(ただし、表の一番下の横方向の罫線を入れない場合には、最後の行の後には\\を付ける必要はありません。)また、横方向の罫線は\hlineで出力します。ここでは、\begin{tabular}の直後に、|l|c|c|c|c|という文字列をパラメータとして与えていますが、これは表における各々の行の揃え方の指定と縦方向の罫線の入れ方の指定です。これは次のような意味をもちます。

- l, c, r, p{<width>} (<width> は単位付きの長さ) これらの指定のうち、\begin{tabular}{...}の“...”のところで(左から)n番目に現れるものはn列目の揃え方の指定です。l, c, rはそれぞれ、左寄せ、中央寄せ、右寄せの指定で、p{<width>} (<width> は単位付きの長さ)は、“(複数行にわたる)配列要素を含むセルの幅を<width>(+配列要素の左右の余白の分)にする”という指定です。
- |: 縦方向の罫線を入れる位置を指定します。丁寧に言えば、n列目の揃え方の指定と(n+1)列目の揃え方の指定(上記のl, c, r, p{<width>}のこと)の間にある“|”は、n列目と(n+1)列目の間の罫線に対応します。(1列目の指定の前または最後の列の指定の後の“|”はそれぞれ表の左端の罫線、表の右端の罫線に対応します。)

なお、罫線を2重線にしたい場合には、|| (縦の罫線の場合)、\hline\hline (横の罫線の場合)のようにします。(3重線なども同様に出力できます。)また、揃え方の指定において“*{3}{c|}”のように書くと、これは“c|c|c|”と書いたのと同じことになります。一般的に書くと“*{<repeat>}{<item>}” (<item> は繰り返す記述、<repeat> は繰り返しの回数) のようになります。

Note: 各々の行の揃え方の指定は省略できません。また、\\の直後に文字*, [で始まる配列要素を書き込みたい場合には、その*または[の直前に“{”を補ってください。(これは、\\の処理の都合によります。)また、\\[<length>] (<length> は単位付きの長さ) のように記述すると、その改行位置での行間隔を<length>だけ増やすことができます。

表を独立させて配置する場合には、今の例のようにtabular環境全体をcenter環境に入れる(か、後述するtable環境に入れる)とよいのですが、次のように表を本文に埋め込むこともできます。

テキストに表：

	×
×	

を埋め込んでみました。

この出力は次の入力から得られたものです。(ここでは、

```

テキストに表：
\begin{tabular}{|l|r|}\hline
& × \\ \hline

```

```

    x &      \\ \hline
\end{tabular}
を埋め込んでみました。

```

ここで、`\begin{tabular}` の直後に、`[t]`、`[c]`、`[b]` のいずれかのオプション指定を追加すると、(表を本文に埋め込んだ場合の)表の垂直方向の位置の調整ができます。実際、次のようになります。

— INPUT —

テキストに表：

```

\begin{tabular}[t]{|l|r|}\hline
    & x \\ \hline
x & \\ \hline
\end{tabular}
を埋め込んでみました。

```

テキストに表：

```

\begin{tabular}[c]{|l|r|}\hline
    & x \\ \hline
x & \\ \hline
\end{tabular}
を埋め込んでみました。

```

テキストに表：

```

\begin{tabular}[b]{|l|r|}\hline
    & x \\ \hline
x & \\ \hline
\end{tabular}
を埋め込んでみました。

```

— OUTPUT —

テキストに表：

	x
x	

を埋め込んでみました。

テキストに表：

	x
x	

を埋め込んでみました。

テキストに表：

	x
x	

を埋め込んでみました。

この例からわかるように、`[t]`、`[c]`、`[b]` の指定は次のような意味をもちます。

- `[t]`: 表の一番上の罫線を出力する場合には表の上端を表の外部のベースライン⁴⁹⁾に合わせます。表の一番上の罫線を出力しない場合には表の 1 行目のベースライン

⁴⁹⁾一つの行に含まれる文字を並べる際の基準線をベースラインといいます。一般的な欧文フォントでは、個々の大文字の下端がベースラインの位置になります。

を表の外部のベースラインに合わせます .

- [c]: 表の上端と下端のちょうど中央の位置を , “数式の軸” と呼ばれる位置 (分数の横線が出力される位置です) に合わせます .
- [b]: 表の一番下の罫線を出力する場合には表の下端を表の外部のベースラインに合わせます . 表の一番下の罫線を出力しない場合には表の最後の行のベースラインを表の外部のベースラインに合わせます .

なお , [t] のような指定をしない場合には , [c] を指定したものとして扱われます .

10.2. 表の体裁の変更

次に , 表に関係するいくつかのパラメータを紹介します .

- `\tabcolsep`: 隣り合う二つの列の間隔の $1/2$ です . (縦の罫線を入れている場合には ,
 $(\text{罫線の両側の余白}) = \text{\tabcolsep} - (\text{罫線の太さ}) / 2$
 になります .)
- `\arraystretch`: 表における行間隔の拡大率です .
- `\doublerulesep`: 罫線を 2 重 (以上) にするときの罫線間隔です . (罫線の中心どうしの間隔が `\doublerulesep` になります .)
- `\arrayrulewidth`: 罫線の太さです .

これらのパラメータを変更した例を挙げます .

— INPUT —

```
\begin{tabular}[b]{|l|r|}\hline
& \times \\ \hline
\times & \\ \hline
\end{tabular}
{\tabcolsep = 3mm
\begin{tabular}[b]{|l|r|}\hline
& \times \\ \hline
\times & \\ \hline
\end{tabular}}
```

— OUTPUT —

	×		×
×		×	

— INPUT —

```

\begin{tabular}[b]{|l|r|}\hline
& \times \\ \hline
\times & \\ \hline
\end{tabular}
{\renewcommand\arraystretch{1.5}}
\begin{tabular}[b]{|l|r|}\hline
& \times \\ \hline
\times & \\ \hline
\end{tabular}

```

— OUTPUT —

	\times		\times
\times		\times	

— INPUT —

```

\begin{tabular}[b]{|l|r|}\hline
& \times \\ \hline
\times & \\ \hline
\end{tabular}
{\doublerulesep = 5pt}
\begin{tabular}[b]{|l|r|}\hline
& \times \\ \hline
\times & \\ \hline
\end{tabular}

```

— OUTPUT —

	\times		\times
\times		\times	

— INPUT —

```

\begin{tabular}[b]{|l|r|}\hline
& \times \\ \hline
\times & \\ \hline
\end{tabular}
{\arrayrulewidth = 1pt}
\begin{tabular}[b]{|l|r|}\hline
& \times \\ \hline
\times & \\ \hline
\end{tabular}

```

— OUTPUT —

	x		x
x		x	

これらの例のように、`\arraystretch` 以外については “`\tabcolsep=3mm`” のように
変更したいパラメータ = 新しい値 (単位付きの長さ)

のように記述すれば変更できます。

```
\setlength{変更したいパラメータ}{新しい値}
```

という記述を用いても構いません。

一方、`\arraystretch` は

```
\renewcommand\arraystretch{1.5}
```

のように

```
\renewcommand\arraystretch{新しい値 (ただの数値)}
```

のように記述すれば変更できます。

Note: 今の例のようにただ一つの表に対してパラメータを変更する場合には、パラメータを変更する範囲を “{” と “}” で囲みます。一方、文書中のすべての表に対してパラメータを変更したい場合には、パラメータ変更の指定をプリアンブルに書きます。

11. 表 (2) [tabular 環境の応用と table 環境]

前節では、表を出力する `tabular` 環境の基本的なことを説明しましたが、ここでは、`tabular` 環境に関する細かい点と、表を配置するために用いる `table` 環境について説明します。

11.1. 複数の列をまとめること・部分的な罫線

まず、複数の列をまとめて 1 列にするために用いる `\multicolumn` コマンドと横方向の罫線を一部の列だけにいれるために用いる `\cline` コマンドを紹介します。例えば、

年度	月			
	4 月	7 月	10 月	1 月
平成 8	55	20	10	7
平成 9	40	7	5	3

のような出力は、

```

\begin{tabular}{|l|c|c|c|c|}\hline
      & \multicolumn{4}{c|}{月}      & \\\cline{2-5}
年度   & 4 月 & 7 月 & 10 月 & 1 月 & \\\hline
平成 8 & 55   & 20   & 10    & 7    & \\\hline
平成 9 & 40   & 7    & 5     & 3    & \\\hline
\end{tabular}

```

のように記述すると出力できます．ここで，`\multicolumn` は次の書式で用います．

```
\multicolumn{<number>}{<alignment>}{<entry>}
```

ここで $\langle number \rangle$, $\langle alignment \rangle$, $\langle entry \rangle$ はそれぞれ，“まとめる列数”，“1 列分の揃え方の指定（と罫線の入れ方）”，“（ $\langle number \rangle$ 列だけまとめたところに書き込む）配列要素”です．上記の例では，`\multicolumn` 自身は 2 列目に書き込まれていて，かつ $\langle number \rangle = 4$ なので，2 列目から 5 列目までの 4 列がまとめられます．次に揃え方の指定が“c”となっていて， $\langle entry \rangle = “月”$ なので，その 4 列分のスペースに“月”が中央寄せで入れられます．なお，揃え方の指定の仕方は `\begin{tabular}` の直後に記述する各々の列の揃え方の指定のと同様に行います．特に，文字“|”を用いて縦の罫線を入れることもできます．（今の例の場合，|を“c”の前に入れなくても，“月”を含むセルの左側の罫線は入りますが，“c”の右側の|を省略すると，“月”を含むセルの右側の罫線が欠けてしまいます．）

Note: `\multicolumn` を用いた場合，複数の列をまとめたものを一つのセルにします．例えば，3 列分のスペースを二つのセルにしようとして，

```
\multicolumn{3}{c|c|}{...}
```

のように書いた場合，エラーが起こります．

なお，`\multicolumn` コマンドで，“まとめる列数”として“1”を指定した場合，（`\begin{tabular}` の直後で指定した）各々の列の揃え方の指定を一時的に変更することができます．実際，次の例のようになります．

— INPUT —

```

\begin{tabular}{|r|r|}\hline
      \multicolumn{1}{|c|}{年}
      & \multicolumn{1}{c|}{人数} & \\\hline
100 & 1234 & \\\hline
1900 & 15625 & \\\hline
\end{tabular}

```

— OUTPUT —

年	人数
100	1234
1900	15625

次に、`\cline` を用いて $\langle begin \rangle$ 列目から $\langle end \rangle$ 列目までに罫線を引く場合には

```
\cline{\langle begin \rangle}-\langle end \rangle}
```

のように記述します．ここで、普通は $\langle begin \rangle \leq \langle end \rangle$ となるようにしてください．（ $\langle begin \rangle > \langle end \rangle$ でもエラーにはなりませんが、その場合には $\langle begin \rangle$ 列目だけに罫線が引かれます．）なお、1 列目、3 列目、5 列目だけに罫線を引く場合のように途中に切れ目がある罫線を入れるには、

```
\cline{1-1}\cline{3-3}\cline{5-5}
```

のようにひとつながりになっている部分ごとに `\cline` コマンドを用いてください．

注意 11.1

LaTeX のデフォルトでは複数の行を結合する機能は提供されていません．ただし、`\cline` を用いて横の罫線の一部を省略すれば、複数の行を結合したかのような体裁になります．必要があれば、“行を結合した”箇所に対して

```
\raisebox{\vshift}[0pt][0pt]{\parbox{\width}{\text{}}}
```

（ $\langle text \rangle$ は“行を結合した”箇所に配置するテキスト、 $\langle width \rangle$ は $\langle text \rangle$ 部分の幅、 $\langle vshift \rangle$ は $\langle text \rangle$ を上下方向に移動させる移動量）のような記述を用いて配列要素を書き込むとよいでしょう．

注意 11.2

日本語の文書では、“縦書きの配列要素”を含むような表を作成することがあります．そのような場合の“縦書きの配列要素”を作成するには、表の中で一時的に縦組みにするとよいでしょう．例えば、

```
\begin{tabular}{|c|l|} \hline
& 前期 \\ \cline{2-2}
& 中期 \\ \cline{2-2}
& \raisebox{.25zh}[0pt][0pt]{\hbox{\tate 初年度}} \\ \hline
& 後期 \\ \hline
\end{tabular}
```

という記述から、

初 年 度	前期
	中期
	後期

という出力が得られます（`\raisebox` については注意 11.1 を参照してください）．

なお、括弧の類のような横組みの際と縦組みの際に異なる字形を用いるような文字が“縦書きの配列要素”に含まれない場合には、次のように `\shortstack` を用い

るのもよいでしょう .

```
\begin{tabular}{|c|l|} \hline
& 前期 \\ \cline{2-2}
& 中期 \\ \cline{2-2}
& \raisebox{.25zh}[0pt][0pt]{\shortstack{初\\ 年\\ 度}}
& 後期 \\ \hline
\end{tabular}
```

この記述に対する出力は

初	前期
年	中期
度	後期

のようになります .

なお , 1 ページに収まらないような非常に長い表を作成する場合には , longtable パッケージが提供する longtable 環境を利用することができます (環境名を tabular から longtable に変更するだけでもそれなりの出力になります) .

11.2. 幅を指定した表

幅が決まった表を作るには tabular* 環境が使えます . tabular* 環境は

```
\begin{tabular*}{\langle width \rangle}{@{\extracolsep{\fill}}\langle alignment \rangle}
...
\end{tabular*}
```

という書式で用います . ここで , $\langle width \rangle$ は表の幅 (10 cm のように単位付きの長さで指定します) で , $\langle alignment \rangle$ は各々の列の揃え方の指定です . なお , “[t]” のような表の出力位置の指定を行うときには , “ $\{\langle width \rangle\}$ ” の直後に書き込みます . また , $@{\extracolsep{\fill}}$ というのは , 表の全体の幅を $\langle width \rangle$ に合わせるための伸び縮みの指定と思って構いません .

ただし , 幅を指定した表を作る場合 , 単に tabular 環境で作った場合の幅と指定した幅とが大きく異なるときには , “c” , “l” といった揃え方の指定がうまくいきません . また , \cline コマンドでの部分的な横の罫線も正しく出力されません (これらは , tabular* 環境の仕様ようです)⁵⁰⁾ .

11.3. 表の配置

最後に , 文書内に表を配置するために用いる table 環境について説明します . これは ,

```
\begin{table}[位置指定]
  \begin{center}
    \caption{説明文}
```

⁵⁰⁾ 横幅を指定した表を自分の好きなようにうまく作るには , 表を作るためのコマンドのうちで最も基本的な \halign コマンドについて学ぶのが最も良い方法です .

```

\begin{tabular}{|c|c|c|}\hline
表の中身
\end{tabular}
\end{center}
\end{table}

```

のように用います。(これは典型的な例を挙げたもので、`tabular` 環境などを `center` 環境に入れる必要はありません。)なお、2 段組を行っている場合には、`table` 環境は 1 段に収まる表を配置します。表を 2 段抜きにして配置する場合には `table*` 環境を (`table` 環境と同様に) 用います。

ここで、`\begin{table}` の直後の “[位置指定]” というのは、表の出力位置 (出力しても構わない位置の指定) の指定で、“[” と “]” の間には `h`, `t`, `b`, `p` のうちの必要なものを並べます。ここで、`h`, `t`, `b`, `p` はそれぞれ

- `h`: その `table` 環境が書き込まれた位置に出力
- `t`: その `table` 環境が含まれるページの上部に出力
- `b`: その `table` 環境が含まれるページの下部に出力
- `p`: 別のページに出力

という意味です。例えば、“`[tbp]`” のように指定すると、“ページの上部・下部または別ページ” に出力する、という指定になります⁵¹⁾。なお、この出力位置の指定を省略した場合のデフォルトの指定は、(`article.cls` などの) クラスファイルで定義されていますが、多くのクラスファイルでは “`tbp`” になっています。

Note: 図表の配置を行うときに、初心者は “`h`” 指定を使いすぎる傾向があります。しかし、“一般には、図表を文書中の特定の位置に配置することはできない” ということに注意が必要です。(図表を置きたい位置が改ページ箇所にぶつかるような場合を考えてみてください。)したがって、 \LaTeX を用いて作成するような文書においては、図表はなるべくページ上部またはページ下部に出力するようにするとよいでしょう。特に、ユーザ自身が組版⁵²⁾を行うのではない場合 (例えば、学術雑誌への投稿用の原稿や書籍の原稿を作成する場合) には、原則として “`h`” 指定を用いないでください。

また、`\caption` コマンドは表に添える短い説明文 (キャプション) を出力します。ここでは、`\caption` を表の上にも書いていますが、もちろん、表の下にも書いても構いません。なお、`\listoftables` というコマンドを用いると、`\caption` を用いた `table` 環境たちの一覧 (表目次) を作成することができます。

Note: `table` 環境に `\label` (これは相互参照に使います) をつけるときには、`\caption` の後につけてください。また、`\caption` を `{\small ...}` の `...` の部分のようなところに記述した場合には、`\label` も `\caption` が含まれている中括弧の間 (で `\caption` よりも後の部分) に記述してください。

⁵¹⁾この “出力可能な位置の指定” は単に出力可能な箇所を列挙しているだけで、“`t`” などの順序は意味をもちません。例えば、“`tbp`” と “`ptb`” はどちらも “ページの上部・下部または別ページ” に出力するという指定になります。

⁵²⁾原稿などを実際に印刷される体裁に整形する作業のことです。

表 11.1. 出席者数の推移

年度 \ 月	4 月	7 月	10 月	1 月
平成 8	55	20	10	7
平成 9	40	7	5	3

`\caption` で出力される “説明文” については表 11.1 (p. 42) を参照してください .
その表は次のような入力から得られたものです .

```
\begin{table}[t]
\begin{center}
\caption{出席者数の推移}\label{table:sample}
\begin{tabular}{|l||c|c|c|c|}\hline
  年度 \ 月 & 4 月 & 7 月 & 10 月 & 1 月 \\ \hline
  平成 8    & 55  & 20  & 10   & 7   \\ \hline
  平成 9    & 40  & 7   & 5    & 3   \\ \hline
\end{tabular}
\end{center}
\end{table}
```

なお , `table` 環境の番号は `table` という名前のカウンタで管理されているので , このカウンタの値を変更すると表の番号を変更できます . 例えば ,

```
\setcounter{table}{4}
```

のような記述を (`\caption` を用いた) `table` 環境の直前に入れると , その `table` 環境の番号は “5” になります . (次の環境の番号から 1 を引いたものを指定することに注意してください .)

注意 11.3

図の番号は `figure` というカウンタを用いて数えられ , 図の番号は `\thefigure` というコマンドを用いて出力されます . 同様に , 表の番号を数えるためには `table` というカウンタが用いられ , 表番号は `\thetable` というコマンドで出力されます . したがって , `\thefigure` , `\thetable` を再定義すれば , 図・表の番号の形式を変更できます . 例えば , (`jarticle` などの文書クラスを用いていて , 表の番号が単に “1” , “2” のようになっている場合に) 表の番号を , §1 の 2 番目の表の番号が 1.2 となるような ,

(`\section` の番号)(表自身の番号)

という形式で出力させるには ,

```
\makeatletter
\def\thetable{\thesection.\arabic{table}}
\@addtoreset{table}{section}
\makeatother
```


という記述をプリアンプルに入れるとよいでしょう．ここで用いた `\@addtoreset` は、カウンタのリセットの指定を行うコマンドです．この例での

```
\@addtoreset{table}{section}
```

は、“カウンタ `section` が増えるとカウンタ `table` をリセットする” という指定です．一方、`jreport` などの文書クラスを用いているときは、表の番号には `\chapter` の番号が添えられて

(`\chapter` の番号)(表自身の番号)

という形式になっています．ここで、`\chapter` の番号を取り除いて、表自身の番号だけにするには、まず、

```
\def\thetable{\arabic{table}}
```

という定義を用います．ただし、これだけでは、“`\chapter` が変わるとに表の番号が 1 に戻る” という設定はそのままになっています．そこで、 \LaTeX でのカウンタの扱いについて解説した文献(例えば [6] の第 3 章)を参照して、カウンタ `chapter` が増加してもカウンタ `table` がリセットされないような設定を追加するとよいでしょう．

12. 図 (1) [picture 環境・figure 環境]

12.1. picture 環境の書式

ここでは、簡単な図を描くために用いる `picture` 環境と図の配置を行うために用いる `figure` 環境について説明します．まず、`picture` 環境は、

```
\begin{picture}(\langle width \rangle,\langle height \rangle)(x_0,y_0)
(( 図の記述 ))
\end{picture}
```

という形式で用います．ここで、 $\langle width \rangle$ 、 $\langle height \rangle$ はそれぞれ、図の幅と高さで、`\unitlength` を単位とした数値で指定します．この `\unitlength` というのは `picture` 環境での単位長です．(`picture` 環境の中の座標を用いて計算した長さが $\langle length \rangle$ ならば実際に出力される図における長さは $\langle length \rangle \times \text{\unitlength}$ になります．なお、特に指定しなければ、`\unitlength = 1 pt` です．)したがって、図は幅が $\langle width \rangle \times \text{\unitlength}$ 、高さが $\langle height \rangle \times \text{\unitlength}$ の長方形の領域に描かれるわけです．そして、 (x_0, y_0) というのはその長方形の左下の頂点の座標です．なお、 (x_0, y_0) は省略することができて、省略した場合には $(0, 0)$ であるものとして扱われます．`picture` 環境で作成する図の大きさは $(\langle width \rangle, \langle height \rangle)$ で指定しますが、図そのものはこの範囲からはみ出しても構いません．ただし、指定した範囲から図がはみ出した場合、はみ出した部分は図の周囲のテキストに重なることがあります．

Note: `picture` 環境では、描ける図形の種類がかなり限定されるので(例えば、正三

角形は描けません), 複雑な図を入りたいときには他のツールを用いて eps 形式⁵³⁾のファイルを作成して, それを `\includegraphics` コマンド (これは次節で扱います) を用いて取り込むようにするのが一般的です.

12.2. 線分・矢印

次に, `picture` 環境での描画用のコマンドのうち, 主なものを説明します. まず, 線分・矢印は

線分: `\put(x,y){\line(<x_dir>,<y_dir>){<shift>}}`

矢印: `\put(x,y){\vector(<x_dir>,<y_dir>){<shift>}}`

という記述で出力できます. ここで, (x,y) は線分の始点の座標で, $(\langle x_dir \rangle, \langle y_dir \rangle)$ は線分の向き (始点から終点に向かう方向) です. また, $\langle shift \rangle$ は次のような値です.

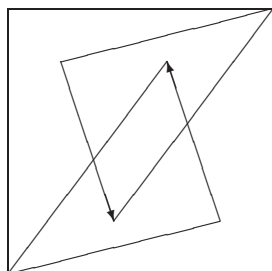
- 線分が垂直でないときには, 始点と終点の x 座標の差の絶対値です.
- 線分が垂直であるときには線分の長さです.

(一般には, $\langle shift \rangle$ は線分の長さではないことに注意してください.)

— INPUT —

```
\begin{picture}(100,100)
  \put( 0, 0){\line(1,0){100}}
  \put( 0, 0){\line(0,1){100}}
  \put( 0, 0){\line(3,4){60}}
  \put( 0, 0){\line(4,1){80}}
  \put(100,100){\line(-1, 0){100}}
  \put(100,100){\line( 0,-1){100}}
  \put(100,100){\line(-3,-4){60}}
  \put(100,100){\line(-4,-1){80}}
  \put( 80, 20){\vector(-1,3){20}}
  \put( 20, 80){\vector(1,-3){20}}
\end{picture}
```

— OUTPUT —



Note: この `\line` コマンドまたは `\vector` コマンドには次の制約があります.

⁵³⁾ 『 \LaTeX 入門』においては画像の形式の一つと理解しておけば充分です.

- $\langle x_dir \rangle$, $\langle y_dir \rangle$ は, $\backslash line$ に対しては絶対値が 6 以下の整数でなければならず, $\backslash vector$ に対しては絶対値が 4 以下の整数でなければなりません. また, $\langle x_dir \rangle$, $\langle y_dir \rangle$ がともに 0 ではない場合, それらの絶対値の最大公約数は 1 でなければなりません.
- 垂直または水平ではないような線分を描くときには, M を $\langle y_dir \rangle / \langle x_dir \rangle$ の絶対値と 1 のうちの大きい方 ($\langle y_dir \rangle / \langle x_dir \rangle = 1$ ならば $M = 1$) とすると,

$$M \times \langle shift \rangle \times \backslash untilength \geq 10 \text{ pt}$$

でなければなりません.

注意 12.1

後述する $\backslash qbezier$ コマンドを用いると, 2 次 Bezier 曲線を描けますが, この $\backslash qbezier$ の三つのパラメータとして, 同一直線上の三つの点の座標を指定すると, 任意の線分を描くことができます. 最も簡単には, 線分の両端と中点の座標を指定して,

```
\qbezier(10,10)(45,50)(80,90)
%% 2 点 (10,10), (80,90) を両端とする線分
```

のようにできます. 線分の両端の座標から中点の座標を自動的に計算して用いるコマンドを用いたい場合には, 次のように定義される $\backslash segment$ を用いることができます.

```
\makeatletter
\def\segment#1(#2)#3({\@segment(#2)()}
\def\@segment(#1,#2)(#3,#4){%
  \@tempdima#1\p@ \advance\@tempdima#3\p@
  \divide\@tempdima\tw@
  \@tempdimb#2\p@ \advance\@tempdimb#4\p@
  \divide\@tempdimb\tw@
  \edef\@segment@temp{\noexpand\qbezier(#1,#2)%
    (\strip@pt\@tempdima,\strip@pt\@tempdimb)(#3,#4)}%
  \@segment@temp}
\makeatother
```

これは, $\backslash segment(10,10)(80,90)$ のように線分の両端の座標を指定して用います.

12.3. 円・円板

円および円板は

円 : $\backslash put(x,y)\{\backslash circle\{\langle diameter \rangle\}\}$

円板: $\backslash put(x,y)\{\backslash circle*\{\langle diameter \rangle\}\}$

のような記述で出力できます. ここで, (x,y) は円の中心の座標で, $\langle diameter \rangle$ は円の直径です.

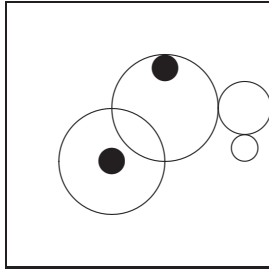
— INPUT —

```

\begin{picture}(100,100)
  \put(0,0){\line(1,0){100}}
  \put(0,0){\line(0,1){100}}
  \put(0,100){\line(1,0){100}}
  \put(100,0){\line(0,1){100}}
  \put(40,40){\circle{40}}
  \put(40,40){\circle*{10}}
  \put(60,60){\circle{40}}
  \put(60,75){\circle*{10}}
  \put(90,60){\circle{20}}
  \put(90,45){\circle{10}}
\end{picture}

```

— OUTPUT —



Note: `\circle` コマンドには次の制約があります .

- 直径が 40 pt を越えるような円または直径が 15 pt を越えるような円板を描くことはできません . ($\langle diameter \rangle$ にこれらの上限よりも大きな値を指定してもエラーにはなりませんが , 実際に出力される円または円板は出力できるサイズのうちで最大のものになります .)
- 円または円板の直径が 15.5 pt 未満のときには , 直径を (pt を単位として表したときに) $\langle diameter \rangle$ で指定した値に最も近い正整数値になるように丸めます . また , 円の直径が 15.5 pt 以上 (40 pt 未満) のときには , 直径を (pt を単位として表したときに) $\langle diameter \rangle$ で指定した値に最も近い 4 の倍数になるように丸めます .

12.4. 線の太さの変更

`\thinlines` , `\thicklines` というコマンドを用いると , `\line` , `\circle` などで描かれる図形における線の太さを変更します . `\thinlines` は , 線の太さを細い方 (通常 0.4 pt) に変更し , `\thicklines` は太い方 (通常 0.8 pt) に変更します . これらのコマンドを用いないときには , `\thinlines` が用いられているものとして扱われます . (なお , これらのコマンドは後述する `\framebox` コマンドなどでの枠の線の太さにも影響します .)

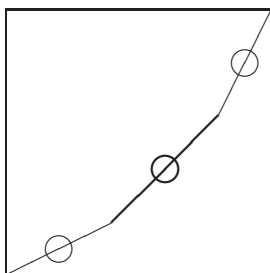
— INPUT —

```

\begin{picture}(100,100)
  \put(0,0){\line(1,0){100}}
  \put(0,0){\line(0,1){100}}
  \put(0,100){\line(1,0){100}}
  \put(100,0){\line(0,1){100}}
  \put( 0, 0){\line(2,1){40}}
  \put(20,10){\circle{10}}
  \thicklines
  \put(40,20){\line(1,1){40}}
  \put(60,40){\circle{10}}
  \thinlines
  \put(80,60){\line(1,2){20}}
  \put(90,80){\circle{10}}
\end{picture}

```

— OUTPUT —



12.5. 図への文字列の書き込み

picture 環境による図の中に、文字列や数式等を記入したい場合は、`\put` を

```
\put(x,y){\langle object \rangle}
```

のように用います。これは、 $\langle object \rangle$ を座標 (x,y) の位置に書き込む、という指定です。ただし、 (x,y) は $\langle object \rangle$ を包む長方形の左下の頂点の座標になります。($\langle object \rangle$ は文字列でなくても構いません。)

— INPUT —

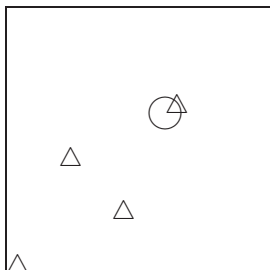
```

\begin{picture}(100,100)
  \put(0,0){\line(1,0){100}}
  \put(0,0){\line(0,1){100}}
  \put(0,100){\line(1,0){100}}
  \put(100,0){\line(0,1){100}}
  \put( 0, 0){\triangle}
  \put(20,40){\triangle}
  \put(40,20){\triangle}
  \put(60,60){\triangle}

```

```
\put(60,60){\circle{12}}
\end{picture}
```

— OUTPUT —



12.6. 同じものの繰り返し

同じものを等間隔に繰り返して並べるには, `\multiput` が使えます. これは,

```
\multiput(x_0,y_0)(\langle x\_shift \rangle,\langle y\_shift \rangle){\langle number \rangle}{\langle object \rangle}
```

のように用います. ここで, $\langle object \rangle$ は並べるもので, (x_0, y_0) は, 1 個目の $\langle object \rangle$ を置く位置の座標, $(\langle x_shift \rangle, \langle y_shift \rangle)$ はある $\langle object \rangle$ から次の $\langle object \rangle$ への移動量で, $\langle number \rangle$ は $\langle object \rangle$ を並べる個数です. したがって, 多くの場合

```
\multiput(x_0,y_0)(\langle x\_shift \rangle,\langle y\_shift \rangle){\langle number \rangle}{\dots}
```

という記述は

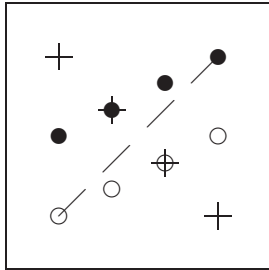
```
\put(x_0,y_0){\dots}
\put(x_0 + \langle x\_shift \rangle, y_0 + \langle y\_shift \rangle){\dots}
\put(x_0 + 2\langle x\_shift \rangle, y_0 + 2\langle y\_shift \rangle){\dots}
...
\put(x_0 + (\langle number \rangle - 1)\langle x\_shift \rangle, y_0 + (\langle number \rangle - 1)\langle y\_shift \rangle){\dots}
```

というコマンドの列と同じ出力を与えます.

— INPUT —

```
\begin{picture}(100,100)
  \put(0,0){\line(1,0){100}}
  \put(0,0){\line(0,1){100}}
  \put(0,100){\line(1,0){100}}
  \put(100,0){\line(0,1){100}}
  \multiput(20,20)(20,10){4}{\circle{6}}
  \multiput(80,80)(-20,-10){4}{\circle*{6}}
  \multiput(15,80)(20,-20){4}{\line(1,0){10}}
  \multiput(20,75)(20,-20){4}{\line(0,1){10}}
  \multiput(20,20)(16.67,16.67){4}{\line(1,1){10}}
\end{picture}
```

— OUTPUT —



12.7. ベジエ曲線

円以外の曲線を描くコマンドとしては, `\qbezier` が用意されています. これは,

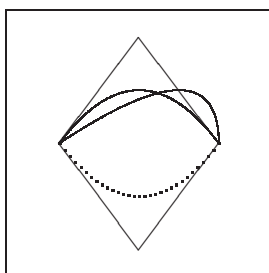
$$\backslash\mathrm{qbezier}(x_0, y_0)(x_1, y_1)(x_2, y_2)$$

のように用いて, 2 点 (x_0, y_0) , (x_2, y_2) を曲線の始点・終点とする 2 次曲線(のうちの特定のパラメータ表示をもつもの)の弧を描きます. その際, 点 (x_0, y_0) から $(x_1 - x_0, y_1 - y_0)$ の方向に向かって曲線を描き始め, 点 (x_2, y_2) には $(x_2 - x_1, y_2 - y_1)$ の方向に進みながら到達します. (始点・終点のそれぞれにおいて引いた接線の交点の座標が (x_1, y_1) になっています. 一般には, `\qbezier` によって描かれる曲線は点 (x_1, y_1) を通らないことに注意してください.) また, `\qbezier[\langle number \rangle](\dots)(\langle number \rangle)` ($\langle number \rangle$ は正整数) のようにオプションをつけると, $[\langle number \rangle]$ をつけない場合と同じ曲線の上に $(\langle number \rangle + 1)$ 個の点を並べた点線を描きます. (ただし, 曲線をパラメータ表示したときのパラメータを等間隔にとるので, 並べられる点の方は等間隔にはなりません.)

— INPUT —

```
\begin{picture}(100,100)
  \put(0,0){\line(1,0){100}}
  \put(0,0){\line(0,1){100}}
  \put(0,100){\line(1,0){100}}
  \put(100,0){\line(0,1){100}}
  \put(20,50){\line(3,4){30}}
  \put(80,50){\line(-3,4){30}}
  \put(20,50){\line(3,-4){30}}
  \put(80,50){\line(-3,-4){30}}
  \qbezier(20,50)(50,90)(80,50)
  \qbezier(20,50)(80,90)(80,50)
  \thicklines
  \qbezier[30](20,50)(50,10)(80,50)
\end{picture}
```

— OUTPUT —



12.8. 枠囲み

文字列 $\langle text \rangle$ を枠に囲んで出力するには、下記のような記述を用いることができます。

実線の枠 : `\framebox($\langle width \rangle$, $\langle height \rangle$){ $\langle text \rangle$ }`

破線の枠 : `\dashedbox($\langle width \rangle$, $\langle height \rangle$){ $\langle text \rangle$ }`

見えない枠: `\makebox($\langle width \rangle$, $\langle height \rangle$){ $\langle text \rangle$ }`

これらは、幅 $\langle width \rangle \times \text{\unitlength}$ 、高さ $\langle height \rangle \times \text{\unitlength}$ の大きさの枠の中に文字列 $\langle text \rangle$ を入れたものを出力します。(したがって、 $\langle width \rangle$ と $\langle height \rangle$ は単なる数値を用いて指定します。) また、`\framebox` などで枠に入れる文字列と枠との位置関係を指定する場合には、

`\framebox(x , y)[$\langle position \rangle$]{...}`

のようにオプションをつけます (`\dashedbox`、`\makebox` の場合も同様です)。また、 $\langle position \rangle$ の意味は表 12.1 に示すとおりです。

簡単に言えば、“t”、“b”、“l”、“r” はそれぞれ“上”、“下”、“左”、“右”に対応しているわけです。また、“tl”のように2文字で指定する場合にはその文字の順序は逆

表 12.1. `\framebox` などでの位置指定オプションの意味

$\langle position \rangle$	位置関係
c	: 枠の中央
tl	: 枠の左上隅
t	: 枠の上辺の中央
tr	: 枠の右上隅
r	: 枠の右辺の中央
br	: 枠の右下隅
b	: 枠の下辺の中央
bl	: 枠の左下隅
l	: 枠の左辺の中央

になっても差し支えありません。(これらの指定を行わないときには “[c]” が指定されているものとして扱われます。) 枠の線の太さは `\thinlines`, `\thicklines` を用いて変更できます。なお, `\linethickness` コマンドを

```
\linethickness{<thickness>}
```

(`<thickness>` は線の太さ (単位付きの長さ)) のように用いて, 枠の線の太さを直接指定することもできます。

また, `\dashbox` コマンドに対しては,

```
\dashbox{<length>}<width>,<height>{<text>}
```

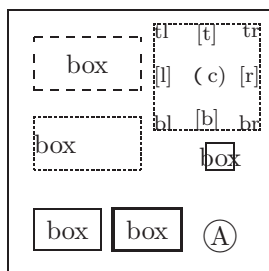
(`<length>` は正の数値) のようにオプションの `<length>` を指定すると, 破線の実線部分と空白部分の長さを `<length> × \unitlength` にすることができます。(`<length>` を指定しない場合には “1” が指定されたものとして扱われ, (`\unitlength` が 1pt である場合には) 破線というよりも点線の枠になります。)

これらのコマンドによる枠を図の中に置く場合には `\put` コマンドを併用して枠の位置を指定します。次の例を参照してください。

— INPUT —

```
\begin{picture}(100,100)
  \put(0,0){\line(1,0){100}}
  \put(0,0){\line(0,1){100}}
  \put(0,100){\line(1,0){100}}
  \put(100,0){\line(0,1){100}}
  \put(10,70){\dashbox{3}(40,20){box}}
  \put(10,40){\dashbox(40,20)[l]{box}}
  \put(80,15){\makebox(0,0){A}}\put(80,15){\circle{12}}
  \put(75,40){\framebox(10,10){box}}
  \put(55,55){\dashbox(40,40)[t1]{\footnotesize t1}}
  \put(55,55){\makebox(40,40)[t]{\footnotesize [t]}}
  \put(55,55){\makebox(40,40)[tr]{\footnotesize tr}}
  \put(55,55){\makebox(40,40)[r]{\footnotesize [r]}}
  \put(55,55){\makebox(40,40)[br]{\footnotesize br}}
  \put(55,55){\makebox(40,40)[b]{\footnotesize [b]}}
  \put(55,55){\makebox(40,40)[bl]{\footnotesize bl}}
  \put(55,55){\makebox(40,40)[l]{\footnotesize [l]}}
  \put(55,55){\makebox(40,40)[c]{\footnotesize ( c )}}
  \put(10,10){\framebox(25,15){box}}
  \thicklines
  \put(40,10){\framebox(25,15){box}}
\end{picture}
```

— OUTPUT —



12.9. 図の配置

§11 で扱ったように、必要に応じてキャプションをつけた表を配置するには `table` 環境を用いましたが、必要に応じてキャプションをつけた図を配置するには `figure` 環境を用います。 `figure` 環境は `table` 環境と同様に次の形式で用います。

```
\begin{figure}[図の位置指定]
  \begin{center}
    \begin{picture}(...)
      図の中身
    \end{picture}
    \caption{説明文}
  \end{center}
\end{figure}
```

ここで、`\begin{figure}` の直後の図の位置指定の記述の仕方は `table` 環境の場合と同じなので、§11 を参照してください。(また、`picture` 環境を `center` 環境に入れる必要はありません。) なお、2 段組時には、`figure` 環境は 1 段に収まる図を配置します。図を 2 段抜きにして配置する場合には `figure*` 環境を用います。

`figure` 環境の使用例については、図 12.1 (p. 52) を参照してください。その図は次のような入力から得られたものです。

```
\begin{figure}[b]
  \begin{center}
    \begin{picture}(100,80)
      \put(20,23.33){\line(1, 0){60}}
      \put(20,56.67){\line(1, 0){60}}
```

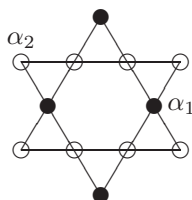


図 12.1. figure 環境のサンプル

```

\put(20,23.33){\line(3, 5){30}}
\put(20,56.67){\line(3,-5){30}}
\put(50, 6.67){\line(3, 5){30}}
\put(50,73.33){\line(3,-5){30}}
\multiput(20,23.33)(20,0){4}{\circle{6}}
\multiput(20,56.67)(20,0){4}{\circle{6}}
\multiput(50, 6.67)(0,66.66){2}{\circle*{6}}
\multiput(30,40)(40,0){2}{\circle*{6}}
\put(20,62){\makebox(0,0)[b]{\alpha_2}}
\put(75,40){\makebox(0,0)[l]{\alpha_1}}
\end{picture}
\caption{figure 環境のサンプル}\label{figure:sample}
\end{center}
\end{figure}

```

この例では、`\put(...){\makebox(0,0)...` を用いて、文字列を書き込む位置を細かく指定していることにも注意してください。

なお、figure 環境の番号は figure という名前のカウンタで管理されているので、このカウンタの値を変更すると表の番号を変更できます。例えば、

```
\setcounter{figure}{6}
```

のような記述を (`\caption` を用いた) figure 環境の直前に入れると、その figure 環境の番号は “7” になります。

13. 図 (2) [graphicx パッケージ]

ここでは、画像ファイル (主に eps ファイル) の貼り付けや、テキストおよび画像の拡大・回転を行うために用いる graphicx パッケージについて、基本的なことを説明します。なお、本節で説明している機能を用いるには、プリアンブルに

```
\usepackage[dvips]{graphicx}
```

のような graphicx パッケージを読み込む指定を入れる必要があります。ここで、“dvips” というオプション (ドライバ指定) はお使いの dviware に応じて変更してください⁵⁴⁾。なお、プレビューと印刷に異なる dviware を使い分けるような場合には、使用する可能性がある dviware に対応する指定をすべて並べるのではなく、プレビュー用の dvi ファイルを作成する際にはプレビュー用の dviware に対応する指定のみを行ってコンパイルし、印刷用の dvi ファイルを作成する際には印刷用の dviware に対応する指定のみを行ってコンパイルするという具合に、一度に適用するドライバ指定はただひとつにしてください (この点については、下記の (2) も参照してください)。

ここで、graphicx パッケージに関しては次のことに注意してください。

- (1) graphics パッケージというものも存在しますが、graphicx パッケージは graphics

⁵⁴⁾概ね dviware の名称になりますが、“dvips” にすれば PostScript 経由で表示・印刷ができます。

パッケージの機能を拡張したものです。(ただし, graphicx パッケージは graphics パッケージを必要とします.)

- (2) graphicx パッケージで提供されている機能は, $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 自身の機能ではありません. 本節で説明するコマンドたちは, 図の取り込みなどの操作を行うための記述を dvi ファイルに埋め込み, それを dviware に解釈させます. したがって, お使いの dviware によっては, この節で説明している例の一部 (場合によっては全部) が正しく表示できないことがあります.
- (3) (2) と関係することですが, 本節で説明している機能を用いたソースファイルをコンパイルして得られる dvi ファイルはお使いの dviware に依存するようになります. (graphicx パッケージを用いるときのオプションで, dviware の違いを吸収することになっていますが, (各 dviware の graphicx パッケージへの対応度の違いにより) 完全に対応できるとは限りません.) 処理環境の違いを気にしなければならないような文書を作成する場合には eps ファイルの貼り付け (オプションなしの `\includegraphics`) 以外の機能は用いないか, せいぜい画像の拡大・縮小の指定 (および “clip” 指定) を用いる程度にするのが無難です.

なお, 本稿では eps ファイルなどの作成法については扱いません. eps ファイルの作成に関しては, 画像 (またはグラフ) 作成用のソフトウェアに関する適切な文献またはウェブページなどを参照してください.

13.1. 図およびテキストの回転・拡大

まず, 図およびテキストに対する操作 (回転など) を行うコマンドを説明します. また, 出力例で用いている `\dots` コマンドは, “ベースラインの位置に, 横に 3 個並んだ点” を出力するためのコマンドです.

図やテキスト `\langle object \rangle` を角度 `\langle angle \rangle` 度だけ回転させるには `\rotatebox` コマンドを

$$\backslash rotatebox[\langle options \rangle]{\langle angle \rangle}{\langle object \rangle}$$

のように用います (`[\langle options \rangle]` の部分は省略できます). また, `[\langle options \rangle]` は回転の中心を次の形式で指定します. (`[\langle options \rangle]` を省略した場合, “`x=0pt,y=0pt`” が指定されているものとして扱います.)

- (1) `origin=\langle position \rangle`:

これは, `\langle object \rangle` をちょうど含むような長方形領域に対して, 回転の中心を `\langle position \rangle` で指定される位置にします. ただし, `\langle position \rangle` は, “`r`”, “`l`”, “`t`”, “`b`”, “`B`” からなる (2 文字以下の) 文字列または “`c`” です. ここで, `\langle position \rangle` の意味は, 文字 “`B`” が含まれないときには, `\framebox` などにおける位置の指定 (§12 を参照してください) と同様です. また, 文字 “`B`” は, 回転の中心の垂直方向の位置を (`\langle object \rangle` の) ベースラインの位置にします. (水平方向の位置は “`r`”, “`l`”, “無指定” に応じて (長方形領域の) 右端, 左端, 中央になります.)

- (2) `x=\langle x_coord \rangle,y=\langle y_coord \rangle`:

これは, 回転の中心を, 長方形領域の左端から `\langle x_coord \rangle` だけ右に進み, ベースラ

インから $\langle y_coord \rangle$ だけ上に上がった位置にします．なお， $\langle x_coord \rangle$ ， $\langle y_coord \rangle$ はともに単位付きの長さです．

使用例を挙げます．

— INPUT —

```
\dots Patapata
\dots \rotatebox{30}{Patapata}
\dots \rotatebox{120}{Patapata}
\dots

\dots \rotatebox[origin=c]{30}{Patapata}
\dots \rotatebox[origin=tr]{30}{Patapata}
\dots \rotatebox[x=2em,y=0pt]{30}{Patapata}
\dots
```

— OUTPUT —

また，図やテキストの拡大・縮小は，

```
\scalebox{ $\langle h\_scale \rangle$ }[ $\langle v\_scale \rangle$ ]{ $\langle object \rangle$ }
\resizebox{ $\langle h\_size \rangle$ }{ $\langle v\_size \rangle$ }{ $\langle object \rangle$ }
```

という記述でできます([$\langle v_scale \rangle$] の部分は省略できます). これらのうち， $\backslash scalebox$ は $\langle object \rangle$ を水平方向に $\langle h_scale \rangle$ 倍に拡大し，垂直方向に $\langle v_scale \rangle$ 倍に拡大します．ただし， $\langle v_scale \rangle$ を省略したときには $\langle v_scale \rangle = \langle h_scale \rangle$ であるものとして扱われます．なお， $\langle h_scale \rangle$ (または $\langle v_scale \rangle$) として負の値を用いた場合には，左右 (または上下) を逆にした後に $\langle h_scale \rangle$ (または $\langle v_scale \rangle$) の絶対値倍に拡大します．($\langle h_scale \rangle$ などに 0 を用いることはできません．) また， $\backslash scalebox{-1}[1]$ の代わりに $\backslash reflectbox$ を用いることもできます．

一方， $\backslash resizebox$ は， $\langle object \rangle$ を幅が $\langle h_size \rangle$ になり，ベースラインよりも上の部分の高さが $\langle v_size \rangle$ になるように拡大します．ただし， $\langle h_size \rangle$ ， $\langle v_size \rangle$ の一方のみに “!” を用いた場合には，“!” でない方の寸法を指定した値にし，“!” にした方は， $\langle object \rangle$ の本来の縦横比を保つような値に調整されます．(両方とも “!” にした場合，すなわち $\backslash resizebox{!}{!}{\dots}$ は $\backslash mbox{\dots}$ として扱われ，原寸大で出力されます．) なお， $\langle object \rangle$ 全体の高さが $\langle v_size \rangle$ になるように拡大したい場合には $\backslash resizebox*$ のように * をつけて用います．

— INPUT —

```

\dots Patapata
\dots \scalebox{1.5}{Patapata}
\dots \scalebox{2}{.7}{Patapata}
\dots

\dots \reflectbox{Patapata}
\dots \scalebox{1}{-1}{Patapata}
\dots \scalebox{-1}{-1}{Patapata}
\dots

\dots \resizebox{5em}{1.5em}{Patapata}
\dots \resizebox{5em}{!}{Patapata}
\dots \resizebox{!}{1.5em}{Patapata}
\dots

```

— OUTPUT —

```

...Patapata ...Patapata ...Patapata...
...stsbataQ ...stsbataQ...
...Patapata ...Patapata ...Patapata...

```

13.2. eps ファイルの取り込み

次に, eps ファイルの取り込み方について説明します. 例えば, ファイル fig.eps は 図 13.1 (p. 56) のような画像を与えるものとします. このとき, 単に,

```
\includegraphics{fig.eps}
```

という記述で, この図が原寸大で出力されます.

拡大・縮小を行うには,

```

\includegraphics[width=<h_size>, height=<v_size>]{<epsfile>}
\includegraphics[scale=<factor>]{<epsfile>}

```

のように記述します. ここで, <epsfile> は eps ファイルのファイル名で, <h_size> , <v_size>

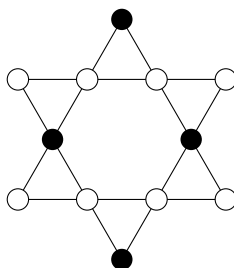


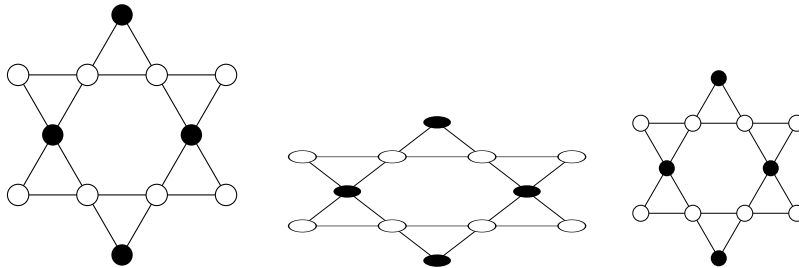
図 13.1. 説明用のサンプル画像

はそれぞれ拡大後の幅と高さです．また， $\langle factor \rangle$ は拡大率です．実際，次のようになります．

— INPUT —

```
\includegraphics{fig.eps}
\includegraphics[width=1.8in,height=0.8in]{fig.eps}
\includegraphics[scale=.75]{fig.eps}
```

— OUTPUT —

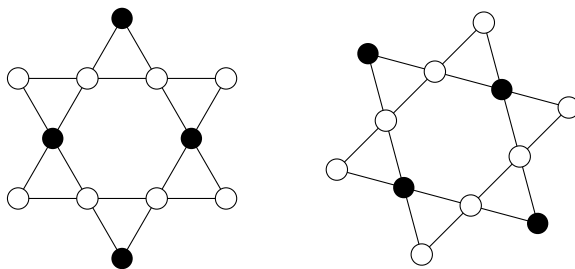


図を回転させることもできます．回転の中心は，(`\includegraphics` のオプション (`[と]` の間の記述) の中で) `\rotatebox` の場合と同様に `origin= $\langle position \rangle$` という形式で指定します．また，回転角は `angle= $\langle angle \rangle$` ($\langle angle \rangle$ は回転角) のように与えます．次の例を参照してください．

— INPUT —

```
\includegraphics{fig.eps}
\includegraphics[origin=c,angle=45]{fig.eps}
```

— OUTPUT —



もちろん，回転と拡大を同時に行うこともできます．また，`\scalebox` などを用いて変形する部分に `\includegraphics` コマンドを入れることもできます．(読者自身でいろいろ実験してみるとよいでしょう．) なお，`\includegraphics` では，オプション指定によってトリミングなどを行うこともできます．しかし，これ以上複雑なことをするよりは，あらかじめ画像の方を編集しておいてから取り込むようにするのが賢明ですから，ここではこれ以上深入りしないことにします．

14. 脚注

文章に脚注を入れるには、基本的には `\footnote` コマンドを用います。例えば⁵⁵⁾、今の“例えば”の直後に“55)”という文字列が現れていて、かつ、このページの下部に⁵⁵⁾説明用の脚注です。

という記述があります。この脚注は単に

```
\footnote{説明用の脚注です。}
```

という記述で作成しました。この例からわかるように、脚注は

```
\footnote{⟨footnote⟩}
```

(`⟨footnote⟩` は脚注にするテキスト) のように記述します。ただし、本稿での脚注記号は、“1)”, “2)”, ... という形式にしていますが、多くの文書クラスのデフォルトでは単に“1”, “2”, ... という形式になっています。

脚注記号は自動的につきますが、(脚注記号に対応する) 番号を指定したい場合には、

```
\footnote[⟨number⟩]{⟨footnote⟩}
```

(`⟨number⟩` は脚注の番号) のように記述します。例えば、

```
\footnote[100]{番号 100 の脚注です。}
```

という記述を用いると¹⁰⁰⁾ (ちょうどここで用いました)、このページの下部にあるように番号 100 の脚注になります⁵⁶⁾。ただし、この例以外の脚注をみるとわかるように、番号を指定しない脚注の番号は、番号を指定した脚注を無視して数えられています。したがって、脚注の番号を特定の番号 N から始めたい場合には、`\section` などの番号の指定の仕方と同様に、

```
\setcounter{footnote}{ $N - 1$ }
```

(“ $N - 1$ ” の部分は計算後の数値を具体的に記述します⁵⁷⁾) という記述を、番号 N の脚注の直前に入れます。

なお、脚注記号の形式を変更する場合には、`\thefootnote` または `\@makefnmark` の定義を変更します。例えば、プリアンプルに

⁵⁵⁾説明用の脚注です。

¹⁰⁰⁾番号 100 の脚注です。

⁵⁶⁾なお、(p)LaTeX の標準クラスファイルなどを用いている場合には `\footnote` のオプションは脚注番号に対応する整数ですが、脚注記号そのものではないことに注意が必要です。例えば、(minipage 環境でのデフォルトの設定のように) 脚注記号を“a”, “b”, ... のようにつけることにすると、“`\footnote[3]{...}`” のように番号を指定した脚注の脚注記号は“c” になります。

⁵⁷⁾calc パッケージを使用した場合には、計算式を使っても構わないようになります。


```

\makeatletter
\def\thefootnote{\arabic{footnote}}
\def\@makefnmark{\textsuperscript{\nomalfont\@thefnmark}}
\makeatother

```

という記述を入れると、脚注記号が、“1)”、“2)”、... という形式（上付き）で出力されるようになります。ここで、\thefootnote は脚注記号の番号部分のみ（今の例では“1”、“2”の部分）の形式を表します。番号部分に添える“装飾”は \@makefnmark の定義に補います。 \@makefnmark の定義の中の \@thefnmark というのが脚注記号の“番号部分”なので、今の例では \@thefnmark に “)” を後置しています⁵⁸⁾。

また、\footnote コマンドを用いる場所によっては、脚注が出力されないときがあります。例えば、表の配列要素に脚注をつけようとして、

```

\begin{center}
\begin{tabular}{|c|c|c|}\hline
月 & 回答者数 & 評価 \footnote{アンケートの回答（1--5）の平均}
\\ \hline
4 & 30 & 3.067 \\ \hline
10 & 25 & 3.120 \\ \hline
\end{tabular}
\end{center}

```

のように記述した場合、表の中には脚注記号がつきますが、脚注自身は出力されません（実際に処理して確認してみてください）。このような場合には、\footnotemark と \footnotetext を用いて、

```

\begin{center}
\begin{tabular}{|c|c|c|}\hline
月 & 回答者数 & 評価 \footnotemark[1] \\ \hline
4 & 30 & 3.067 \\ \hline
10 & 25 & 3.120 \\ \hline
\end{tabular}
\end{center}
\footnotetext[1]{アンケートの回答（1--5）の平均}

```

のようにします。ここで、\footnotemark、\footnotetext は、それぞれ、

- \footnotemark: 脚注記号のみを出力
- \footnotetext: 脚注のテキストのみを出力

という処理を行うコマンドで、

```

\footnotemark[⟨number⟩]
\footnotetext[⟨number⟩]{⟨footnote⟩}

```

⁵⁸⁾この例では、\thefootnote を “\def\thefootnote{\arabic{footnote}}” のように再定義するのは避ける方が無難です。（相互参照の際に脚注の参照を行わないか、相互参照の際にも脚注記号が“1)”の形式で出力されてもよいのであればこの限りではありません。）

($\langle number \rangle$ は脚注の番号, $\langle footnote \rangle$ は脚注にするテキストで, $[\langle number \rangle]$ の部分は省略可) のように用います.

注意 14.1

表に対する注釈をその表の直後に置くこともしばしばありますが, そういうレイアウトは, 表と注釈を `minipage` 環境に入れれば実現できます. ただし, `minipage` 環境内では表の中であっても `\footnote` をそのまま用いることができます. その代わり, `minipage` 環境内の `\footnote` は “`minipage` 環境の中の” 脚注を作成します.

一方, `minipage` 環境の中の `\footnotemark` は, `minipage` 環境の外部で用いている脚注記号の形式と番号を用いて脚注記号を作成します. したがって,

```
... \footnote{...}%% (1)
\begin{center}
  \begin{minipage}{***}% ***の部分は適当な長さ
    ... \footnote{...}%% (2)
    ... \footnotemark%% (3)
  \end{minipage}
\end{center}
\footnotetext{...}%% (4)
```

のように記述した場合, (2) の脚注は `minipage` 環境内の注釈になります. また, (3) の `\footnotemark` は (1) の脚注の番号の次の番号に対応する脚注記号を作成します. そして (4) の `\footnotetext` は (3) で作成した脚注記号に対応する脚注のテキストを与えます.

15. 相互参照

15.1. 相互参照の方法

\LaTeX では, 章・節などの文書構造の番号や, 数式 (後述します) の番号などの相互参照を行うためのコマンドが用意されています. 具体的には,

- 参照されるところに “`\label{label-string}`” と記入します.
- 参照を行うところに “`\ref{label-string}`” と記入します.

ここで, $\langle label-string \rangle$ は “ラベル” として用いる文字列です. ($\langle label-string \rangle$ には, ソースファイルの中にそのまま書き込むことができる文字ならば問題なく使えます.) 例えば, 本稿の原稿ファイルのこの節の冒頭には,

```
\label{sec:cross-references}
```

というラベルをつけています. そこで, 文書中に “`\ref{sec:cross-references}`” と記述すると, 単に本節の番号の “15” が出力されます. これは, `\section` の番号を参照した例ですが, `\subsection{...}` の直後で `\label` を用いると, その `\subsection` の番号を参照することができます. また, `enumerate` 環境などの `\item` の直後で用いると, その項目の番号を参照することができます. (`\label` を用いた位置において \LaTeX

が参照用の“番号”として用意しているものが参照されるわけです。)なお、`\ref`の代わりに`\pageref`を用いると、対応する`\label`が書き込まれた位置のページ番号(やはり番号のみです)が出力されます。

ここで、`\ref`は、参照対象の番号のみを出力し、(特に何も設定しない限り)“§”のような記号は、文書の作成者自身が補わなければなりません。これは一見すると不便なようにも思えますが、複数の`\ref`を並べる場合(例えば、“§§10–11 参照”のような記述を行う場合)を考えると、`\ref`または`\pageref`は余分な記号を出力しないように定義してあった方が都合がよいとわかります。

相互参照の仕方は上記のとおりで、あとは実際に使ってみると要領がつかめることでしょう。なお、“ラベル”として用いる文字列は、同じものを2回以上用いることがなければどのようにつけても構いません。しかし、“section1”のような、参照される番号を含むようなラベルを用いることは避けましょう⁵⁹。(ラベルをつけた後に文章を変更した場合、`\label`たちの順序が入れ換わることがある、ということに注意してください。)

15.2. 相互参照のしくみ

次に、 \LaTeX の相互参照のしくみの概略を説明します。

- (1) `\begin{document}`のところで相互参照に関して行われること:
aux ファイルがあればそれを読み込み、相互参照に関する情報を取得します。
- (2) 本文中の`\label`、`\ref`、`\pageref`が行うこと:
 - `\label`: 相互参照に関する情報を aux ファイルに書き出します。
 - `\ref`、`\pageref`: (1) で取得した情報を用いて参照された番号を出力します。ただし、(1) で取得した情報の中に`\ref`、`\pageref`が参照しているラベルについての情報がない場合には、“??”という文字列を出力します。
- (3) `\end{document}`のところで相互参照に関して行われること:
aux ファイルを再び読み込み、(1) で取得した情報と(2) で出力された情報との間に相違がないかどうかを検査します。

\LaTeX は相互参照をこのようにして行っているので、相互参照を行っているファイル进行处理するときには少なくとも2回はコンパイルを行う必要があります。なぜなら、1回目のコンパイルの時点では、aux ファイルには相互参照に関する(正しい)情報が書き込まれていないので、文書中の`\ref`、`\pageref`のところでは、“??”(または間違った情報)が出力されるからです。また、2回目のコンパイルでは、1回目に aux ファイルに書き込まれた情報を用いて、`\ref`と`\pageref`はほぼ正しい出力を与えます。

ここで、“ほぼ正しい”と書きましたが、実際、1回目のコンパイル時には“??”のように出力されていたところが、2回目には別の文字列に置き換わるわけですから、その結果、各々の`\label`の位置におけるページ番号がずれるということが考えられます。また、目次などの作成の都合でページ番号が変化することもあります。そこで、

⁵⁹文字列“section1”の“1”を“単なる識別番号”と割り切って、“section1 というラベルがついていたとしても第1節であるは限らない”ということを理解して相互参照を行うのであれば、この限りではありません。

LaTeX は (3) の処理を行って、相互参照に関する情報に変化がないか否かを検査するわけです。

次に、相互参照に関する LaTeX の警告メッセージとその各々への対処法を説明します。なお、これらのメッセージは log ファイルの末尾付近に出力されます。

- LaTeX Warning: Label(s) may have changed.
Rerun to get cross-references right.
これは、相互参照に関する情報に変化があったことを表します。このメッセージが現れなくなるまでコンパイルを繰り返してください。
- LaTeX Warning: There were multiply-defined labels.
これは、ラベルとして用いた文字列たちに重複があることを表します。このメッセージが現れた場合には、log ファイルの中に
LaTeX Warning: Label '*⟨label-string⟩*' multiply defined.
または
LaTeX Warning: Citation '*⟨label-string⟩*' multiply defined.
(*⟨label-string⟩* は重複しているラベル) というメッセージがあるので、*⟨label-string⟩* というラベルを用いた `\label`、`\ref` および `\bibitem` (これは §17 で説明します) などを調べて、ラベルに重複がなくなるようにラベル名を変更してください。
- LaTeX Warning: There were undefined references.
これは、`\ref` または `\pageref` に対応する `\label` が存在しない (または `\cite` に対応する `\bibitem` が存在しない (`\cite`、`\bibitem` は、§17 で扱います)) ことを表します。このメッセージが現れた場合には、log ファイルの中に
LaTeX Warning: Reference '*⟨label-string⟩*' on page ..
undefined on input line ...
(*⟨label-string⟩* は “`\label{⟨label-string⟩}`” を欠いたラベル) または
LaTeX Warning: Citation '*⟨label-string⟩*' on page ..
undefined on input line ...
(*⟨label-string⟩* は “`\bibitem{⟨label-string⟩}...`” を欠いたラベル) というメッセージがあります。それに応じて、適切な箇所に `\label{⟨label-string⟩}` (または `\bibitem{⟨label-string⟩}...`) を追加するか、余分な `\ref`、`\pageref`、`\cite` を削除してください。

なお、プリアンブルに `\nofiles` というコマンドを書きこむと、aux ファイルなどの更新が行われなくなります。したがって、“LaTeX Warning: Label(s) may have changed.” という警告メッセージが現れなくなった後 (すなわち、相互参照についての情報が確定した後) に、プリアンブルに `\nofiles` と指定すると、文書の一部だけを処理しても差し支えないようになります。

Note: figure 環境、table 環境に `\label` をつけるときには、その環境の中の `\caption` の後の部分につけてください。(`\caption` を用いていない figure 環境または table 環境には (図表の番号がつかないので)、`\label` を用いても意味がありません。)

16. 目次

目次を出力するには、目次を出力したい位置で `\tableofcontents` コマンドを用います。本稿の冒頭の目次も `\tableofcontents` コマンドで作成しました⁶⁰⁾。

また、`\listoftables`、`\listoffigures` を用いると、それぞれ表目次、図目次を出力することができます。ただし、これらのコマンドによる図目次・表目次は `table` 環境または `figure` 環境を用いて出力した表または図のうち `\caption` をつけたものについての目次になります⁶¹⁾。

ここで、目次作成のしくみの概略について説明します。

- `\tableofcontents` コマンドが行うこと:
処理しているソースファイルの拡張子を `.toc` に変えたファイル（以下、“`toc` ファイル”といいます）が存在すればそれを読み込んで目次を作成します。そうでなければ単に目次の見出しのみを出力します。
- `\section` などが目次の作成に関して行うこと:
`aux` ファイルに個々の節の番号・見出し・ページ番号といった目次の作成についての情報を書き出します。
- `\end{document}` のところで、目次の作成に関して行われること:
(2) で `aux` ファイルに書き出された情報を用いて `toc` ファイルを作成します。

（なお、表目次、図目次はそれぞれ、拡張子が `.lot`、`.lof` のファイルを用いて作成されます。） \LaTeX はこのようにして目次を作成するので、目次を含むファイルを処理するときには、普通は少なくとも 3 回のコンパイルを必要とします。実際、1 回目のコンパイル時には、`toc` ファイルが存在しない（存在していても目次に関する情報は正しくない）ので、目次は正しくありません。また、1 回目のコンパイルで作成される `toc` ファイルは、目次が正しくない出力におけるページ番号を用いているので、ページ番号が（普通は目次のページ数だけ）ずれています。2 回目のコンパイルでは、まだ目次は正しくありませんが、`toc` ファイルにはほぼ正しいページ番号が書き込まれます。そこで、3 回目のコンパイルになるとほぼ正しい目次が得られるわけです。

ただし、相互参照のときとは異なり、目次に関する情報に変更があっても何も警告はありません。（参照するか否かには関係なく、すべての `\section`、`\subsection` などの直後に `\label` をつけておくとい良いでしょう。そうすると、`\section` などの位置におけるページ番号が変更されると、相互参照に関する情報も変更されるので、相互参照に関する警告が現れます。（相互参照に関する警告が現れなくなったら、目次も正しくなっているというわけです。）なお、相互参照のときと同様にプリアンブルで `\nofiles` を用いると、`toc` (`lot`, `lof`) ファイルの更新を行わないようになります。

注意 16.1

目次に掲載される項目は自動的に抽出されますが、特定の項目を追加する場合には

⁶⁰⁾目次の体裁は多少調整していますが、`jarticle.cls` などのクラスファイルを用いた場合と同様の目次を作成しました。

⁶¹⁾図目次・表目次にはキャプション文字列が用いられます。キャプション文字列を図目次・表目次に載せるための処理は（原則として）`\caption` コマンドによって行われます。


```
\addcontentsline{toc}{section}{\langle entry \rangle}
```

(これは、 $\langle entry \rangle$ を “section” レベルの項目として目次に掲載する指定です) のような記述を、本文中の適当な位置 (項目 $\langle entry \rangle$ に対応する箇所) に記述します。

一方、目次項目を削除する場合には toc ファイルから削除したい項目に対応する記述を削除します。たいていの場合、削除することになるのは `\contentsline` から始まる行で、`\contentsline` のパラメータになっている文字列を見れば削除する項目の見当はつきます。ただし、toc ファイルを手動で変更する場合には、`\nofiles` コマンドをプリアンプルに入れて toc ファイルの更新を抑制する必要があります。

なお、これらはあくまでも “対症療法” ですが、目次に抽出する項目のカスタマイズを行うには \LaTeX の内部に関する知識を多少必要とします。

17. 参考文献の作成

次に、参考文献の作成に用いる環境と文献の引用に用いるコマンドについて説明します。まず、参考文献は、`thebibliography` 環境を用いて記述するのが普通です。(ここでは、 $\text{BiB}\text{\TeX}$ と文献データベースを併用する方法については述べません。ただし、 $\text{BiB}\text{\TeX}$ は、文献データベースと、aux ファイルの中の “引用文献に関する情報” を用いて、“読み込ませるべき `thebibliography` 環境” を書き込んだファイルを作成します。)

まず、`thebibliography` 環境は次のように記述します。(“[*reference*] ” の部分は省略できます。)

```
\begin{thebibliography}{\langle longest-label \rangle}
\bibliitem[reference]{\langle label \rangle} \langle entry \rangle
((\bibitem... の繰り返し))
\end{thebibliography}
```

参考文献表での文献の番号のつけ方は文書クラスに依存します。(`thebibliography` 環境の定義は各ドキュメントクラスファイルで行われます。)例えば、文書クラスが `article` の場合には、番号は “[1]”, “[2]” ... という形式で出力されますが、この “[”, “]” の中の番号のうち (実際に出力したときに) 最も幅が広がるものを `\begin{thebibliography}` の直後の `\langle longest-label \rangle` のところに書き込みます。(この `\langle longest-label \rangle` は参考文献の番号を書き込むスペースの大きさを決めるのに用いられます。)また、“[*reference*] ” の部分を与えた場合、その項目の “番号” は `\langle reference \rangle` として与えた文字列になります。`\langle label \rangle` はその文献を引用するときに使う “ラベル” で、`\langle entry \rangle` は文献の記述です。

一方、文献の引用は、`\cite` コマンドで行います。

- `\cite{\langle label_1 \rangle, \langle label_2 \rangle, \dots, \langle label_n \rangle}`: n 個のラベル $\langle label_1 \rangle$, $\langle label_2 \rangle$, ..., $\langle label_n \rangle$ の各々に対応する文献の番号を並べて出力します。
- `\cite[option]{\langle label \rangle}`: $\langle label \rangle$ に対応する文献の番号と `\langle option \rangle` を並べて出力します⁶²⁾。

⁶²⁾この場合にも、 $\langle label \rangle$ 部分に複数のラベルをコンマで区切ったものを用いることができます。

ただし、 $\langle label \rangle$ または $\langle label_k \rangle$ ($1 \leq k \leq n$) は `\bibitem` のパラメータとして与えたラベル文字列で、 $\langle option \rangle$ は文字列です。

本稿の末尾の参考文献表は

```
\begin{thebibliography}{9}
  \item[] まず, ‘‘バイブル’’的なものを挙げます.
  \bibitem{Knuth}
    Donald E.~Knuth ( 鷲谷好輝訳 ),
    ‘‘改訂新版 \TeX{}ブック’’,
    アスキー出版局 ( 1992 ).
  \bibitem{Lamport}
    Leslie Lamport ,
    ‘‘\textit{\LaTeX: A Document Preparation System}’’ (2nd ed.) ,
    Addison-Wesley ( 1994 ).
  %% 中略
\end{thebibliography}
```

という記述で作成したものです⁶³⁾。したがって、本稿において

`\LaTeX`でのマクロ作成を本格的に試みるのならば、基本的な文献（例えば、`\cite{Knuth,Lamport}`）を手元において常に参照できるようにしておくといでしょう。特に、`\cite[Appendix~D]{Knuth}`の内容は難しいのですが、それだけに有用な情報を含んでいます。

のように記述すると、

\LaTeX でのマクロ作成を本格的に試みるのならば、基本的な文献（例えば、 $[1, 2]$ ）を手元において常に参照できるようにしておくといでしょう。特に、 $[1, \text{Appendix D}]$ の内容は難しいのですが、それだけに有用な情報を含んでいます。

のように出力されます。

また、 \LaTeX は文献の引用（`\cite`の処理）に関しても相互参照と同じ機構で処理します。したがって、引用を行っている場合には、少なくとも2回のコンパイルを必要とします。なお、文献の引用に関する警告メッセージは相互参照に関するメッセージと共通のものが多く、対処法とともに §15 で説明しています。

18. 著者名・タイトルの表示 / 概要の表示

レポート・論文などに限らず、多くの文書には文書のタイトルと著者名が表示してあります。実際、 \LaTeX にはタイトルを出力するためのコマンドも用意してあります。

\LaTeX ではタイトルなどを出力するには次のようにします。

⁶³⁾なお、この記述では `\item[]` を用いて“参考文献ではない記述”を書き込んでいることにも注意するとよいでしょう。実際、`thebibliography` 環境は一種の箇条書きの環境なので、(箇条書きの)項目を `\item` を用いて記述することができます。ただし、単に `\item` を用いると“デフォルトの見出し記号”（ここでは文献番号）がついてしまうので、`\item` の後に `[]` を置いて見出し記号を空文字列にしています。

- (1) `\title{<title>}`, `\author{<authors>}` (`<title>` はタイトル, `<authors>` は著者リスト) という記述をプリアンプルに入れます.
- (2) 必要があれば, `\date{<date>}` (`<date>` は文書の作成日) という記述をプリアンプルに入れます.
- (3) `\maketitle` という記述を, (`\begin{document}` 以降の) タイトルを出力したいところに入れます.

なお, `\title`, `\author`, `\date` をプリアンプルに記述する必要はなく, これらは `\maketitle` の前であればどこに記述しても構いません⁶⁴⁾. また, `\date` コマンドを用いない場合には, 文書の日付としてソースファイルをコンパイルした日の日付 (`\today` コマンドで与えられます) が用いられます. 一方, 著者が複数いる場合には, 個々の著者名を `\and` で区切ります. 著者の所属などを脚注にしたい場合には, `\thanks` コマンドを `\thanks{<affiliation>}` (`<affiliation>` は著者の所属など) のように用いてください. `\title` や `\author` のパラメータの中では, 単に `\footnote` を用いると脚注が正しく出力されないことがあります.

また, 文書クラスとして `article` などを用いている場合には, 文書の概要は `abstract` 環境を用いて出力することができます⁶⁵⁾. これは,

```
\begin{abstract}
(( 概要として記述する内容 ))
\end{abstract}
```

のように用います. なお, `\documentclass` のオプションに “`titlepage`” を加えると, `\maketitle` によるタイトルと `abstract` 環境による概要は独立したページに出力されます.

注意 18.1

(`\documentclass` に `titlepage` オプションを用いていないときに) 2 段組にした場合, `abstract` 環境による概要は (下記の例のような凝った記述をしなければ) 2 段組部分に置かれます. 一方, 文書の先頭 (`\begin{document}` の直後) で

```
\twocolumn[
  \maketitle
  \begin{abstract}
    (( 概要の記述 ))
  \end{abstract}]
```

のような記述を行うと, 概要を 1 段組部分に置くことができます⁶⁶⁾.

⁶⁴⁾ プリアンプルに記述した方が管理しやすいと思いますが, これは筆者の趣味です.

⁶⁵⁾ 書籍の場合には “概要” というよりも “前書き” (序文) ということになることが多いので, `book.cls` のような “書籍用” クラスファイルでは `abstract` 環境を定義していないことがあります.

⁶⁶⁾ なお, 今の例のように `\twocolumn` のオプション部分に入れた `abstract` 環境などは, `\documentclass` の `titlepage` オプションの有無によらず本文部分の第 1 ページの上部に出力されます.

19. ページスタイル・2 段組

19.1. ページスタイルの指定

実際に、文書クラスとして `article` または `jarticle` を用いた \LaTeX 文書进行处理して印刷またはプレビューしてみるとわかるように、(プリアンブルで特に何も指定しなければ) ページの下端の中央にページ番号が表示されていて、ページの上端には何も書かれていない (いきなり本文が始まっている) はずです。一方、理工系書籍の類ではページの上端に各節の見出し文字列などが表示されていることがあります。そのようなページのヘッダ (ページの上端に記述される内容)、フッタ (ページの下端に記述される内容) は “ページスタイル” によって指定されています。例えば、先程述べた `article` または `jarticle` クラスでのデフォルトのページスタイルは `plain` スタイルです。

\LaTeX では、次の 4 種のスタイルを用意しています。(文書クラスによってはさらに多くのスタイルを用意していることもあります。)

(1) `plain`:

ヘッダには何も出力しません。フッタには中央にページ番号を出力します。

(2) `empty`:

ヘッダ、フッタともに何も出力しません。

(3) `headings`:

ヘッダには `\section` (または `\chapter` など) の見出し (目次用の短い見出しを与えたときにはその短い見出し) とページ番号が出力されます。フッタには何も出力されません。

(4) `myheadings`:

ヘッダにはページ番号とヘッダに入れる文字列 (これはユーザが与えることができます) が出力されます。フッタには何も出力されません。

ただし、ページスタイルの定義は、文書クラスに依存することに注意してください。(上記の説明は `article` などの欧文用の標準文書クラスに基づいています。) また、これらのページスタイルを用いるときには、

```
\pagestyle{<style>}
```

(`<style>` は使用するスタイル) のように記述します。`\pagestyle` コマンドを文書の途中で用いると、ページスタイルを変更することができます。(文書全体にわたって用いるページスタイルの指定はプリアンブルで行うとよいでしょう。) また、(章見出しが出力されるページのように) 1 ページだけ一時的にページスタイルを変更したい場合には `\thispagestyle` コマンドを

```
\thispagestyle{<style>}
```

(`<style>` は使用するスタイル) のように使います。`\thispagestyle` コマンドを用いると、`\thispagestyle{<style>}` を用いたページのスタイルを `<style>` スタイルにし、次のページからは元のスタイルに戻します。なお、`myheadings` スタイルを用いたときに

ヘッダに出力する文字列は、`\markright`、`\markboth` コマンドを用いて指定します。これらは、

```
\markright{<odd-head>}
\markboth{<even-head>}{<odd-head>}
```

(ただし、`<odd-head>`、`<even-head>` はそれぞれ奇数ページのヘッダで用いる文字列、偶数ページのヘッダで用いる文字列) のように用います。ただし、奇数ページと偶数ページを区別しない場合 (例えば、文書クラスとして `article` を `twoside` オプションを指定せずに用いているような場合) には、ヘッダには常に奇数ページ用の文字列が用いられます。

19.2. 2 段組・多段組

さて、§4 では、`\documentclass` コマンドに “`twocolumn`” オプションをつけて用いることで、2 段組にすることができる、という話をしました。ただし、`\documentclass` のオプションによる指定は文書全体にわたって有効な指定です。文書の特定の部分だけを 2 段組 (あるいは 1 段組) にするには、`\twocolumn`、`\onecolumn` というコマンドを用います。これらの意味は次のとおりです。

- `\onecolumn`: 改ページを行い、このコマンド以降のページを 1 段組で出力します。
- `\twocolumn`: 改ページを行い、このコマンド以降のページを 2 段組で出力します。

簡単に言えば、これらのコマンドで 1 段組と 2 段組を切り換えることができるわけです。ただし、切り換えの際に必ず改ページが行われることに注意してください。また、`\twocolumn` に

```
\twocolumn[<one-column material>]
```

のようにオプションをつけると、`<one-column material>` 部分を 1 段組にしたものを (`\twocolumn` の後の) 最初のページの上部に配置します⁶⁷⁾。

一方、3 段組以上の多段組を行うには、`multicol` パッケージを用います。プリアンブルに

```
\usepackage{multicol}
```

という記述を入れると、`multicols` 環境を使えるようになります。この環境は、

```
\begin{multicols}{<number>}
(( <number> 段組にするテキスト ))
\end{multicols}
```

(ただし、`<number>` は 2 以上 10 以下の整数) のように用います。なお、`multicols` 環境と `twocolumn` オプション (または `\twocolumn` コマンド) を併用しないでください (併用した場合、意図しない出力が得られます)。

⁶⁷⁾`multicols` 環境でも (第 1) オプション引数を用いてこの例と同様の 1 段組部分を作ることができます。

また, `multicols` 環境で多段組を行う場合には, `multicols` 環境の前後では改ページは行われません。したがって, 1 段組部分と 2 段組部分の切り換えの際にいちいち改ページしたくない場合には, 2 段組にも `multicols` 環境を用いる, ということができます。

20. 空白の調整

§3 では, ソースファイル中の空白文字は何個連続していてもただ 1 個の空白文字として扱われる, ということを述べました。しかし, “解答欄” のようなところではまとまった大きさの空白が必要になります。ここでは, そういった空白の調節に用いるコマンドについて説明します。

まず, 単に空白を出力するには, 文字 `~` または `_` (`\` + 空白文字) というコマンドを用います。実際, 次の例のようになります。

— INPUT —

```
(~), (~~), (~~~), (~~~~),
(\ ), (\ \ ), (\ \ \ ), (\ \ \ \ )
```

— OUTPUT —

```
( ), ( ), ( ), ( ), ( ), ( ), ( ), ( )
```

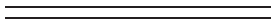
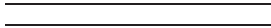

なお, `_` の箇所では改行できますが `~` の箇所での改行は禁止されています。

次に, 行方向の小さな空白を入れるコマンドを挙げます。

<code>this\thinspace space</code>	→	this space
<code>this\enspace space</code>	→	this space
<code>this\quad space</code>	→	this space
<code>this\qqquad space</code>	→	this space
<code>this\negthinspace space</code>	→	thisspace

これらのコマンドのうち, `\thinspace` は `\,` と記述することもできます。

一方, 垂直方向の小さな空白は次の表のコマンドで作れます。

<code>\hrule\smallskip\hrule</code>	→	
<code>\hrule\medskip\hrule</code>	→	
<code>\hrule\bigskip\hrule</code>	→	

なお, この例で用いている `\hrule` というのは水平方向の罫線を出力するコマンドです。

また, 大きさを指定した空白を出力するには次のコマンドが使えます。

- 水平方向: `\hspace{<length>}`
- 垂直方向: `\vspace{<length>}`

ここで, `<length>` は空白の大きさです。例えば, `\hspace` を用いると次のようになります。(`\vspace` は空白を出力する方向が異なるだけなので, `\vspace` の出力例は省略します。)

— INPUT —

```
patapata\hspace{3em}oyoyo\\
patapata\hspace{5em}oyoyo\\
patapata\hspace{10em}oyoyo
```

— OUTPUT —

```
patapata      oyoyo
patapata      oyoyo
patapata      oyoyo
```

この例では、長さの単位として `em` (`1em` は概ね大文字の `M` の幅⁶⁸⁾) を用いていますが、`cm`、`mm` といった単位を用いても構いません。ただし、`\hspace` を行の左端で用いる（または `\vspace` をページの先頭で用いる）と空白が出力されません。行の左端（またはページの先頭）にも空白を出力したい場合には、`\hspace*{...}`（または `\vspace*{...}`）のように `*` を添えて用います。

注意 20.1

具体的な寸法がわかっている空白は `\hspace` などを用いて作成できますが、空白の大きさが具体的な寸法ではなく“何らかの文字列と同じ”という場合には `\phantom` というコマンドを用いることができます。例えば、

```
patapataoyoyoarere
patapata\phantom{oyoyo}arere
```

という記述からは

```
patapataoyoyoarere
patapata      arere
```

という出力が得られます。この例からわかるように、`` (`⟨text⟩` は文字列) は `⟨text⟩` と同じ幅（および高さ）の寸法の空白を作成します。また、`\phantom` と似たコマンドとして `\hphantom`、`\vphantom` があります。これらの意味は次のとおりです（`⟨text⟩` は文字列です）。

- `\hphantom{⟨text⟩}`: `⟨text⟩` と同じ幅の空白を作成します。ただし、高さを無視します。
- `\vphantom{⟨text⟩}`: `⟨text⟩` と同じ高さ（および深さ⁶⁹⁾）をもった幅のない支柱を作成します。

例えば、ディスプレイ数式の中では、`` と `\hphantom{\int}` は両方とも積分記号 (`\int`) と同じ幅の空白を作成しますが、`` の方は積分記号と同じ高さをもつので行間隔に影響を与えます。

⁶⁸⁾これはあくまでも目安にすぎません。

⁶⁹⁾文字などについて、ベースラインから下に出ている部分の高さを“深さ” (depth) といいます。

最後に，“なるべく大きな空白”を作るコマンドを紹介します．

- 水平方向: `\hfill`
- 垂直方向: `\vfill`

やはり，水平方向について出力例を挙げます．

— INPUT —

```
patapata\hfill oyoyo\
patapata\hfill oyoyo\hfill arere\
patapata\hfill oyoyo\hfill arere\hfill ottotto
```

— OUTPUT —

patapata			oyoyo
patapata		oyoyo	arere
patapata	oyoyo	arere	ottotto

21. 改行・改ページの起こりやすさの調節

§3 では，改行したい位置には `\` を書き込むと述べましたが，ここでは，改行（または改ページ）の調節に関するコマンドのうち `\` 以外のものを紹介します．

まず，改行・改ページのどちらについても使えるものを挙げます⁷⁰⁾．

- `\break`: このコマンドのところで改行・改ページを行います．
- `\nobreak`: このコマンドのところでの改行・改ページを禁止します．
- `\allowbreak`: このコマンドのところでの改行・改ページを許可します．

これらは，段落の中で用いた場合には改行の調節を行い，段落の間（例えば，空白行の直後）で用いた場合には改ページの調節を行います．また，`\allowbreak` は改行・改ページを促進も抑制もしませんが，普通は改行が起こらないような位置に `\allowbreak` を入れると，その位置も改行位置の候補に加えられます．

次に，改行の調節を行うコマンドを挙げます（“`[<number>]`”の部分は省略できます）．

- `\linebreak[<number>]`
- `\nolinebreak[<number>]`

ここで，`<number>` は 0 以上 4 以下の整数です．これらのコマンドは，これらのコマンドの位置での改行の調節を行います．`<number>` が 0 の場合，すなわち `\linebreak[0]`，`\nolinebreak[0]` はともに `\allowbreak` と同じ働きをします．また，`<number>` が大きくなるほど，`\linebreak` は改行を促進し，`\nolinebreak` は改行を抑制します．

⁷⁰⁾— 一般に，文書作成時には“分割してはいけない箇所”を指定するのが無難です．行分割・ページ分割を強制的に行うのは，文書の内容が確定して体裁の調整を行うだけになった段階であるのがよいでしょう．実際，ページ分割を強制的に行った箇所の前に文章を追加すると，強制的に行っていたページ分割の位置が（改ページ位置として）不適切になることがあります．（もちろん，“章”の変わり目などでの改ページには問題はありません．）

特に $\langle number \rangle$ が 4 の場合を考えると、`\linebreak[4]` は `\break` と同じ働きをし、`\nolinebreak[4]` は `\nobreak` と同じ働きをします。`[\langle number \rangle]` を与えない場合には、 $\langle number \rangle$ の値として 4 が用いられます。なお、 $\langle number \rangle$ の値として 5 以上または負の値を与えても構いませんが、その場合は 4 を与えたものとして扱われます。

Note: `\linebreak`、`\nolinebreak` は、段落の間で用いるとエラーになります。

次に、改ページの調節を行うコマンドを挙げます (“`[\langle number \rangle]`” の部分は省略できます)。

- `\pagebreak[\langle number \rangle]`
- `\nopagebreak[\langle number \rangle]`

ここで、 $\langle number \rangle$ は 0 以上 4 以下の整数です。これらのコマンドを段落の間で用いた場合にはこれらのコマンドの位置での改ページの調節を行います。一方、段落の中で用いた場合には、これらのコマンドを用いた箇所を含む行（出力結果における行です）とその次の行との間での改ページの調節を行います。また、 $\langle number \rangle$ が 0 の場合には `\pagebreak`、`\nopagebreak` はともに改ページを単に許可するだけで、促進も抑制も行いません。また、 $\langle number \rangle$ が大きくなるほど、`\pagebreak` は改ページを促進し、`\nopagebreak` は改ページを抑制します。特に $\langle number \rangle$ が 4 の場合を考えると、`\pagebreak[4]` は `\break` とほぼ同じ働きをし、`\nopagebreak[4]` は `\nobreak` とほぼ同じ働きをします。`[\langle number \rangle]` を与えない場合、 $\langle number \rangle$ の値として 4 が用いられます。なお、 $\langle number \rangle$ の値として 5 以上または負の値を与えても構いませんが、その場合は 4 を与えたものとして扱われます。

最後に、改ページを行う（強制する）コマンド（改ページ専用のもの）を挙げます。

- `\newpage`
- `\clearpage`
- `\cleardoublepage`

これらのうち、2 段組の際に `\newpage` を 1 段目で用いると、改ページではなく改段を行います（すなわち、1 段目を終了し 2 段目を始めます）。また、`\documentclass` のオプションに `twoside` が用いられているときに奇数ページで `\cleardoublepage` を用いると、改ページが 2 回行われて、次のページが奇数ページになるようにします。`\clearpage` は（1 段組であるか 2 段組であるかによらず）改ページを行います。ただし、`\clearpage`、`\cleardoublepage` の際には、未出力のフロート（`table`、`figure` 環境で実現されているような移動可能な項目）の出力などが伴います。

Note: これらのコマンドを段落の中で用いた場合には、その段落を `\newpage` などを用いたところで終了させてから改ページを行います。また、これらのコマンドを `\newpage\newpage` のように 2 個（以上）並べた場合、2 個目以降は無視されます。

22. 数式 (1) [L^AT_EX における数式]

前節までで（さほど大規模ではない）文章を書くための基本的なことの説明が大体終わりましたので、ここからしばらくは数式について説明します。まず、L^AT_EX の“数

式”とは、(基本的には)次の(1)–(3)のいずれかにあてはまる部分です。

- (1) 文字\$で挟まれた部分、\(&\)の間の部分
- (2) \[と\]の間の部分、文字列\$\$で挟まれた部分
- (3) math, displaymath, equation, eqnarray, eqnarray*の各環境の中

これらのうち、(1)にあてはまる部分とmath環境は段落の中に埋め込んだ数式を出力し、(2)にあてはまる部分と、(3)の残り四つの環境はディスプレイ数式を出力(すなわち、数式だけを独立させて出力)します。(amsmathパッケージを用いると(3)で挙げたもの以外のディスプレイ数式関係の環境を利用できますが、amsmathパッケージについてもここでは深入りしません。)ただし、“\$\$で挟まれた部分”というのは、(1)の“\$で挟まれた部分”と紛らわしくなり、また、L^AT_EXが用意しているディスプレイ数式の体裁のカスタマイズ機構からも逸脱します⁷¹⁾。したがって、ディスプレイ数式を\$\$で挟んだ形式で記述することは推奨しません。

ここで、数式の例を挙げます。

— INPUT —

```
数式の例です：$PV=nRT$, $ PV = nRT $, \(< PV = nRT \&\) \&
ここから \begin{math} PV = nRT \end{math} ここまではmath環境です。
\begin{displaymath}
PV = nRT
\end{displaymath}
改行されていますね。
\[ PV = nRT \]
これも displaymath 環境と同様です。\\
equation 環境には番号がつきます。
\begin{equation}
PV = nRT
\end{equation}
```

— OUTPUT —

```
数式の例です：PV = nRT, PV = nRT, PV = nRT
ここから PV = nRT ここまではmath環境です。
```

$$PV = nRT$$

改行されていますね。

$$PV = nRT$$

これも displaymath 環境と同様です。

⁷¹⁾例えば、文書クラスとして jarticle を用いている場合には、\documentclass のオプションとして “fleqn” を指定すると (equation 環境などの、L^AT_EX が提供する環境・コマンドによる) ディスプレイ数式を左寄せにして出力することができます。しかし、\$\$で挟んだ形式によるディスプレイ数式は “fleqn” オプションの指定を無視します。(\$\$で挟んだ形式でディスプレイ数式を出力する、というのは T_EX に組み込まれている (基礎的な) 機能です。)

equation 環境には番号がつきます．

$$PV = nRT \quad (1)$$

この出力例の最初の行からわかるように，数式においては文字（または記号）の間に空白を入れても無視されます．数式での文字や記号の間の空白の大きさは $\text{T}_{\text{E}}\text{X}$ が自動的に調整します．ただし，手動で調節するためのコマンドもあります（それらは §33 で扱います）．また，ここでは `eqnarray` 環境については扱っていませんが，これについては §31 で扱います．

23. 数式 (2) [添字・分数]

ここでは， x^2 ， x^3 ， a_1 ， a_2 のような上下の添字や， $\frac{2}{3}$ ， $\frac{n}{m}$ のような分数について説明します．なお，ここから先の数式関係の説明での入力例では，文字 \$ などの数式への切り換えを行うための記述は省略しています．

23.1. 添字

数式中の文字など (`\langle item \rangle` とします) に上添字 `\langle superscript \rangle` または下添字 `\langle subscript \rangle` をつけるには，次のように記述します．

- `\langle item \rangle^{\langle superscript \rangle}`
- `\langle item \rangle_{\langle subscript \rangle}`

例えば，数式中で `x^2`，`x^3` のように記述すると x^2 ， x^3 が出力され，`a_1`，`a_2` のように記述すれば a_1 ， a_2 が出力されます．なお，添字がただ 1 文字の場合には添字を `{}`，`}` で囲まなくても構いません．したがって，今の例は `x^2`，`x^3` あるいは `a_1`，`a_2` のように記述しても同じです．ただし，添字が 2 文字以上のときには `{}`，`}` を省略できないということに注意してください．実際，`a_{10}` という入力に対しては a_{10} という出力が得られますが，`a_10` に対しては a_10 という出力が得られます．また，上付きの添字と下付きの添字を両方つけて x_2^3 のように出力する場合には，`x_2^3` または `x^3_2` のように入力します（上添字と下添字はどちらを先に記述しても構いません）．

一方，添字の中に添字をもつような数式を使うこともできます．例えば， x^{y^z} ， a_{b_c} は，`x^{\{y^z\}}`，`a_{\{b_c\}}` のように記述すると出力できます．ここで，（上添字と下添字を両方用いる場合と混同して）`x^y^z` のように入力すると，

! Double superscript.

というエラーメッセージが現れます．実際，`x^y^z` という記述は `\{x^y\}^z` と `x^{\{y^z\}}` のどちらであるか不明です．下添字の場合についても同様です．ちなみに，`\{x^y\}^z` に対しては x^{yz} という出力が得られ，これは上記の `x^{\{y^z\}}` に対する出力とは異なっています．なお，プライム記号は `\prime` を上添字として用いると出力できますが，簡単に `'` を用いて出力することもできます．例えば，`f^{\prime}` と `f'` はともに f' という出力を与え，`f_0^{\prime\prime}` と `f_0''` はともに f_0'' という出力を与えます．

以下，添字に関して細かいことを述べます．

(1) 左側につける添字

TeX には左側に添字をつけるための書式はありませんが、 ${}_nC_m$ のようなものは、`{_n \mathrm{C}_m}` と記述すれば出力できます。(ここで用いた `\mathrm` は数式中で直立のローマン体を用いるためのコマンドです。)これからわかるように、添字は全く空であるような数式に対してつけても構いません。また、この例では `_n \mathrm{C}_m` と書いても構わないのですが、今述べたように `{}` を補っておくのが無難です。実際、 ${}_a{}_1{}_nC_m{}_a{}_2$ という記述を行うとき、`_a_1 _n \mathrm{C}_m _a_2` と記述すれば問題はありますが、`_a_1 _n \mathrm{C}_m _a_2` のように記述すると、`_a_1 _n` の部分が TeX にとって意味不明になります。

(2) 上添字と下添字の位置関係の調節

一部の文字や記号（例えば P のようなもの）に上添字と下添字を両方つけるとき、普通に P_{-1}^{-1} とすると P_1^1 のようになり、上下の添字のずれが目立ちます。一方、(1) で用いた、“{} に添字をつける” という方法を用いて、 P_{-1}^{-1} のように記述すると、 P_1^1 という出力が得られます。また、{} に添字をつけるというのは、添字を揃える場合だけでなく、 A_{ij}^{kl} のように添字をずらす場合にも利用できます⁷²⁾。実際、これは、 $A_{\{ij\}}^{\{kl\}}$ という入力から得られたものです。

(3) 背が低い文字に下添字をつける場合

ほとんどすべてのギリシャ文字や p のような文字に下添字だけをつけると, p_1 のようになり, あまり添字らしくありません. そのような場合, 空の上添字を用いると多少添字らしくすることができます. 例えば, p_{-1} , p_{-1}^{\wedge} という入力からは, それぞれ p_1 , p_1 のような出力が得られます.

(4) プライム記号について

単に、 f_0'' のように出力する場合には、`f_0''` と記述すればよく、`\prime` コマンドを用いることはあまりないのですが、`\prime` を用いた出力法も知っておいてください。実際、 A^* という出力を得たい場合、`A^{*\prime}` という入力では正しくこの出力を与えますが、`A^*` という入力はエラーになります。これは、文字 $'$ の数式中での意味によります。 \LaTeX (というよりもむしろ plain TeX) は、数式中に文字 $'$ の列 (途中に空白があってはいけません) が現れると、それを `\prime\prime\prime\prime\prime` (`\prime` は文字 $'$ の個数と同じ個数並びます) として扱うように定義されています。したがって、`A^*` という記述は、`A^{*\prime}` として扱われ “Double superscript” のエラーを引き起こします。なお、これと同じ理由で、 f'' を出力したいときに隣り合う $'$ の間に空白を入れて `f''` と⁷³⁾記述するとエラーになります。

23.2. 分数

次に分数について説明します．分数は，`\frac` コマンドを用いて出力できます．`\frac` の書式は次のとおりです（ただし，`\langle numerator \rangle` は分子，`\langle denominator \rangle` は分母です）．

$$\frac{\langle \text{numerator} \rangle}{\langle \text{denominator} \rangle}$$

⁷²⁾ただし、“ x_1 の 2 乗” は単に “ x_1^2 ” と記述すればよく、これをわざわざ “ x_1^2 ” のように添字をずらすのは不適切です。添字をずらすという処理は“意味上の必然性”がある箇所に限り行うものです。

⁷³⁾記号“ ”は空白文字を明示するために用いている記号です。

例えば、この節の最初に示した $\frac{2}{3}$, $\frac{n}{m}$ は、`\frac{2}{3}`, `\frac{n}{m}` という入力から得られたものです。分数の分母・分子に再び分数を入れて、

$$\frac{\frac{a}{b}}{\frac{x}{y}} = \frac{ay}{bx}$$

すなわち

$$\frac{\frac{a}{b}}{\frac{x}{y}} = \frac{ay}{bx}$$

という出力を得ることもできます。ただし、あまり複雑な分数は読みづらいので、必要に応じて a/b という記法を用いた方がよいでしょう。また、分母に分数を入れるということを繰り返すと、次のように連分数を出力することもできます。

— INPUT —

```
\displaystyle\frac{1}{\displaystyle 2 +
\frac{1}{\displaystyle 3 + \frac{1}{4 + \cdots}}}
```

— OUTPUT —

$$\frac{1}{2 + \frac{1}{3 + \frac{1}{4 + \cdots}}}$$

この例では、`\displaystyle` というコマンドを用いていますが、これは“ディスプレイ数式のスタイルで出力する”という指定を行うコマンドです (§29 参照)。また、`\cdots` は (“数式の軸⁷⁴”の高さに) 横に 3 個並んだ点を出力するコマンドです。

注意 23.1

分数は `\frac` コマンドで出力できるわけですが、分母、分子の数式の幅がともに小さい場合、単に `\frac{2}{3}`, `\frac{n}{m}` のように入力すると、出力は $\frac{2}{3}$, $\frac{n}{m}$ のようになり、分数の横線が短いように感じるかもしれません。そのような場合には、分母・分子のうちで幅が広い方の両側に適当な幅の空白を追加すると、横線の長さを長くすることができます。実際、`\frac{n}{\;m\;}` (`\;` は数式中で $5/18\text{em}$ の (状況によって $10/18\text{em}$ まで伸びる) 空白を入れるコマンドです) のように記述すると、 $\frac{n}{m}$ のように出力されます。ここで、`\;` のようなコマンドをいちいち記入するのが面倒なら、`\frac` を再定義して、

```
\def\frac#1#2{\begingroup\;#1\;\endgroup\over\;#2\;}
```

のようにしてもよいでしょう。

⁷⁴ 数式の縦方向の基準となる位置 (の一つ) で、分数の横線の位置になります。

24. 数式 (3) [根号・省略記号]

ここでは、 $\sqrt{2}$ 、 $\sqrt[3]{4}$ のような根号や、 x_1, x_2, \dots, x_n のような記述に用いる省略記号 (3 個並んだ点) について説明します。

24.1. 根号

根号は `\sqrt` コマンドによって出力できます。実際、先程の $\sqrt{2}$ 、 $\sqrt[3]{4}$ は、それぞれ `\sqrt{2}`、`\sqrt[3]{4}` という記述から得られたものです。なお、`\sqrt` コマンドの書式は次のとおりです。(“[*exponent*]” の部分は省略できます。)

`\sqrt[exponent]{formula}`

ここで、*formula* は根号の中身にする式です。また、[*exponent*] ($\sqrt[3]{4}$ の 3 のところです) が与えられたときには、根号の左肩に *exponent* を出力します。

この `\sqrt` コマンドは、次の例のように根号の中身の大きさに応じて根号の大きさを変化させます⁷⁵⁾。

— INPUT —

`\sqrt{2 + \sqrt{2 + \sqrt{2 + \sqrt{2 + \sqrt{2}}}}}`

— OUTPUT —

$$\sqrt{2 + \sqrt{2 + \sqrt{2 + \sqrt{2 + \sqrt{2}}}}}$$

一方、根号の大きさを、根号の中身の大きさにしたがって決めると、`\sqrt{g} + \sqrt{h}` という入力に対する出力は $\sqrt{g} + \sqrt{h}$ のように根号の大きさが不揃いになります。この場合に根号の大きさが揃うように変更するには、`\mathstrut` というコマンドを用いて、

`\sqrt{\mathstrut g} + \sqrt{\mathstrut h}`

のように入力します。そうすると、 $\sqrt{g} + \sqrt{h}$ という出力が得られます⁷⁶⁾。なお、ここで用いた `\mathstrut` というのは、文字 “(” と同じ高さで深さ⁷⁷⁾をもち、幅が 0 であるような “見えない支柱” です。

また、`\sqrt` にオプションをつけて $\sqrt[p]{p}$ のような数式 (これは `\sqrt[p]{p}` という入力から得られたものです) を出力する場合、根号の左肩の文字の位置を調節して $\sqrt[p]{p}$

⁷⁵⁾根号の中身がかなり大きくなると、根号の左側の部分に垂直な線分が用いられるようになります。これを好まない人もいるようですが、(教育的配慮を真に必要とする場合を除き) 垂直な線分が用いられるようになる前に $(\dots)^{1/2}$ の形式に書き換えるのが賢明でしょう。

⁷⁶⁾ただし、このような場合に根号の高さを揃えるか否かはほとんど趣味の問題です。

⁷⁷⁾TeX では、文字などの大きさについてベースライン (行の基準線) から上に出ている部分の高さを “高さ” (height) といい、ベースラインから下に出ている部分の高さを “深さ” (depth) といいます。一方、“幅” (width) については普通の意味の幅です。

のように出力したいことがあります。これは、

```
\sqrt[\raise.5ex\hbox{$\scriptscriptstyle p$}]{p}
\kern.25ex]{p}
```

のようにして出力しました。ここで用いた `\kern`、`\raise` は文字（正確にはボックス）の位置を移動させるコマンドで、`\scriptscriptstyle` は数式を“添字の中の添字”のスタイルにするというコマンドです (§29 参照)。なお、`amsmath` パッケージを使用すると（プリアンブルに `\usepackage{amsmath}` という記述を入れると）、

```
\sqrt[\leftroot{1}\uproot{4}]{p}
```

という記述から $\sqrt[p]{p}$ と同様の出力が得られます。

24.2. 省略記号

次に、 a, b, \dots, z のような省略記号について説明します。省略記号として用意されているのは、`\cdots`、`\ldots`（横に並んだ点を出力）、`\vdots`（縦に並んだ点を出力）、`\ddots`（斜めに並んだ点を出力）の 4 個のコマンドで、これらは次のような出力を与えます。

```
\cdots → …   \ldots → …   \vdots → ⋮   \ddots → ⋱
```

ここで、`\cdots` は“数式の軸”と呼ばれる位置（分数の横線の位置）上に並んだ点を出力するのに対し、`\ldots` はベースラインの位置に並んだ点を出力することに注意してください。また、`\ddots`、`\vdots` は行列などの記述を行うときに用いられます。

なお、`\cdots`、`\ldots` の使い分けは、次のように決めるのが無難です。

- (1) $a_1 + a_2 + \dots + a_n$ 、 $b_1 < b_2 < \dots < b_n$ の場合のような、演算子（この例では + 記号または < 記号）の間の省略記号には `\cdots` を用います。
- (2) (1) 以外の（横方向の）省略記号（例えば、 x_1, x_2, \dots, x_n 、 $(x+1)(x+2)\dots(x+n)$ のような場合）には `\ldots` を用います。

数学的には、 $(x+1)(x+2)\dots(x+n)$ の場合の “)” とその次の “(” との間には乗算の演算子が入っているので、この場合には `\cdots` にするのも許容範囲です⁷⁸⁾。

注意 24.1

ファイル `fontmath.ltx` を読むと、

```
\def\ddots{\mathinner{\mkern1mu\raise7\p@
\vdots{\kern7\p@\hbox{.}}\mkern2mu
\raise4\p@\hbox{.}}\mkern2mu\raise\p@\hbox{.}}\mkern1mu}}
```

という定義がありますが、これが `\ddots` の定義です。そこで、この定義の中の 3 個の点の順序を入れ換えて、

⁷⁸⁾日本の印刷・出版業界には数式の記述に関する十分な見識（および十分な数学的素養）をもった人間は少ないので、省略記号を何でも `\cdots`（というよりも、和文文字の“…”）で済ませてしまうという不見識がまかり通っていますが、本来、`\ldots` と `\cdots` は厳格に使い分けるべきものです。

```
\makeatletter
\def\varddts{\mathinner{\mkern1mu
  \raise\p@\hbox{.}\mkern2mu\raise4\p@\hbox{.}\mkern2mu
  \raise7\p@\vbox{\kern7\p@\hbox{.}}\mkern1mu}}
\makeatother
```

のように `\varddts` を定義してみます。このとき、`\varddts` と入力すると \cdots という出力が得られます。

25. 数式 (4) [演算子]

数式で用いられる演算子 (+, − のようなもの) のうち, +, = などは数式中にそのまま書けばよいのですが, 多くの記号はコマンドを用いて記述します。ここでは, そのような演算子を出力するコマンドを紹介します。

Note: ($\text{T}_{\text{E}}\text{X}$ ソースファイルでない) 一般のテキストファイルにおいては, 数式用の記号類を和文文字を用いて書くことがあります, $\text{T}_{\text{E}}\text{X}$ 文書では数式中にそのような和文文字の記号類を入れないでください。処理系 (またはドキュメントクラスファイルなどの中での設定) によってはエラーになりますし, エラーにならなくても記号類とその前後の文字との間の空白の大きさが不自然なものになります。ただし, 真にやむを得ない場合には, 数式中で用いる和文文字を `\mbox` に入れて用いるようにする (例えば, `\mbox{ }` のように記述する) のが無難です。

25.1. 2 項演算子

まず, 2 項演算子を挙げます。

<code>\triangleleft</code>	→ \triangleleft	<code>\cap</code>	→ \cap	<code>\odot</code>	→ \odot
<code>\triangleright</code>	→ \triangleright	<code>\cup</code>	→ \cup	<code>\oslash</code>	→ \oslash
<code>\bigtriangleup</code>	→ \triangle	<code>\sqcap</code>	→ \sqcap	<code>\otimes</code>	→ \otimes
<code>\bigtriangledown</code>	→ ∇	<code>\sqcup</code>	→ \sqcup	<code>\ominus</code>	→ \ominus
<code>\wedge</code>	→ \wedge	<code>\amalg</code>	→ \amalg	<code>\oplus</code>	→ \oplus
<code>\vee</code>	→ \vee	<code>\times</code>	→ \times	<code>\mp</code>	→ \mp
<code>\circ</code>	→ \circ	<code>\div</code>	→ \div	<code>\pm</code>	→ \pm
<code>\bigcirc</code>	→ \bigcirc	<code>\ast</code>	→ \ast	<code>\wr</code>	→ \wr
<code>\uplus</code>	→ \uplus	<code>\star</code>	→ \star	<code>\cdot</code>	→ \cdot
<code>\setminus</code>	→ \setminus				

これらの記号のうち, \wedge, \vee はそれぞれ `\land, \lor` としても出力できます。

25.2. 関係演算子

次に, 関係演算子を挙げます。

<code>\leq</code>	→ \leq	<code>\subset</code>	→ \subset	<code>\equiv</code>	→ \equiv
<code>\geq</code>	→ \geq	<code>\supset</code>	→ \supset	<code>\cong</code>	→ \cong
<code>\neq</code>	→ \neq	<code>\subseteq</code>	→ \subseteq	<code>\succ</code>	→ \succ
<code>\in</code>	→ \in	<code>\supseteq</code>	→ \supseteq	<code>\prec</code>	→ \prec

```

\notin → ∉ ; \sqsubseteq → ⊆ ; \succeq → ⋮
\ni → ∋ ; \sqsupseteq → ⊇ ; \preceq → ⋮
\gg → ≫ ; \sim → ∼ ; \dashv → ⊥
\ll → ≪ ; \simeq → ≈ ; \vdash → ⊢
\asymp → ∞ ; \mid → | ; \models → ⊨
\approx → ≈ ; \parallel → ∥ ; \bowtie → ⋈
\propto → ∝ ; \perp → ⊥ ; \doteq → ⋮
\not → /

```

これらの記号のうち、 \leq, \geq, \neq, \ni は、それぞれ `\le`, `\ge`, `\ne`, `\owns` としても出力できます。また、`\not` は、定義の上では関係演算子ですが、この記号は実際には他の関係演算子の上に重ねて“否定”の意味を表した記号を作るのに用います。例えば、`\not\equiv` と書くと、 \neq という出力が得られます。

注意 25.1

数式中のコロンは“1:2”のような“比”を表す関係演算子として扱われます。一方、写像の記述“ $f: X \rightarrow Y$ ”におけるコロンは決して“比”の演算子ではなく、むしろ(欧文における)句読点のコロンに近いものです。そのような“数式用句読点”のコロンは `\colon` というコマンドで出力できます⁷⁹⁾。

また、 \LaTeX には2個の記号(など)を積み重ねて新しい関係演算子を作るコマンド `\stackrel` が用意してあります。これは、

```
\stackrel{\langle upper-symbol \rangle}{\langle lower-symbol \rangle}
```

のように用いて、 $\langle upper-symbol \rangle$ を $\langle lower-symbol \rangle$ の上に載せたものを出力します(ただし、 $\langle upper-symbol \rangle$ は添字のスタイルで出力されます)。例えば、`\stackrel{.}{+}` のように記述すると、 $\overset{.}{+}$ のように出力されます。(この記号は、2項演算子として用いることが普通なので、2項演算子として定義した方がよいのですが、ここでは、`\stackrel` の説明を行うのが目的なので細かい修正はしていません。なお、`amssymb` パッケージを用いると、`\dotplus` というコマンドでこの記号を出力できます。)

注意 25.2

例えば、 $X \overset{\times}{\underset{\sigma}{\times}} Y$ のように記述したいときに、

```
X \stackrel{\times}{\underset{\sigma}{\times}} Y
```

のように入力すると、 $X \overset{\times}{\underset{\sigma}{\times}} Y$ のように出力されます。この場合、うまく $X \overset{\times}{\underset{\sigma}{\times}} Y$ という出力を得るためには、`\stackrel` の定義を参考にして、“上下関係を逆にした”コマンドを定義してそれを用います。実際、`\stackrel` はファイル `latex.ltx` の中で、

```
\def\stackrel#1#2{\mathrel{\mathop{\#2}\limits^{\#1}}}
```

⁷⁹⁾ 関係演算子のコロンと句読点のコロンは厳格に使い分けるべきものです。

のように定義されています．そこで，定義中の \sim を $_$ に変えて，

```
\def\varstackrel#1#2{\mathrel{\mathop{#2}\limits_{#1}}}
```

のように定義してみます．このとき，

```
X \varstackrel{\sigma}{\times} Y
```

という入力で $X \times_{\sigma} Y$ が出力されます．ただし，`amsmath` パッケージを用いると，`\stackrel` または今の `\varstackrel` とほぼ同様の意味のコマンド `\overset` または `\underset` を利用できます．

一方，二つの記号 $< =$ を積み重ねて \leq のような記号を作ろうとすると，`\stackrel` (および上記の `\varstackrel`) ではうまくいきません．このような場合には，2 個の記号を積み重ねるための汎用的なコマンドを用意し，それを利用します．例えば，

```
\makeatletter
\def\mathcomposite{%
  \@ifstar
    {\def\@mathcomposite@option{%
      \baselineskip\z@skip\lineskiplimit-\maxdimen}%
      \@mathcomposite}%
    {\let\@mathcomposite@option\offinterlineskip
      \@mathcomposite}}
\def\@mathcomposite{%
  \@ifnextchar[{\@mathcomposite{\@mathcomposite[0]}}
\def\@mathcomposite[#1]#2#3#4{%
  #2{\mathchoice
    {\@mathcomposite@{#1}{#3}{#4}\displaystyle{1}}%
    {\@mathcomposite@{#1}{#3}{#4}\textstyle{1}}%
    {\@mathcomposite@{#1}{#3}{#4}%
      \scriptstyle\defaultscriptratio}%
    {\@mathcomposite@{#1}{#3}{#4}%
      \scriptscriptstyle\defaultscriptscriptratio}}}
\def\@mathcomposite@#1#2#3#4#5{%
  \vcenter{\m@th\@mathcomposite@option
    \dimen@ \f@size\p@\dimen@#1\dimen@\dimen@#5\dimen@
    \divide\dimen@ 18
    \edef\@mathcomposite@skipamount{\the\dimen@}%
    \ialign{\hfil$#4##$\hfil\cr
      #2\cr
      \noalign{\vskip\@mathcomposite@skipamount}%
      #3\cr}}
\makeatother
```

という記述をプリアンブルに入れると，

```
\mathcomposite{\mathrel}{<}{=}
```


のように記述することで上記の \leq が出力できるようになります (ただし, この記号は `amssymb` パッケージでも提供されています). ここで定義した `\mathcomposite` は一般的には

```
\mathcomposite{<type>}{<upper-symbol>}{<lower-symbol>}
```

(`<type>` は記号の種類, `<upper-symbol>` は上側の記号, `<lower-symbol>` は下側の記号) のように用います. なお, “記号の種類” は “`\mathcomposite` で作った記号をどのような演算子として扱うか” という点の指定で, 2 項演算子, 関係演算子, 大型演算子の各々に対して, それぞれ `\mathbin`, `\mathrel`, `\mathop` を用います. さらに,

```
\mathcomposite[<number>]{...}...
```

(`<number>` は数値) のようにオプションをつけて用いると, 積み重ねる二つの記号の間隔を約 `<number> mu` だけ広げます. (`mu` は数式での長さの単位で, `1 mu` は `1/18 em` に相当します.) また, `\mathcomposite` を用いるときには, このコマンドを直接用いるよりも,

```
\def\leq{\mathcomposite{\mathrel}{<>}{=}}
```

のように定義してこの `\leq` を用いるようにする, という方法を用いるとよいでしょう. なお, `\mathcomposite*` のように `*` をつけて用いると, 2 個の記号を重ね書きしたものを出力します.

25.3. 大型演算子

ここでは, 大型演算子⁸⁰⁾を紹介します.

<code>\sum</code>	$\rightarrow \sum$	<code>\bigcap</code>	$\rightarrow \bigcap$	<code>\bigvee</code>	$\rightarrow \bigvee$
<code>\prod</code>	$\rightarrow \prod$	<code>\bigcup</code>	$\rightarrow \bigcup$	<code>\bigwedge</code>	$\rightarrow \bigwedge$
<code>\coprod</code>	$\rightarrow \coprod$	<code>\bigotimes</code>	$\rightarrow \bigotimes$	<code>\biguplus</code>	$\rightarrow \biguplus$
<code>\int</code>	$\rightarrow \int$	<code>\bigoplus</code>	$\rightarrow \bigoplus$	<code>\bigsqcup</code>	$\rightarrow \bigsqcup$
<code>\oint</code>	$\rightarrow \oint$	<code>\bigodot</code>	$\rightarrow \bigodot$	<code>\smallint</code>	$\rightarrow \int$

これらの大型演算子に対しては, `\int` と `\oint` を除き, ディスプレイ数式 (のスタイルの数式) では

$$\sum_{n=1}^N$$

のように演算子の上下に添字がつきます. しかし, それ以外 (のスタイルの数式) では

⁸⁰⁾“大型演算子” と称していますが, これは記号自身の大きさによるものではなく, 記号の用いられ方に関する分類です.

$$\sum_{n=1}^N$$

のように、他の記号類と同じ添字のつき方になります⁸¹⁾。

また、大型演算子の直後に `\nolimits` と書くと、数式のスタイルに関係なく、添字を記号の右上または右下につけるようになります。一方、大型演算子の直後に `\limits` と書くと、数式のスタイルによらず常に添字を記号の上下につけるようになります。例えば、

`\sum\limits_{n=1}^N a_n`

のように入力すると（単に \$ で挟んでいるので、ディスプレイ数式ではないことに注意してください）、

$$\sum_{n=1}^N a_n$$

のように出力されます。（先程の、ディスプレイ数式の場合の出力と比べてみてください。記号 \sum の大きさとその下の $n = 1$ の部分の大きさに注意するとよいでしょう。）

大型演算子に限らず、数式用記号はあまり大きいサイズで用いることはできないのですが、`\section` などのタイトル文字列の中などのように大きなサイズの文字を用いるところで数式用記号を用いたい場合には、`exscale` パッケージを使用する（プリアンブルに `\usepackage{exscale}` という記述を入れる）とある程度大きなサイズの大型演算子を使えるようになります。

25.4. 関数名

さて、 $\sin x$ のように出力するつもりで単に `\sin x` と（数式中で）書くと、 $\sin x$ のように出力されます。“`\sin`” のようなよく用いられる関数などについては、それらを出力するためのコマンドが用意されています。例えば、今の $\sin x$ に対しては、`\sin x` と記述します。次の表は、`\sin` のような関数名（など）のためのコマンドの表です。

<code>\sin</code>	→ <code>\sin</code>	;	<code>\arcsin</code>	→ <code>\arcsin</code>	;	<code>\max</code>	→ <code>\max</code>
<code>\cos</code>	→ <code>\cos</code>	;	<code>\arccos</code>	→ <code>\arccos</code>	;	<code>\min</code>	→ <code>\min</code>
<code>\tan</code>	→ <code>\tan</code>	;	<code>\arctan</code>	→ <code>\arctan</code>	;	<code>\sup</code>	→ <code>\sup</code>
<code>\cot</code>	→ <code>\cot</code>	;	<code>\sinh</code>	→ <code>\sinh</code>	;	<code>\inf</code>	→ <code>\inf</code>
<code>\sec</code>	→ <code>\sec</code>	;	<code>\cosh</code>	→ <code>\cosh</code>	;	<code>\exp</code>	→ <code>\exp</code>
<code>\csc</code>	→ <code>\csc</code>	;	<code>\tanh</code>	→ <code>\tanh</code>	;	<code>\log</code>	→ <code>\log</code>
<code>\arg</code>	→ <code>\arg</code>	;	<code>\coth</code>	→ <code>\coth</code>	;	<code>\ln</code>	→ <code>\ln</code>
<code>\deg</code>	→ <code>\deg</code>	;	<code>\gcd</code>	→ <code>\gcd</code>	;	<code>\lg</code>	→ <code>\lg</code>
<code>\lim</code>	→ <code>\lim</code>	;	<code>\dim</code>	→ <code>\dim</code>	;	<code>\hom</code>	→ <code>\hom</code>
<code>\limsup</code>	→ <code>\limsup</code>	;	<code>\det</code>	→ <code>\det</code>	;	<code>\Pr</code>	→ <code>\Pr</code>
<code>\liminf</code>	→ <code>\liminf</code>	;	<code>\ker</code>	→ <code>\ker</code>			

⁸¹⁾和の記号の類に対しては、添字を常に上下につけることになっているわけではありません。ただし、ディスプレイ数式以外での添字のつき方には（欧文組版における）歴史的事情があることも否定はしませんが。

これらはすべて文字列の前にバックスラッシュ記号をつけたもので出力できるので、特に意識して覚える必要はないでしょう。

また、“剰余”(あるいは“法”)の \bmod を出力するコマンドは、`\bmod`、`\pmod` の 2 種類が用意されています。これらのうち、`\bmod` は 2 項演算子の \bmod で、`\pmod` は括弧付きの \bmod です。例えば、

$$x = y \bmod p, x = y \pmod{p}$$

という入力からは

$$x = y \bmod p, x = y \pmod{p}$$

という出力が得られます。

注意 25.3

例えば、`\tr A` のような出力を得たい場合には、`\sin` などと同様の `\tr` というコマンドが使えれば都合がよいのですが、これは定義されていません。そこで、`\sin` などの定義を参考に定義してみます。実際、ファイル `latex.ltx` を読んでみると、

```
\def\sin{\mathop{\operator@font sin}\nolimits}
```

という記述があります。この中の、“sin” という文字列を他の文字列に取り換えれば、`\sin` と同様のコマンドが定義できます。例えば、コマンド `\tr` をブリアンブルで

```
\makeatletter
\def\tr{\mathop{\operator@font tr}\nolimits}
\makeatother
```

のように定義すると、`\tr A` という入力で `\tr A` が出力されるようになります。(ここで、`\makeatletter`、`\makeatother` というコマンドが現れていますが、これらは文字 “@” を含む名前のコマンドを用いることができるようにするためのコマンドです。)

また、今の `\tr` の定義では最後に `\nolimits` というコマンドを用いているので、`\tr` につけた添字は常に文字列 “tr” の右上または右下につくようになります。一方、`\lim` のようにディスプレイ数式では添字を上下につけるような“関数記号”については、`\nolimits` をつけないで定義します。例えば、`\lim` の定義は、

```
\def\lim{\mathop{\operator@font lim}}
```

のようになっています⁸²⁾。

⁸²⁾“教育的配慮”などの事情によって `\lim` のようなものへの添字を常に上下につける場合には、`\nolimits` を取り除く代わりに `\nolimits` を `\limits` に取り換えることになります。

26. 数式 (5) [括弧]

ここでは、数式で用いる括弧類について説明します。括弧類のうち“(”, “)”, “[”, “]” の 4 種類については、数式中にそのまま書き込むことができます。その他の括弧類は次の表にあるようなコマンドを用いて出力します。

<code>\{</code>	\rightarrow	$\{$;	<code>\lfloor</code>	\rightarrow	\lfloor	;	<code>\lgroup</code>	\rightarrow	$\{$
<code>\}</code>	\rightarrow	$\}$;	<code>\rfloor</code>	\rightarrow	\rfloor	;	<code>\rgroup</code>	\rightarrow	$\}$
<code>\langle</code>	\rightarrow	\langle	;	<code>\lceil</code>	\rightarrow	\lceil	;	<code>\lmoustache</code>	\rightarrow	\langle
<code>\rangle</code>	\rightarrow	\rangle	;	<code>\rceil</code>	\rightarrow	\rceil	;	<code>\rmoustache</code>	\rightarrow	\rangle
<code>\vert</code>	\rightarrow	$ $;	<code>\Vert</code>	\rightarrow	$\ $				

これらの記号のうち、`\{`, `\}`, `\vert`, `\Vert` は、それぞれ `\lbrace`, `\rbrace`, `|`, `\|` のように記述することもできます。さらに、`txfonts` パッケージの類は“2重の角括弧”のようなものも提供しています。

また、`\left`, `\right` というコマンドを用いると、括弧類の大きさを括弧の間にある数式のサイズに応じた大きさにすることができます。例えば、

```
\left( \left( \left( \left( \left(
x^2
\right)^3 \right)^4 \right)^5 \right)^6 \right)^7
```

のように入力すると、

$$\left(\left(\left(\left(x^2\right)^3\right)^4\right)^5\right)^6\right)^7$$

のような出力が得られます。この例からわかるように、`\left` は開き括弧の直前で用い、`\right` は閉じ括弧の直前で用います。ただし、`\left` と `\right` はこれらの間にある数式の大きさをういて括弧の大きさを決めるので、`\left` と `\right` は必ず対にして用いなければならないという点に注意してください。また、`\langle`, `\rangle` には大きさの上限があります。

一方、`\left`, `\right` は異なる括弧に対して用いても構いません。例えば、

```
\left[ -a^2, a^2 \right)
```

のように入力すると、

$$[-a^2, a^2)$$

のように出力されます。特に、括弧の代わりにピリオドを用いると括弧を省略することができます。(`\left`, `\right` は対にして用いなければならないので、開き括弧または閉じ括弧の一方しか必要でない場合には、他方に対してピリオドを用いるわけです。) 例えば、

`\left(A \right.`

という入力に対しては,

`(A`

のように出力されます.

Note: 文字 “<”, “>” をそのまま用いると, これらの文字は不等号, すなわち関係演算子として扱われます. したがって, `\langle`, `\rangle` を用いるべきところを不等号で代用しないでください⁸³⁾. 実際, `\langle`, `\rangle` の代わりに不等号を用いると, 次のように空白の大きさがおかしくなります⁸⁴⁾.

— INPUT —

```
$\langle v, w \rangle = \langle w, v \rangle$ \quad
$< v, w > = < w, v >$
```

— OUTPUT —

$$\langle v, w \rangle = \langle w, v \rangle \quad < v, w > = < w, v >$$

注意 26.1

括弧類のほとんどすべてには開き括弧と閉じ括弧の区別がありますが, `\vert` (または`|`) と `\Vert` (または`\|`), すなわち絶対値やノルムの記号には開き括弧と閉じ括弧の区別がありません. そのため, これらの記号は (数式中の空白の入れ方に関しては) 通常文字として取り扱われます. したがって, $|-x| = |x|$ のように記述すると, $-$ の前後の空白は “ $a - b$ ” の場合と同じ大きさに設定されて (つまり, マイナス記号は 2 項演算子として扱われて) “ $|-x| = |x|$ ” のように出力されます. そのような場合には, 文字 `|` が括弧類であることを明示するために `\left`, `\right` を補って

```
\left| -x \right| = |x|
```

のように入力することができます. 実際, この入力からは “ $|-x| = |x|$ ” という出力が得られます. (今の例では, $|\{-x\}| = |x|$ と入力するのが最も簡単です.)

⁸³⁾`\langle`, `\rangle` を用いた場合の括弧の開き具合を気にする人がいますが, 個々の文字の字形は使用するフォントによって定まっているのですから, 文書作成時には (固有名詞などに用いる文字以外の) 個々の文字の字形にこだわる必要はありません. 字形が気に入らなければ, 然るべき字形をもったフォントを使うように設定すればよいのです. 数式の記述においては, 括弧類を使うべきところに関係演算子を使うような “意味を取り違えた” 記述をしてはいけません. (現実には, 然るべき字形をもったフォントが見出せるほど充分に多くの数式用フォントが存在するとは限らないのですが, それは別の問題です.)

⁸⁴⁾等号 (をはじめとする関係演算子) とその両辺の値との間には多少空白が入るべきものです. (世間に存在する出版物には (初等教育用の教科書を含め) 数式をベタ組みにしているものが見られますが, それは単にその出版物を担当した編集者の不見識を表しているにすぎません. 実際, 編集者用ハンドブックの類における数式組版に関する記述には, (必要最小限の数学的素養と, 充分に実務をこなせるだけの組版に関する見識を持ち合わせた人間に言わせると) かなり “非常識” な記述が含まれます.)

注意 26.2

多少数式の記述に慣れてくると、

```
\begin{eqnarray}
\left\{ PV \& = \& nRT \right\} \\\
\ldots
\end{eqnarray}
```

のように、&記号を挟んで \left と \right を用いたくなるかもしれませんが、しかし、これはエラーになります。実際、eqnarray 環境や array 環境などでは、隣り合う&の間（および&と\\の間など）の各々を独立した数式として処理して、その後でそれらの小さな数式たちを並べます。したがって、上記の例では“\left\{ PV”、“=”、“nRT \right\}”の各々が独立した数式になり、最初と最後の数式には単独の \left, \right が現れています。これは、“\left と \right は対にして用いなければならない”という規則に違反しているのでエラーになるわけです。

このエラーを避けるためには、&を挟むような \left と \right の組がなくなるように適宜 \left. または \right. を補うか、後述する \bigl, \bigr などのコマンドを用いて括弧の大きさを調節するとよいでしょう。例えば、

```
\begin{eqnarray}
\left\{ PV \right. \& = \& \left. nRT \right\} \\\
\ldots
\end{eqnarray}
```

または

```
\begin{eqnarray}
\bigl\{ PV \& = \& nRT \bigr\} \\\
\ldots
\end{eqnarray}
```

のように記述します。

ただし、\left, \right はそれらの間にある数式のサイズに基づいて括弧の大きさを決定する（数式の意味を考えているわけではない）ので、適切でない大きさの括弧が選ばれる場合があります。例えば、

```
\left| |x| - |y| \right|
```

のように入力すると、“ $||x| - |y||$ ”のように出力されます。この場合、外側の絶対値記号を少し大きくして、“ $||x| - |y||$ ”のような出力を得るには、

```
\bigl| |x| - |y| \bigr|
```

のように入力します。ここで用いた \bigl, \bigr というのは、括弧類の大きさを指定するコマンドです。この \bigl と同様に括弧類の大きさを直接指定するためのコマンドには次のようなものがあります。

- `\bigl`, `\bigr`
- `\Bigl`, `\Bigr`
- `\biggl`, `\biggr`
- `\Biggl`, `\Biggr`

これらは、下にあるものほど大きな括弧を出力します。また、`\bigl` のようなコマンド名の最後の文字が “l” であるものは開き括弧用（というよりも“左側の括弧”用）で、コマンド名が文字 “r” で終わっているものは閉じ括弧用（というよりも“右側の括弧”用）です。（なお、一部の関係演算子（`\mid` など）は、その直前に `\bigr`, `\Bigr` など（上記のリストのコマンドたちの名前の最後の文字を “m” に変えたもの）を用いてサイズを変更することができます。）

ただし、`\bigl` などは、括弧の大きさを固定した大きさにするので、括弧の間にある数式の大きさに応じて適当なサイズのことをユーザ自身が選ばなければなりません。次の例を参照してください。

— INPUT —

```
$\Bigl| \sum_{n=0}^N a_n \Bigr|
\leq \sum_{n=0}^N |a_n|$
\[\Bigl| \sum_{n=0}^N a_n \Bigr|
\leq \sum_{n=0}^N |a_n| \]
\[\Biggl| \sum_{n=0}^N a_n \Biggr|
\leq \sum_{n=0}^N |a_n| \]
```

— OUTPUT —

$$\left| \sum_{n=0}^N a_n \right| \leq \sum_{n=0}^N |a_n|$$

$$\left| \sum_{n=0}^N a_n \right| \leq \sum_{n=0}^N |a_n|$$

$$\left| \sum_{n=0}^N a_n \right| \leq \sum_{n=0}^N |a_n|$$

この出力例の最初の2行からわかるように、`\Bigl` などを用いた場合、括弧のサイズは括弧で囲まれた数式などのサイズとは関係しません⁸⁵⁾。

注意 26.3

二項係数 $\binom{n}{m}$ を出力するには、`\{n \choose m\}` のように記述します。ここで、`\choose` を用いましたが、これは、

```
\def\choose{\atopwithdelims{}}
```

のように定義されています。この `\atopwithdelims` は、

⁸⁵⁾それだからこそ、括弧のサイズの“手動”での調整に使えるわけです。

```
{⟨numerator⟩ \atopwithdelims ⟨Ldelimiter⟩ ⟨Rdelimiter⟩ ⟨denominator⟩}
```

($\langle \text{numerator} \rangle$ は分子で $\langle \text{denominator} \rangle$ は分母, $\langle \text{Ldelimiter} \rangle$ と $\langle \text{Rdelimiter} \rangle$ は左右の括弧) のように用いて, “横線のない分数の両側を指定した括弧で囲んだもの” を出力します. したがって, $\left[\frac{n}{m} \right]$ のようなものを出力したい場合には,

```
{n \atopwithdelims [] m}
```

のように記述すればよいわけです. なお, $\backslash \text{left}$ または $\backslash \text{right}$ の場合と同様に, $\backslash \text{atopwithdelims}$ に対しても括弧の代わりにピリオドを用いると括弧が省略されます. さらに, $\backslash \text{atopwithdelims}..$ (これは横線のない分数だけを出力します) の代わりに $\backslash \text{atop}$ と書くこともできます⁸⁶⁾.

27. 数式 (6) [数式用記号]

§5 で紹介した記号の一部は \$ で挟んで出力しましたが, それらは数式用の記号として定義されています. しかし, §5 で紹介したもの以外にも多くの数式用記号があります. ここでは, §5 で扱わなかった数式用記号と数式用アクセント記号について説明します.

27.1. ギリシャ文字・ヘブライ文字

まず, ギリシャ文字の表を挙げます.

$\backslash \alpha$	$\rightarrow \alpha$	$\backslash \beta$	$\rightarrow \beta$	$\backslash \gamma$	$\rightarrow \gamma$
$\backslash \delta$	$\rightarrow \delta$	$\backslash \epsilon$	$\rightarrow \epsilon$	$\backslash \zeta$	$\rightarrow \zeta$
$\backslash \eta$	$\rightarrow \eta$	$\backslash \theta$	$\rightarrow \theta$	$\backslash \iota$	$\rightarrow \iota$
$\backslash \kappa$	$\rightarrow \kappa$	$\backslash \lambda$	$\rightarrow \lambda$	$\backslash \mu$	$\rightarrow \mu$
$\backslash \nu$	$\rightarrow \nu$	$\backslash \xi$	$\rightarrow \xi$	$\backslash \pi$	$\rightarrow \pi$
$\backslash \rho$	$\rightarrow \rho$	$\backslash \sigma$	$\rightarrow \sigma$	$\backslash \tau$	$\rightarrow \tau$
$\backslash \upsilon$	$\rightarrow \upsilon$	$\backslash \phi$	$\rightarrow \phi$	$\backslash \chi$	$\rightarrow \chi$
$\backslash \psi$	$\rightarrow \psi$	$\backslash \omega$	$\rightarrow \omega$	$\backslash \Gamma$	$\rightarrow \Gamma$
$\backslash \Delta$	$\rightarrow \Delta$	$\backslash \Theta$	$\rightarrow \Theta$	$\backslash \Lambda$	$\rightarrow \Lambda$
$\backslash \Xi$	$\rightarrow \Xi$	$\backslash \Pi$	$\rightarrow \Pi$	$\backslash \Sigma$	$\rightarrow \Sigma$
$\backslash \Upsilon$	$\rightarrow \Upsilon$	$\backslash \Phi$	$\rightarrow \Phi$	$\backslash \Psi$	$\rightarrow \Psi$
$\backslash \Omega$	$\rightarrow \Omega$	$\backslash \epsilon$	$\rightarrow \epsilon$	$\backslash \vartheta$	$\rightarrow \vartheta$
$\backslash \varpi$	$\rightarrow \varpi$	$\backslash \varrho$	$\rightarrow \varrho$	$\backslash \varsigma$	$\rightarrow \varsigma$
$\backslash \varphi$	$\rightarrow \varphi$				

これらは, ($\backslash \text{var}...$ という名前のもので以外は) 記号の読み方がそのままコマンド名になっています. ただし, オミクロン (ϕ) や大文字のアルファ (A) のようなものは普通のアルファベットを用いても構わないので, それらを入力するコマンドは用意され

⁸⁶⁾ただし, amsmath パッケージは $\backslash \text{atopwithdelims}$ のような $\text{T}_{\text{E}}\text{X}$ の組込みコマンドを (ユーザ自身に使わせないように) 再定義しています. amsmath パッケージ使用時には, “ $\{n \ \backslash \text{choose} \ m\}$ ” の代わりに “ $\backslash \text{binom}\{n\}\{m\}$ ” のように記述する, という具合に amsmath パッケージが提供するコマンドで代替します.

ていません⁸⁷⁾。

次の表はヘブライ文字の表です。ただし、数式用記号としては、数式で用いられることが（ヘブライ文字の中では比較的多く）ある 4 種類の文字に対してのみコマンドが用意されています。

<code>\aleph</code>	→	\aleph	;	<code>\beth</code>	→	\beth
<code>\gimel</code>	→	\gimel	;	<code>\daleth</code>	→	\daleth

これらのコマンドのうち、`\aleph` は標準の（何もパッケージを用いない状態の） \LaTeX でも使えますが、それ以外を用いるためには `amssymb` パッケージが必要です。これらの 4 種の文字を用いるときには `\usepackage{amssymb}` という記述をプリアンブルに入れるとよいでしょう。

27.2. 数式用の雑多な文字

次の表は、その他一般の“文字”の表です。

<code>\forall</code>	→	\forall	;	<code>\infty</code>	→	∞	;	<code>\wp</code>	→	\wp
<code>\exists</code>	→	\exists	;	<code>\partial</code>	→	∂	;	<code>\emptyset</code>	→	\emptyset
<code>\neg</code>	→	\neg	;	<code>\nabla</code>	→	∇	;	<code>\angle</code>	→	\angle
<code>\top</code>	→	\top	;	<code>\Re</code>	→	\Re	;	<code>\imath</code>	→	\imath
<code>\bot</code>	→	\bot	;	<code>\Im</code>	→	\Im	;	<code>\jmath</code>	→	\jmath
<code>\backslash</code>	→	\backslash	;	<code>\ell</code>	→	ℓ	;	<code>\hbar</code>	→	\hbar

ここで、`\neg` は、`\lnot` と記述しても出力できます。また、`\imath` と `\jmath` は、それぞれ i または j に数式用アクセント記号をつけるために用いる“点のない i ”，“点のない j ”です⁸⁸⁾。なお、数式中の小文字のエル“ l ”は、単に“ l ”と記述すれば充分です。（一般的な数式イタリックのフォントでは、単に“ l ”と記述しても他の文字と明確に区別できます。）`\ell` コマンドを用いるのは、原則として、真に“ ℓ ”の字形が必要である場合に限ります。

次の表の“文字”たちは、`latexsym` パッケージで定義されるものです。（したがって、この表の文字を用いるには、プリアンブルに `\usepackage{latexsym}` という記述を入れます。）

<code>\sqsubset</code>	→	\sqsubset	;	<code>\lhd</code>	→	\lhd	;	<code>\unlhd</code>	→	\unlhd
<code>\sqsupset</code>	→	\sqsupset	;	<code>\rhd</code>	→	\rhd	;	<code>\unrhd</code>	→	\unrhd
<code>\Box</code>	→	\Box	;	<code>\diamond</code>	→	\diamond	;	<code>\Join</code>	→	\Join
<code>\leadsto</code>	→	\leadsto	;	<code>\mho</code>	→	\mho				

⁸⁷⁾ 仮に字形が同じであったとしても）ギリシャ文字を通常の（ラテン文字の）アルファベットで代用するというのは“意味を取り違えた記述”の範疇に属するので、本来ならオミクロンなどを出力するためのコマンドを用意すべきところです。単に `\def\omicron{o}`（これは文字“ o ”で代用する場合の定義）のように定義すれば充分なので、定義する手間だけの問題です。“古い”処理系にはむやみにコマンドを定義する余りがなかったので、今の `\omicron` のようなものを定義しなかったのもやむを得ないことだったのでしょう...

⁸⁸⁾ したがって、アクセント記号をつけずに `\imath` や `\jmath` を単独で用いることは、（本稿のように記号自身の説明を行う場合を除き）あり得ません。

27.3. 矢印

次に、各種の矢印を挙げます．

<code>\leftarrow</code>	= \leftarrow	;	<code>\rightarrow</code>	= \rightarrow
<code>\Leftarrow</code>	= \Leftarrow	;	<code>\Rightarrow</code>	= \Rightarrow
<code>\leftrightharpoonup</code>	= \leftrightarrow	;	<code>\Leftrightarrow</code>	= \Leftrightarrow
<code>\uparrow</code>	= \uparrow	;	<code>\downarrow</code>	= \downarrow
<code>\Uparrow</code>	= \Uparrow	;	<code>\Downarrow</code>	= \Downarrow
<code>\updownarrow</code>	= \updownarrow	;	<code>\Updownarrow</code>	= \Updownarrow
<code>\leftharpoonup</code>	= \leftharpoonup	;	<code>\rightharpoonup</code>	= \rightharpoonup
<code>\leftharpoondown</code>	= \leftharpoondown	;	<code>\rightharpoondown</code>	= \rightharpoondown
<code>\rightleftharpoons</code>	= \rightleftharpoons			
<code>\hookrightarrow</code>	= \hookrightarrow	;	<code>\hookleftarrow</code>	= \hookleftarrow
<code>\nrightarrow</code>	= \nrightarrow	;	<code>\nearrow</code>	= \nearrow
<code>\swarrow</code>	= \swarrow	;	<code>\searrow</code>	= \searrow
<code>\mapsto</code>	= \mapsto	;	<code>\longmapsto</code>	= \longmapsto
<code>\longrightarrow</code>	= \longrightarrow	;	<code>\longleftarrow</code>	= \longleftarrow
<code>\Longrightarrow</code>	= \Longrightarrow	;	<code>\Longleftarrow</code>	= \Longleftarrow
<code>\longleftrightharpoonup</code>	= \longleftrightarrow	;	<code>\Longleftrightharpoonup</code>	= \Longleftrightarrow
<code>\iff</code>	= \iff			

これらのうち、 \leftarrow 、 \rightarrow はそれぞれ、`\gets`、`\to` と記述しても出力できます．また、`\Longleftrightharpoonup` と `\iff` は同じ記号を出力しますが、`\iff` の方は矢印の両側に空白を少し入れます．また、“ \mapsto ”、“ \longrightarrow ” は（写像などによる）要素間の対応を表すのに用います．例えば、“写像 $f: X \rightarrow Y$ というのは、 $X \ni x \mapsto y = f(x) \in Y$ のように…”のように用います．

27.4. 数式用アクセント

最後に、数式用アクセント記号を挙げます．

<code>\tilde{a}</code>	$\rightarrow \tilde{a}$;	<code>\acute{a}</code>	$\rightarrow \acute{a}$;	<code>\dot{a}</code>	$\rightarrow \dot{a}$
<code>\hat{a}</code>	$\rightarrow \hat{a}$;	<code>\grave{a}</code>	$\rightarrow \grave{a}$;	<code>\ddot{a}</code>	$\rightarrow \ddot{a}$
<code>\check{a}</code>	$\rightarrow \check{a}$;	<code>\breve{a}</code>	$\rightarrow \breve{a}$;	<code>\widetilde{ab}</code>	$\rightarrow \widetilde{ab}$
<code>\bar{a}</code>	$\rightarrow \bar{a}$;	<code>\vec{a}</code>	$\rightarrow \vec{a}$;	<code>\widehat{ab}</code>	$\rightarrow \widehat{ab}$
<code>\mathring{a}</code>	$\rightarrow \mathring{a}$						

これらのうち `\widetilde{}` と `\widehat{}` は、アクセントをつける記号の幅に応じて伸縮します．ただし、3 文字分の幅程度が限度です．なお、 $(PQR)^{\sim}$ のような出力を得るには、`\mathstrut` にアクセント記号をつけて、

`(PQR)\tilde{\mathstrut}`

のようにできます．この場合には `\sim` を上添字にして $(PQR)^{\sim}$ のように記述してもよいでしょう．

注意 27.1

アクセント記号をつける場合, \TeX はアクセントをつける文字(の傾き具合)に応じて適切な位置につけようとしています. そのため, ただ一つのアクセントをつけた場合には申し分ない出力が得られる代わりに, アクセントを2重につけて $\text{\hat{\hat{A}}}$ のように入力すると, 出力は “ $\hat{\hat{A}}$ ” のようになります. ここで, 2重につけたアクセント記号の位置が揃うようにするには, \skew というコマンドを用いて

```
\skew{5}\hat{\hat{A}}
```

のようにするとよいでしょう. (この入力に対する出力は “ $\hat{\hat{A}}$ ” のようになります.) この \skew は,

```
\skew{<number>}{<accent>}{<formula>}
```

のように用います. ただし, $\langle\text{accent}\rangle$ は \hat のような数式用アクセント記号, $\langle\text{formula}\rangle$ は $\langle\text{accent}\rangle$ をつける数式, $\langle\text{number}\rangle$ はアクセント記号の位置の補正量を表す数値です. (したがって, アクセント記号が適切な位置につくように $\langle\text{number}\rangle$ の値を手動で調節する必要があります.)

なお, amsmath パッケージを用いると単に $\text{\hat{\hat{A}}}$ のように2重以上にアクセント記号を重ねても適切な位置にアクセント記号がつきます⁸⁹⁾.

注意 27.2

アクセント記号の上に (アクセント記号ではない) 文字などを載せた

$$(0, \dots, 0, \overset{i}{1}, 0, \dots, 0)$$

(第 i 成分のみが 1 で, 他の成分は 0 のベクトル) のような記述は, §25 で述べた $\text{\stackrel}{rel}$ と組み合わせて

$$(0, \ldots, 0, \{\text{\stackrel{i}{rel}}{\text{\breve{1}}}\}, 0, \ldots, 0)$$

のようにできます.

28. 数式 (7) [数式での書体変更]

ここでは数式での書体変更について説明します. 数式以外のところでの書体変更については, §6 を参照してください. 数式での書体変更は,

```
\math...{\langle formula \rangle}
```

⁸⁹⁾古い版の amsmath パッケージを用いた場合, アクセント記号の位置が揃うようにするためには \Hat のような数式用アクセント記号用のコマンドの大文字版 (コマンド名の最初のアルファベットを大文字にしたもの) を使う必要がありました. また, やはり古い版の amsmath パッケージでは, 文字サイズが 10 pt を超えるような場合にアクセント記号の位置の調整がうまくいかない場合がありました.

($\langle formula \rangle$ は書体を変更する部分) のようにします。ここで、... の部分は書体変更コマンドごとに異なる文字列で、標準的に使えるものとしては次のものがあります。

<code>\mathrm{roman}</code>	→	roman
<code>\mathbf{boldface}</code>	→	boldface
<code>\mathit{italic}</code>	→	<i>italic</i>
<code>\mathsf{sans serif}</code>	→	sansserif
<code>\mathtt{typewriter}</code>	→	typewriter
<code>\mathnormal{math italic}</code>	→	<i>mathitalic</i>
<code>\mathcal{CALIGRAPHIC}</code>	→	<i>CALIGRAPHIC</i>

これらのうち、`\mathnormal` と `\mathcal` は数式専用の書体です。(`\mathnormal` は数式用イタリックにしますが、数式でのデフォルトの書体は数式用イタリックなので、`\mathnormal` を用いることはあまりありません。) ここで、`\mathcal` で出力できるのは大文字のアルファベットのみであるということに注意してください⁹⁰⁾。また、これらのコマンドはテキスト用の書体変更コマンドとは異なり、組み合わせて用いた場合には最も内側のコマンドで指定した書体になります。実際、

```
\mathbf{\mathit{bold italic?}}
```

という入力からは、

bolditalic?

のような出力が得られます。

なお、先程の数式用書体変更コマンドの表には `\mathsl` または `\mathsc` (それぞれ `\textsl` と `\textsc` の数式版) がありませんが、実際、それらは定義されていません。スラント体またはスモールキャピタルを数式中で使いたい場合には、

```
\DeclareMathAlphabet{\mathsl}{OT1}{cmr}{m}{sl}
\DeclareMathAlphabet{\mathsc}{OT1}{cmr}{m}{sc}
```

という記述をプリアンブルに入れるとよいでしょう⁹¹⁾。

次の表は、`amssymb` パッケージで用意される数式用書体の表です。(これらの書体を用いる場合には、`\usepackage{amssymb}` または `\usepackage{amssymb}` という記述をプリアンブルに入れます。(`amssymb` パッケージの中でも `amssymb` パッケージは読み込まれるので、`amssymb` パッケージを用いれば充分です。))

<code>\mathfrak{euler fraktur}</code>	→	eulerfraktur
<code>\mathbb{BLACKBOARD BOLD}</code>	→	BLACKBOARDBOLD

ここで、`\mathbb` も大文字のアルファベットのみに対して使えるコマンドです。

⁹⁰⁾小文字を出力しようとした場合には、エラーにはなりませんが意図しない記号類が出力されます。

⁹¹⁾`\textsl` などを数式中で使うことはできませんが、テキスト用の書体変更コマンドを用いると添字の中であっても数式の外部での文字サイズで出力されてしまいます。したがって、数式用の書体変更コマンドを定義した方がよいでしょう。

ここまでで述べた書体変更コマンドは、アルファベット（と書体変更を許している一部の記号）と数字の書体を変更しますが、その他の数式用記号の書体は変更されません。例えば、

```
\alpha' = \mathbf{\alpha}
```

のように入力しても、出力は、

$$\alpha' = \alpha$$

のようになり、`\mathbf` は `\alpha` の書体を変更しません。このような場合、一般の数式用記号（大型演算子を除く）を太字にするには、`\boldmath` というコマンドが使えます。今の場合は、

```
\alpha' = \mbox{\boldmath $\alpha$}
```

のように記述すれば、

$$\alpha' = \alpha$$

のように出力されます。今の例のように数式の一部のみを太字にする場合には、

```
\mbox{\boldmath $\langle symbol \rangle$}
```

（`\langle symbol \rangle` は太字にしたい数式用記号）のように用います。一方、数式全体を太字にする場合には、単に

```
{\boldmath $...$}
```

のように記述します。

この `\boldmath` は、他の数式用書体変更コマンドとは異なり、数式の外部で用いなければなりません⁹²⁾。（最初の例では `\mbox` の中で `\boldmath` を用いていることに注意してください⁹³⁾。）`\boldmath` を他の書体変更コマンドと同様に用いたい場合には、`amsmath` パッケージで提供される `\boldsymbol` コマンドを用いるとよいでしょう。

一方、`\boldmath` を用いても大型演算子は太字になりません。例えば、

```
\mbox{\boldmath $\sum_{n=1}^N a_n$}
```

という入力からは

$$\sum_{n=1}^N a_n$$

⁹²⁾ 実際、`\boldmath` は書体変更コマンドではなく、“数式のバージョン”を変更するコマンドです。

⁹³⁾ `\mbox` の中は数式ではなくなるので、`\mbox` の中なら `\boldmath` を用いることができます。また、`\mbox` の中は数式ではないので、`\mbox` の中の `\alpha` は `\mbox` の中で再び数式にしてその数式の中に入れなければなりません。

という出力が得られ、これは `\boldmath` を用いない出力と（書体に関しては）同じです。このような記号を太字で出力したい場合には、`amsmath` パッケージで用意されている `\pmb` (poor man's bold の省略形だそうです) というコマンドを用いるとよいでしょう。実際、

```
\pmb{\sum}_{n=0}^N a_n
```

という入力から

$$\sum_{n=0}^N a_n$$

という出力が得られるようになります。この例からわかるように、`\pmb` は

```
\pmb{\langle symbol \rangle}
```

(`\langle symbol \rangle` は太くする記号) のように用います。ただし、`\pmb` は同じ記号を少しづつずらして重ね書きするだけなので、ときとして美しくない出力になることがあります。

最後に、数式中に普通の文章を書き込む方法について説明します。例えば、

```
{A_p | p is a prime number}
```

のような記述を行いたいときに、単に `\mathrm` を用いて

```
{A_p \mid p \mathrm{is a prime number}}
```

のように入力すると、単語間の空白がすべて無視されて

```
{A_p | pisaprimenumber}
```

のように出力されます。このような場合、

```
{A_p | p is a prime number}
```

という出力を得る方法には、

- (1) “`\$ \{A_p \mid p$ is a prime number$ \}`” のように数式モードに出入りする。
- (2) “`\$ \{A_p \mid p \ \mbox{\ \ is a prime number} \}`” のように数式中のテキストの部分を `\mbox{...}` の...の部分に入れる。
- (3) “`\$ \{A_p \mid p \mathrm{\ \ is\ a\ prime\ number} \}`” のように `\mathrm` を用い、単語間には明示的に空白を入れる。

という3通りの方法があります⁹⁴⁾。(2)、(3)で、`_`を用いているのは空白の調節のためです (§20 を参照してください)。テキストの部分で改行が行われても構わない場合

⁹⁴⁾数式中で `\textrm` を用いるというのは、本質的には (2) と同じです。

は (1) を用い、改行が起こると不都合がある場合には (2) を用いるとよいでしょう。
((3) はテキストの部分の空白を `_` にするのが面倒なので、あまりお勧めしません。)

29. 数式 (8) [数式のスタイル]

これまでの説明に現れた、大型演算子への添字のつき方はディスプレイ数式とそれ以外の数式とは異なる、といった話から推察されるように、数式には複数のスタイルが存在します。ここでは、そのような「数式のスタイル」について説明します。

まず、 \TeX の数式には次の 4 種類のスタイルがあります⁹⁵⁾。

- `\displaystyle`:
ディスプレイ数式 (の添字などではない部分) のスタイル
- `\textstyle`:
テキスト中の数式 (の添字などではない部分) のスタイル
- `\scriptstyle`:
添字 (中で、さらに添字になっているような部分以外の部分) のスタイル
- `\scriptscriptstyle`:
添字の中の添字のスタイル

なお、`\scriptscriptstyle` が最も下位のスタイルで、“添字の中の添字の中の添字”のようなものに対しても `\scriptscriptstyle` が用いられます。数式のスタイルは、 \TeX が自動的に選択するので、たいていの場合ユーザ自身がスタイルを指定する必要はありません。ただし、(\TeX が選択した) 標準的なスタイル以外のスタイルで出力する場合には、上記のコマンドを用いてスタイルを直接指定できます。実際、次の例のようになります。

— INPUT —

```
\sum_{n=1}^N a_n$ \quad $\displaystyle\sum_{n=1}^N a_n$
```

— OUTPUT —

$$\sum_{n=1}^N a_n \quad \sum_{n=1}^N a_n$$

— INPUT —

```
$a_{\varepsilon}$ \quad $a_{\textstyle\varepsilon}$
```

— OUTPUT —

$$a_{\varepsilon} \quad a_{\varepsilon}$$

⁹⁵⁾これらは、さらに細分されますがここでは扱いません。

なお、数式の一部分だけスタイルを変更したい場合には、スタイルを変更する部分を `{ }` で囲みます。例えば、`$a{\scriptstyle b}c$` のように入力すると、`b` だけが `\scriptstyle` になった “ abc ” という出力が得られます。

次に、添字のスタイルの定められ方を説明します。T_EX は添字を見つけると、その時点の数式のスタイルに応じて添字のスタイルを次のように設定します。

- `\displaystyle` , `\textstyle` の場合:
添字は `\scriptstyle` になります。
- `\scriptstyle` , `\scriptscriptstyle` の場合:
添字は `\scriptscriptstyle` になります。

例えば、`$x^{y^z}` , `a_{b_c}` という入力からは x^{y^z} , a_{b_c} という出力が得られます。ここで、“`x^{y^z}`” , “`a_{b_c}`” の全体は `\textstyle` ですから、その中の添字の “`y^z`” , “`b_c`” の部分は `\scriptstyle` になります。また、それらの中の添字の “`z`” , “`c`” の部分は `\scriptscriptstyle` になるわけです。

分数の分母・分子のスタイルは次のようになります⁹⁶⁾。

- `\displaystyle` の分数:
分母・分子は `\textstyle` になります。
- `\textstyle` の分数:
分母・分子は `\scriptstyle` になります。
- `\scriptstyle` , `\scriptscriptstyle` の分数:
分母・分子は `\scriptscriptstyle` になります。

例えば、

```
\[
  \frac{1}{\frac{1}{x} + \frac{1}{y}}
= \frac{2xy}{x+y}
\]
```

という入力からは、

$$\frac{1}{\frac{1}{x} + \frac{1}{y}} = \frac{2xy}{x+y}$$

という出力が得られます。ここで、数式全体は `\displaystyle` なので、左辺にある “`\frac{1}{x}`” , “`\frac{1}{y}`” は `\textstyle` になります。したがって、左辺の “`x`” , “`y`” は `\scriptstyle` で出力されます。また、右辺の分母・分子は `\textstyle` で出力されています。

⁹⁶⁾なお、出版用の原稿を作成する場合には、インライン数式（段落中に埋め込まれた数式）の中の分数にいちいち `\displaystyle` を適用するようなことをしないでください。（そのような記述は“子供じみて”いますし、教育的配慮を真に必要とする場合であっても `\frac` の定義または数式関係の設定を変更することによって分数の出力形式を（組版時に）調整することができます。）

30. 数式 (9) [下線など]

ここでは、下線をついたり \overrightarrow{AB} のような出力を行うためのコマンドについて説明します。このような数式の上下につけるものには、次のようなものがあります。

$$\begin{array}{llll} \backslash underline{ABC} & \rightarrow & \underline{ABC} & ; \backslash overline{ABC} & \rightarrow & \overline{ABC} \\ \backslash underbrace{ABC} & \rightarrow & \underbrace{ABC} & ; \backslash overbrace{ABC} & \rightarrow & \overbrace{ABC} \\ \backslash overleftarrow{ABC} & \rightarrow & \overleftarrow{ABC} & ; \backslash overrightarrow{ABC} & \rightarrow & \overrightarrow{ABC} \end{array}$$

これらのうち、`\underline`、`\overrightarrow`、`\overleftarrow` は数式でないとこ
ろでも用いることができます。ただし、`\underline` で下線をつけた範囲では改行が
行われないので、長いテキストに下線をつけるのは避けた方がよいでしょう⁹⁷⁾。

また、数式の上下につける括弧は

$$\overbrace{a + a + \cdots + a}^{n \text{ terms}}$$

のように“添字”をとともなうことも多いのですが、これは `\overbrace{...}` に添字を
つければ出力できます。

実際、上記の出力は、

$$\overbrace{a+a+\cdots+a}^{n}; \mathrm{terms}}$$

という入力から得られたものです。(ここで用いた `\;` は数式中の空白を調節するコマ
ンドです。) `\underbrace` に関しても同様で、

$$\begin{array}{l} a^n = \\ \underbrace{a \times a \times \cdots \times a}_{n \text{ 個}} \end{array}$$

という入力からは、

$$a^n = \underbrace{a \times a \times \cdots \times a}_{n \text{ 個}}$$

という出力が得られます。ここで、添字の中にテキストを入れるために `\mbox` を用い
ていますが、この `\mbox` の中では文字サイズを指定しなければ数式の外での文字サイ
ズが用いられます。したがって、今の入力例で `\scriptsize` を省略したら、“個”の
文字が不自然に大きくなってしまいます。

なお、上記の6種のコマンドのうち、`\overline` と `\underline` は、添字の中で用い
た場合にも正しいサイズで出力されます。一方、その他の4種は、`\overbrace{...}` な
どの中身を(特に指定しない限り) `\displaystyle` にするので、添字の中で用いた場
合には次の例のようになります⁹⁸⁾。

⁹⁷⁾なお、強調の類に下線を用いるのは書体変更が自由にはできなかった(タイプライタ時代の)名残とも
言われています。単に強調を行うだけであれば、書体の変更を行った方がよいでしょう。ただし、(j)ulem
パッケージなど、行分割可能な下線を出力するためのパッケージも知られています。

⁹⁸⁾amsmath パッケージを使用した場合、`\overrightarrow` と `\overleftarrow` については、`\underline`
と同様に数式のスタイルに応じた処理を行うようになります。

— INPUT —

```
f_{\overline{AB}}\quad f_{\overrightarrow{AB}}
```

— OUTPUT —

$$f_{\overline{AB}} \quad f_{\overrightarrow{AB}}$$

`\overrightarrow`などを添字の中で用いた場合にも正しく

$$f_{\overrightarrow{AB}}$$

のような出力を得るには,

```
f_{\overrightarrow{\scriptstyle AB}}
```

のように中身のスタイルを指定します.

注意 30.1

例えば, \widehat{AB} のように “弧” の記号を出力するには⁹⁹⁾, `\stackrel` を用いて `\stackrel{\textstyle\mathrm{frown}}{\mathrm{AB}}` くらいの記述を行います.

なお, “弧 ABCDE” のようなものに対しては, 楽譜用のフォントを用いるなどの方法で長い弧を載せるようにしたパッケージなども存在するので, 興味があればそのようなものを用いるのもよいでしょう.

31. 数式 (10) [数式番号付きのディスプレイ数式]

31.1. equation/eqnarray 環境の基本

ここでは, 数式番号付きのディスプレイ数式を出力するために用いる `equation` 環境と `eqnarray` 環境について説明します. まず, 1 行のみの数式は `equation` 環境で出力できます. `equation` 環境は

```
\begin{equation}
(( 数式 ))
\end{equation}
```

のように用います. 例えば,

```
\begin{equation}
PV = nRT
\end{equation}
```

という入力からは,

⁹⁹⁾ただし, この記法は日本の (初等教育での) 方言のようで, 一般には特に何もつけずに “弧 AB” のように記述すれば充分です.

$$PV = nRT \quad (1)$$

という出力が得られます．この出力例のように，`equation` 環境では数式を中央寄せにし，数式番号を右側に出力します．数式を左寄せにして出力する場合には，`\documentclass` のオプションに “`fleqn`” を加えます．また，数式番号を左側に出力する場合には，(やはり `\documentclass` の) “`leqno`” オプションを用います．なお，“`leqno`” オプションは次に説明する `eqnarray` 環境に対しても適用されます．(“`fleqn`” オプションも `eqnarray` 環境に適用されますが，`eqnarray` 環境の場合には `&` 記号の位置を揃えてできる “数式からなる配列 (表)” の全体を左寄せにするので，各々の行が左寄せになるとは限りません．)

複数行にわたるディスプレイ数式を出力するためには，`eqnarray` 環境が使えます．`eqnarray` 環境の書式は次のとおりです．

```
\begin{eqnarray}
\langle left_1 \rangle & \langle middle_1 \rangle & \langle right_1 \rangle \\
\langle left_2 \rangle & \langle middle_2 \rangle & \langle right_2 \rangle \\
.....
\langle left_n \rangle & \langle middle_n \rangle & \langle right_n \rangle
\end{eqnarray}
```

ここで， $\langle left_k \rangle$ ($k = 1, 2, \dots, n$) は k 行目の数式の左辺で， $\langle middle_k \rangle$ と $\langle right_k \rangle$ はそれぞれ k 行目の数式の中辺と右辺です．(1 行にちょうど 2 個の `&` を入れる必要はありません．例えば，2 個目の `&` が無い場合には右辺が空であるものとして扱われます．ただし，1 行に 3 個以上の `&` が含まれるとエラーになります．) これは表の書き方に似ていますが，実際，`eqnarray` 環境は列の揃え方の指定を “`rc1`” にした表だということもできます．(表については §10 を参照してください．) ただし，`eqnarray` 環境では，1 列目と 3 列目の配列要素 (つまり，各数式の左辺と右辺) は `\displaystyle` の数式になり，2 列目の配列要素は `\textstyle` の数式になります．(この `\displaystyle` などについては §29 を参照してください．) なお，数式番号をつけない行がある場合には，その行に `\nonumber` というコマンドを入れます．また，すべての行に数式番号をつけない場合には `eqnarray*` 環境を用います．

Note: `\` の直後に文字 “`*`”，“`[`” で始まる配列要素を置くときには，配列要素の直前に `{}` を入れてください．(これは `\` の処理の都合によります．) また，`\` [`\langle length \rangle`] (`\langle length \rangle` は単位付きの長さ) のように記述すると，行間隔を `\langle length \rangle` だけ増やすことができます．

`eqnarray` 環境の例をいくつか挙げます．

— INPUT —

```
\begin{eqnarray}
PV & = & nRT \\
\left( P + \frac{a}{V^2} \right) (V - nb) & = & nRT
\end{eqnarray}
```

— OUTPUT —

$$PV = nRT \quad (2)$$

$$\left(P + \frac{a}{V^2}\right)(V - nb) = nRT \quad (3)$$

— INPUT —

```
\begin{eqnarray}
&& PV = nRT \nonumber \\
&& \left( P + \frac{a}{V^2} \right) (V - nb) = nRT \\
\end{eqnarray}
```

— OUTPUT —

$$PV = nRT$$

$$\left(P + \frac{a}{V^2}\right)(V - nb) = nRT \quad (4)$$

— INPUT —

```
\begin{eqnarray*}
\Gamma(x+1) &= & \int_0^\infty t^x e^{-t} dt \\
&= & \left[ t^x \cdot (-e^{-t}) \right]_0^\infty \\
&\quad + \int_0^\infty x t^{x-1} e^{-t} dt \\
&= & x \Gamma(x) \\
\end{eqnarray*}
```

— OUTPUT —

$$\begin{aligned} \Gamma(x+1) &= \int_0^\infty t^x e^{-t} dt \\ &= [t^x \cdot (-e^{-t})]_0^\infty + \int_0^\infty x t^{x-1} e^{-t} dt \\ &= x \Gamma(x) \end{aligned}$$

ここの2番目の例のように記述し、さらに `\documentclass` のオプションに “fleqn” を加えると、eqnarray 環境での数式の左寄せができます。

31.2. eqnarray 環境の応用

さて、単に複数の数式を並べて

$$(f(x)g(x))' = f'(x)g(x) + f(x)g'(x) \quad (5)$$

$$\left(\frac{g(x)}{f(x)}\right)' = \frac{g'(x)f(x) - f'(x)g(x)}{f(x)^2} \quad (6)$$

のように出力する場合には、

```
\begin{eqnarray}
& (f(x)g(x))' = f'(x)g(x) + f(x)g'(x) \\
& \left( \frac{g(x)}{f(x)} \right)' \\
& = \frac{g'(x)f(x) - f'(x)g(x)}{f(x)^2}
\end{eqnarray}
```

のようにすることが考えられます。しかし、eqnarray 環境では 2 列目の配列要素は \textstyle で出力されるので、この入力に対しては、

$$(f(x)g(x))' = f'(x)g(x) + f(x)g'(x) \quad (7)$$

$$\left(\frac{g(x)}{f(x)} \right)' = \frac{g'(x)f(x) - f'(x)g(x)}{f(x)^2} \quad (8)$$

のように出力されてしまいます。したがって、実際には、\displaystyle を補って、

```
\begin{eqnarray}
& \displaystyle (f(x)g(x))' = f'(x)g(x) + f(x)g'(x) \\
& \displaystyle \left( \frac{g(x)}{f(x)} \right)' \\
& = \frac{g'(x)f(x) - f'(x)g(x)}{f(x)^2}
\end{eqnarray}
```

のように記述します。これは、2 列目の要素だけの事情ですが、\displaystyle を補うのが面倒ならば下記の注意 31.1 を参照してください。(ただし、amsmath パッケージを用いると、gather 環境を用いて今の例と同様のことを行うことができます。)

注意 31.1

eqnarray 環境で出力する数式をすべて (何列目の配列要素でも) \displaystyle で出力する場合には、次の記述をプリアンプルに入れてみてください。

```
\makeatletter
\def\eqnarray{%
  \stepcounter{equation}%
  \def\@currentlabel{\p@equation\theequation}%
  \m@th
  \global\@eqnswtrue \global\@eqcnt\z@
  \tabskip\@centering \let\\\@eqncr
  $$\everycr{}\halign to\displaywidth\bgroup
    \hskip\@centering
    $\displaystyle\tabskip\z@skip{##}$\@eqnse1
  &\global\@eqcnt\@ne
    \hskip \tw@\arraycolsep%% optional space
    \hfil$\displaystyle{\{}##{\}}$\hfil
  &\global\@eqcnt\tw@
    \hskip \tw@\arraycolsep%% optional space
    $\displaystyle{##}$\hfil\tabskip\@centering
  &\global\@eqcnt\thr@@
    \hbox to\z@\bgroup\hss##\egroup
    \tabskip\z@skip
  \cr}
```

```
\makeatother
```

この定義では（基本的には）数式のスタイルを変更しているだけなので，列どうしの間隙間が広すぎるように思えるかもしれません．その場合には，“optional space” というコメントをつけた行にある

```
\hskip \tw@arraycolsep
```

という記述を削除してください．（削除する代わりに，適当な大きさの空白を入れるコマンドに取り換えるのもよいでしょう．）

注意 31.2

equation 環境，eqnarray 環境での数式番号は，デフォルトでは，(1)，(2) のように単に番号のみが出力されます．これを，例えば，§3 の数式には先頭から順に (3.1)，(3.2) などとなるように \section の番号を添えて番号をつける場合には

```
\def\theequation{\thesection.\arabic{equation}}
\makeatletter
\@addtoreset{equation}{section}
\makeatother
```

という記述をプリアンブルに入れます．（なお，\thesection は \section の番号を出力するコマンドです．）数式番号の書式は，\theequation というコマンドで指定されるので，上記の \def\theequation{...} の部分のように \theequation を再定義すると，数式番号の書式が変更されます．ただ，それだけでは，\section が変わっても数式番号はリセットされない（つまり，§1 に 10 個の（番号付きの）数式があったとすると，数式番号が (1.1) ...，(1.10)，(2.11)，(2.12) ... のようについてしまう）ので，\@addtoreset を用いて，\section が変わると数式番号をリセットするようにしています．なお，amsmath パッケージを用いた場合には今の一連の定義を

```
\numberwithin{equation}{section}
```

のようにして行えます．

また，数式番号を囲む括弧を変更するには \@eqnnum の定義を変更します．この \@eqnnum は

```
\def\@eqnnum{{\normalfont \normalcolor (\theequation)}}
```

のように定義されていますが，この定義中の括弧 (,) が (equation 環境と eqnarray 環境での) 数式番号を囲む括弧です．したがって，数式番号を囲む括弧を [と] に変更する場合には，上記の定義の括弧を変更した

```
\makeatletter
\def\@eqnnum{{\normalfont \normalcolor [\theequation]}}
\makeatother
```

という記述をプリアンブルに入れるとよいでしょう．ただし，この変更は `amsmath` パッケージで提供される数式環境には影響しません．`gather` 環境等での数式番号を囲む括弧も変更する場合には，`\tagform@` の定義の中の括弧を変更した

```
\makeatletter
\def\tagform@#1{%
  \maketag@@@{[\ignorespaces#1\unskip\@italiccorr]}
\makeatother
```

のような記述も (`amsmath` パッケージを読み込んだ後で) 用います．

一方，数式番号に一時的に“補助番号”をつけて，(1a)，(1b) などのようにする場合には，`amsmath` パッケージで用意されている `subequations` 環境を用いるとよいでしょう．この環境は，

```
\begin{subequations}
  (( 補助番号をつけたい数式 ))
\end{subequations}
```

のように用います．例えば，

```
\begin{subequations}
  \begin{eqnarray}
    x + y = 10 \\
    2x + 4y = 26
  \end{eqnarray}
\end{subequations}
```

のように入力すると，

$$x + y = 10 \tag{9a}$$

$$2x + 4y = 26 \tag{9b}$$

のように出力されます．なお，`amsmath` パッケージを用いずにこの `subequations` 環境を用いたい場合には，とりあえず，

```
\makeatletter
\newcounter{subeqncnt}
\def\thesubeqncnt{\alph{subeqncnt}}%% 「補助番号」の形式
\def\subequations{\begingroup%
  \stepcounter{equation}\edef\@tempa{\theequation}%
  \let\c@equation\c@subeqncnt\c@subeqncnt\z@
  \edef\theequation{\@tempa\noexpand\thesubeqncnt}}
\let\endsubequations\endgroup
\makeatother
```

のような定義をプリアンブルに入れるとよいでしょう．

注意 31.3

例えば,

$$\begin{array}{rcl} ax + by & = & p \\ (a + b)x + aby & = & q \end{array} \quad (10)$$

のように, 複数の式に一つの番号をつけるには, `equation` (または `eqnarray`) 環境と (§32 で説明する) `array` 環境を組み合わせる,

```
\begin{equation}
\begin{array}{rcl}
a x + b y & = & p \\
(a + b) x + a b y & = & q
\end{array}
\end{equation}
```

のように記述することができます. また, `array` 環境の使い方を換えれば数式を中央寄せにすることもできます.

— INPUT —

```
\begin{equation}
\begin{array}{c}
a x + b y = p \\
(a + b) x + a b y = q
\end{array}
\end{equation}
```

$$\begin{array}{rcl} ax + by & = & p \\ (a + b)x + aby & = & q \end{array} \quad (11)$$

ただし, 必要に応じて `array` 環境の各配列要素に `\displaystyle` を補うとよいでしょう.

一方, `amsmath` パッケージを用いれば, 上記の出力 (と同様の出力) はそれぞれ,

```
\begin{equation}
\begin{aligned}
a x + b y & = p \\
(a + b) x + a b y & = q
\end{aligned}
\end{equation}

\begin{equation}
\begin{gathered}
a x + b y = p \\
(a + b) x + a b y = q
\end{gathered}
\end{equation}
```



```
\end{equation}
```

という入力からも得られます .

32. 数式 (11) [行列]

ここでは行列の書き方について説明します . 容易に想像できるように , 行列を書くためには “配列要素が数式であるような表” を書けばよいわけです . 実際 , \LaTeX はそのような表を書くために `array` 環境を用意しています . `array` 環境は `tabular` 環境と同様に ,

```
\begin{array}{列の揃え方の指定}
(1,1) 成分 & (1,2) 成分 & \dots & (1,n) 成分 \\
(2,1) 成分 & (2,2) 成分 & \dots & (2,n) 成分 \\
\dots\dots\dots
(m,1) 成分 & (m,2) 成分 & \dots & (m,n) 成分 \\
\end{array}
```

のように用います . ここで , “列の揃え方の指定” は文字 “c” , “r” , “l” , “|” などを繰り返したもので , ここで挙げた 4 種類について説明すると ,

- c: 中央寄せの指定
- r: 右寄せの指定
- l: 左寄せの指定
- |: 縦線を入れる指定

です . (列の揃え方は `tabular` 環境と全く同様に指定できるので , 詳しくは §10 を参照してください .) ただし , `array` 環境は数式の中で用います . また , `array` 環境の各配列要素は自動的に (`\textstyle` の) 数式になります . なお , `array` 環境だけでは行列を囲む括弧は出力されないので , 適当な括弧を補います . 例えば ,

```
\left(
\begin{array}{cccc}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{m1} & a_{m2} & \cdots & a_{mn}
\end{array}
\right)
```

という入力に対しては ,

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

という出力が得られます .

また , §10 では `\arraystretch` などの表の体裁に関するパラメータを述べましたが , これらのうちの `\tabcolsep` 以外のものは `array` 環境の体裁を決めるのにも用いられています . 一方 , `array` 環境での列間隔 (の $1/2$) は , `\arraycolsep` によって指定されます . (表の体裁を変更した例については , §10 を参照してください .) もちろん , `\hline` , `\cline` を用いることもできます .

— INPUT —

```
\left(
\begin{array}{c|ccc}
\alpha & & & * & \\
0 & a_1 & & & \\
\vdots & & & \ddots & \\
0 & & & & a_n
\end{array}
\right)
```

— OUTPUT —

$$\left(\begin{array}{c|ccc} \alpha & & & * \\ \hline 0 & a_1 & & \\ \vdots & & & \ddots \\ 0 & & & a_n \end{array} \right)$$

また , `array` 環境を次のように用いることもできます .

— INPUT —

```
\[ |x| = \left\{
\begin{array}{rl}
x & \text{\mbox{($x \ge 0$ のとき)}} \\
-x & \text{\mbox{($x < 0$ のとき)}}
\end{array}
\right. \]
```

— OUTPUT —

$$|x| = \begin{cases} x & (x \geq 0 \text{ のとき}) \\ -x & (x < 0 \text{ のとき}) \end{cases}$$

この例では , `\left\{` と `\right.` を組み合わせていることにも注意してください . (`\cases` というコマンドを用いても , 今の例と同様の出力を得ることができます . ただし , `\cases` は , plain $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 流の記述を要求するコマンドなので , ここでは扱わないことにします .)

また , 行列には , 次の例のように大きな文字を入れたい場合があります .

$$\begin{pmatrix} a_1 & & * \\ & a_2 & \\ O & & \ddots \\ & & & a_n \end{pmatrix}$$

そのような場合には,

```
\def\hugesymbol#1{%
  \mbox{\strut\rlap{\smash{\Huge$#1$}}\quad}}
```

という記述をプリアンブルに入れて, この `\hugesymbol` コマンドを用いるとよいでしょう. 実際, 上記の例は,

```
\left( \begin{array}{cccc}
a_1 & & & \hugesymbol{*} \\
& a_2 & & \\
& & \ddots & \\
\hugesymbol{O} & & & a_n
\end{array} \right)
```

という入力から得られたものです. (ただし, 数式用の記号類を大きなサイズで出力する場合には, `exscale` パッケージを用いる (プリアンブルに `\usepackage{exscale}`) という記述を入れる) 必要がある場合があります.)

33. 数式 (12) [数式での空白の調整]

§20 ではテキスト (非数式部分) での空白を調節するコマンドを説明しましたが, ここでは, 数式での空白の調節を扱います. まず, 数式での空白は次のようなコマンドで調節できます. これらのうち, `\, (\,+ 空白文字)`, `\quad`, `\qquad` の 3 種はテキストでの空白の調節に用いられるものと同じです.

<code>\$M\!N\$</code>	\rightarrow	MN
<code>\$MN\$</code>	\rightarrow	MN
<code>\$M\,N\$</code>	\rightarrow	$M N$
<code>\$M\>N\$</code>	\rightarrow	$M N$
<code>\$M\;N\$</code>	\rightarrow	$M N$
<code>\$M\ N\$</code>	\rightarrow	$M N$
<code>\$M\quad N\$</code>	\rightarrow	$M \quad N$
<code>\$M\qquad N\$</code>	\rightarrow	$M \qquad N$

(この表の 2 行目は比較を行うために入れてあります.) ここで, `\!` は空白をつめるのに用います. なお, `\quad`, `\qquad` は数式のスタイルによらず, (数式の外で用いられているフォントによって決まる) 一定の大きさの空白を入れます. その他の数式用の空白調節のコマンドは, 数式のスタイルに応じて空白の大きさが変化します. 例えば,

```
$M\quad N$ \quad $\scriptstyle M\quad N$\quad
$M\;N$ \quad $\scriptstyle M\;N$
```

という入力に対しては,

$$\begin{array}{cc} M & N \\ M & N \end{array}$$

という出力が得られます.

以下, 数式での空白について注意したい例を挙げます. 一般的には, 根号・積分記号・(a/b の形式の) 分数・括弧の前後と, 最後の例での $dx dy$ のような空白がないと意味がわかりにくくなることに注意するとよいでしょう. なお, 次の表の [] 内の出力は, 左側の入力の中の空白関係のコマンドを省いた場合の出力です.

$$\begin{array}{lll} \sqrt{\log x} & \rightarrow \sqrt{\log x} & [\sqrt{\log x}] \\ x^2/2 & \rightarrow x^2/2 & [x^2/2] \\ \log n & \rightarrow \log n & [\log n] \\ \Gamma_n & \rightarrow \Gamma_n & [\Gamma_n] \\ \int_0^1 \int_0^1 f(x,y) dx dy & \rightarrow \int_0^1 \int_0^1 f(x,y) dx dy & [\int_0^1 \int_0^1 f(x,y) dx dy] \end{array}$$

最後の例では, 積分記号の間隔を狭めるために $\!$ を用いています¹⁰⁰). \displaystyle の数式での積分記号の間隔を調節するには, $\!$ を複数 (2, 3 個程度) 用いるとよいでしょう¹⁰¹). なお, §20 で紹介した \phantom , \hphantom , \vphantom は数式中で用いることもできます.

34. 定理型の環境

ここでは, “定理” のようなものを記述するための環境を定義するためのコマンド \newtheorem について説明します. \newtheorem コマンドは, ソースファイルのプリアンブルにおいて次の書式で用います¹⁰²).

- (1) $\newtheorem{\langle env_name \rangle}{\langle title_string \rangle}$
- (2) $\newtheorem{\langle env_name \rangle}{\langle title_string \rangle}[\langle parent_counter \rangle]$
- (3) $\newtheorem{\langle env_name \rangle}{\langle title_string \rangle}[\langle sharing_env \rangle]$

ただし, $\langle env_name \rangle$ は新しく定義する環境の名称で, $\langle title_string \rangle$ は $\langle env_name \rangle$ 環境の見出しに用いる文字列です. また, $\langle parent_counter \rangle$ は (section, subsection などの) カウンタの名称で, $\langle sharing_env \rangle$ はすでに \newtheorem コマンドによって定

¹⁰⁰)なお, この例の積分の記述の “ dx ”, “ dy ” では “ d ” を数式イタリックで記述していますが, それが “本来の (意味の上では正しい) 表記です (数学ではそのように記述します). 数学以外では “ dx ” などの “ d ” を直立体で記述することもあるようですが, そのような分野ごとの慣例については読者自身で確認してください.

¹⁰¹) \amsmath パッケージを用いた場合, 2 重積分・3 重積分の記述を行うためのコマンド \iint , \iiint も利用できます.

¹⁰²)本文中で用いてもエラーにはなりませんが, 文書全体にわたって適用する定義等は原則的にはプリアンブルで行います.

義された環境の名称です．また，本節では `\newtheorem` で定義した環境を“定理型の環境”と呼ぶことにします．

上記の書式 (1)–(3) は $\langle env_name \rangle$ 環境の番号の形式を次のように定めます．

- 書式 (1) の場合: $\langle env_name \rangle$ 環境の番号は単に 1, 2, ... という形式で出力されます．
- 書式 (2) の場合: $\langle env_name \rangle$ 環境の番号は $\langle parent_counter \rangle$ の番号を前に添えて出力されます．例えば, $\langle parent_counter \rangle$ が section の場合, §3 にある $\langle env_name \rangle$ 環境たちには, 3.1, 3.2, ... のように番号がつきます．
- 書式 (3) の場合: $\langle env_name \rangle$ 環境の番号と $\langle sharing_env \rangle$ 環境には同じ形式の通し番号がつきます．例えば,

```
\newtheorem{Thm}{Theorem}
\newtheorem{Lem}[Thm]{Lemma}
```

のように Thm 環境と Lem 環境を定義した場合, 番号 1 の Lem 環境の直後の Thm 環境の番号は 2 になります．

注意 34.1 この注意¹⁰³⁾は, 本稿のソースファイルのプリアンブルで

```
\newtheorem{Rem}{注意}[section]
```

のように定義した Rem 環境を用いて作成しています．特に, この環境の番号は “34.1” のようになっていることを確認するとよいでしょう．

書式 (2) の場合 $\langle parent_counter \rangle$ の番号の番号の直後の区切りとしては(デフォルトでは)ピリオドが用いられます．これを他の記号に取り換える場合には, `\@thmcountersep` というコマンドの定義を変更します．例えば,

```
\makeatletter
\def\@thmcountersep{-}
\makeatother
```

という定義をプリアンブル(で `\newtheorem` を用いる前)に入れると, 区切り記号はハイフンになって, 1-1, 1-2, ..., 2-1, 2-2, ... という形式で番号が出力されるようになります．

また, 定理型の環境はオプションをつけて用いることができます．例えば, 次の注意 34.2 は,

```
\begin{Rem}[定理型の環境のオプションについての注意]
```

のようにオプションをつけた Rem 環境です．

¹⁰³⁾これは, `\newtheorem` の説明のために入れた Rem 環境です．この注意と次の注意の体裁は本稿の他の注意の体裁とは異なっていますが, それはその 2 個の注意自身が “説明用” の例になっていることによります．

注意 34.2 (定理型の環境のオプションについての注意) 定理型の環境のオプションに `\cite[Theorem 6.3]{MH96}` のような、文字を含む文字またはコマンドの列を与える場合には、

```
\begin{Thm}[\cite[Theorem 6.3]{MH96}]
```

のようにオプション (の中身) を `{,}` で囲んでください。単に

```
\begin{Thm}[\cite[Theorem 6.3]{MH96}]
```

と記述すると、Thm 環境のオプションは “`\cite[Theorem 6.3]`” だけであるものとして扱われ、(`\cite` の処理の際に) エラーが生じます。

なお、 \LaTeX のデフォルトでは、定理型の環境の中の欧文のテキストはイタリック体で出力されます。これを (直立の) ローマン体に変更する場合には、環境の始まりの `\begin{...}` の直後に `\normalfont` と書き込んでください。(他の書体にする場合には、`\normalfont` の後にさらに適当な書体変更コマンドを追加してください。)

また、すべての定理型の環境について環境内のテキストを (直立の) ローマン体で出力するように変更しても構わない場合には、theorem パッケージを用いて (プリアンブルに `\usepackage{theorem}` という記述を入れて)、`\theorembodyfont{\rmfamily}` という指定を行うこともできます。一方、amsthm パッケージを用いると `\theoremstyle` コマンドによって定理型環境の見出しおよび本文の形式を “定義”、“定理” などのために用意されている形式の中から選ぶことができるようになります。(ここでは、これらのパッケージについては深入りしません。)

注意 34.3

環境 $\langle env \rangle$ を `\newtheorem` コマンドを用いて定義した場合、(他の環境と共通の番号を用いる指定をしていない場合には) $\langle env \rangle$ という名前のカウンタが定義されます。このカウンタの出力形式 (`\the\langle env \rangle` という名前のコマンドで指定されます) を変更すると、 $\langle env \rangle$ 環境の番号の形式が変わります。例えば、

```
\newtheorem{Thm}{Theorem}
\def\theThm{\Alph{Thm}}
```

のように定義すると、Thm 環境には、“Theorem A”、“Theorem B”... のような見出しがつけます。

一方、Thm 環境を、

```
\newtheorem{Thm}{Theorem}
```

のように定義したときに、見出しが “Theorem 1.”、“Theorem 2.”... のように最後にピリオドがついた形で出力されるように変更するのは少し面倒です。(単に、`\def\theThm{\arabic{Thm}.}` のように定義すると、Thm 環境の番号を相互参照した場合にも番号にピリオドがついてしまいます。)この場合には、`\@begintheorem`、`\@opargbegintheorem` というコマンドの定義を変更します。ファイル `latex.ltx` には、

```
\def\@begintheorem#1#2{\trivlist
  \item[\hskip \labelsep{\bfseries #1\ #2}]\itshape}
\def\@opargbegintheorem#1#2#3{\trivlist
  \item[\hskip \labelsep{\bfseries #1\ #2\ (#3)}]
  \itshape}
```

という定義があります．ここで，パラメータ #1-#3 にはそれぞれ，

- #1: タイトル文字列 (`\newtheorem{...}{\langle title \rangle}` の `\langle title \rangle`)
- #2: 環境の番号
- #3: 環境のオプション文字列 (`\begin{...}[\langle option \rangle]` の `\langle option \rangle`)

があてはまるので，ここでは，パラメータ #2 の前後に適当な記号を入れればよいわけですね．例えば，

```
\makeatletter
\def\@begintheorem#1#2{\trivlist
  \item[\hskip \labelsep{\bfseries #1\ #2.}]\itshape}
\def\@opargbegintheorem#1#2#3{\trivlist
  \item[\hskip \labelsep{\bfseries #1\ #2.\ (#3)}]
  \itshape}
\makeatother
```

という定義をプリアンブルに入れば，Thm 環境を

```
\newtheorem{Thm}{Theorem}
```

のように定義したとき，見出しは “Theorem 1.”， “Theorem 2.” ... のようになります．

なお，`\@begintheorem`，`\@opargbegintheorem` の定義の最後にある `\itshape` を取り除くと，定理型の環境の中のテキストも環境の外のテキストで用いられている書体で出力されるようになります．

35. 大規模な文書の作成

多くの章・節をもつような大規模な文書を書く場合，文書全体を一つのファイルにするのではなく，章ごとまたは節ごとに一つのファイルにすると，編集の効率がよくなります¹⁰⁴⁾．また，ソースファイルを分割して修正したところだけをコンパイルできるようにしておく，修正箇所の仕上がりを確認するのに便利です．そこで，ソースファイルを分割するための方法について説明します．

ソースファイルを分割するには，まず，次の 2 種類のファイルを用意します．

- (1) 次のような，プリアンブルと `\begin{document}`，`\end{document}` だけを記述したファイル

¹⁰⁴⁾ ファイルの管理の仕方が悪ければ，かえって効率が悪くなるかもしれませんが，ここではそういうことを問題にしても仕方がありません．


```

\documentclass[12pt]{jarticle}
%% プリアンプルには、後で必要な記述を補います。
\begin{document}
\end{document}

```

(2) 分割した本文だけを含むファイルたち

ここで、(2) で作る個々のファイルには、`\documentclass` や `\begin{document}`、`\end{document}` は要りません。以下、(2) で作ったファイルのファイル名は `sec1.tex`、`sec2.tex`、`...`、`secN.tex` であるものとして説明します。

次に、(1) で作ったファイルの `\begin{document}` と `\end{document}` の間の部分に、`\input` または `\include` によるファイルを読み込む指定を入れます。具体的には

```

\documentclass[12pt]{jarticle}
\begin{document}
\input{sec1}
\input{sec2}
.....
\input{secN}
\end{document}

```

または

```

\documentclass[12pt]{jarticle}
\begin{document}
\include{sec1}
\include{sec2}
.....
\include{secN}
\end{document}

```

のようにします。あとは、この `\input` または `\include` を書き込んだファイルをコンパイルすれば、ファイル `sec1.tex`、`sec2.tex`、`...`、`secN.tex` の中身を並べたものを本文とする文書ができます。

ここで、`\input`、`\include` は次のような意味のコマンドです。

- `\input{<filename>}`: ファイル `<filename>` を読み込みます。ただし、`<filename>` が `<file>.tex` (`<file>` はピリオドを含まない文字列) という形の文字列である場合には、拡張子を省略しても構いません。
- `\include{<filename>}`: 改ページを行った後、ファイル `<filename>.tex` を読み込みます。`\input` とは異なり、読み込ませるファイルの拡張子 `.tex` は (ピリオドも含めて) 省略しなければなりません。(なお、`\documentclass` のオプションに `twoside` を用いているときには、ファイル `<filename>.tex` の中身が奇数ページから書き始められるように、ファイルを読み込む前に `\cleardoublepage` を行います。)

ここで、`\input`、`\include` はファイルを読み込むコマンドなので、先程の例でのファイル `sec1.tex` などには `\documentclass` などは要らないわけです。というのも、も

し、ファイル `sec1.tex` にも `\documentclass` が書き込まれていると、`\input{sec1}` または `\include{sec1}` のところで `\documentclass` が現れます。しかし、コンパイルするファイルの先頭でもすでに `\documentclass` が用いられているので、ファイル `sec1.tex` の中の `\documentclass` はエラーを引き起こします。同様の理由で、分割した各々のファイルの中に、`\begin{document}`、`\end{document}` を書き込むと不都合があります。

最後に、分割したファイルたちの一部だけをコンパイルする方法を説明します。まず、`\input` を用いた場合には、

```
\documentclass[12pt]{jarticle}
\begin{document}
\input{sec1}
%\input{sec2}
%.....
%\input{secN}
\end{document}
```

のように、コンパイルしたくないファイルたち（に対する `\input`）の前に `%` をつけてコメントアウトします。上記の例では、`\input{sec1}` 以外をすべてコメントにしているので、ファイル `sec1.tex` の部分だけをコンパイルします。

一方、`\include` を用いている場合には、

```
\documentclass[12pt]{jarticle}
\includeonly{sec1,secN}
\begin{document}
\include{sec1}
\include{sec2}
.....
\include{secN}
\end{document}
```

のように、コンパイルしたいファイルのファイル名のリストを `\includeonly` というコマンドを用いて与えます（`\includeonly` はブリアンブルで用います）。上記の例では、ファイル `sec1.tex` と `secN.tex` の部分をコンパイルします。

36. 索引の作成

\LaTeX （のデフォルトの機能）だけを用いるのでは索引を作成することはできませんが、 \LaTeX と連携して索引を作成するために用いることができる `mendex`（あるいは `makeindex`¹⁰⁵⁾）というソフトウェアが用意されています¹⁰⁶⁾。そこで、本節では `mendex` を用いた索引の作成方法の基礎を説明します。

¹⁰⁵⁾`mendex` は `makeindex` を日本語対応にしたものです。

¹⁰⁶⁾実は、`mendex` または `makeindex` は“汎用的”な索引作成ツールで、 \LaTeX 以外と連携させることもできます。ただし、`mendex` または `makeindex` のデフォルトの設定は \LaTeX との連携を意識したものになっています。

36.1. 索引を作成する方法の概要

索引を作成する場合、 \LaTeX 文書の側では次の処理を行います。

- プリアンブルで `makeidx` パッケージを読み込みます。すなわち、プリアンブルに `\usepackage{makeidx}` という記述を入れます。
- プリアンブルに “`\makeindex`” という記述を入れます。
- 索引項目が現れた位置で `\index` というコマンドを用い、索引項目を指定します。（`\index` の使用法は後述します。）
- 索引を出力したい位置に “`\printindex`” と記述します。

これらの準備を行った後、 \LaTeX および `mendex` を用いて次のように処理します。

- (1) ソースファイルを \LaTeX で処理します。その結果、索引項目を出力したファイル（`idx` ファイルといいます¹⁰⁷⁾）が作成されます。なお、`idx` ファイルには索引項目が（概ね）出現順に書き出されているだけなので、そのままでは索引としては使えません。
- (2) (1) で作成した `idx` ファイルを `mendex` で処理します。ファイル `filename.idx` を処理する場合、最も単純にはシェル¹⁰⁸⁾のコマンドプロンプトから “`mendex filename`” と入力します¹⁰⁹⁾。その結果、並べ替えなどの処理を済ませた “実際に使用することができる索引” を書き出したファイル（`ind` ファイルといいます¹¹⁰⁾）が作成されます。なお、`ind` ファイルの中身は（`mendex` のデフォルトでは）“`theindex` 環境” になっています¹¹¹⁾。
- (3) (2) のように `ind` ファイルを作成した後、再びソースファイルを \LaTeX で処理します。その結果、`\printindex` コマンドの位置で `ind` ファイルが読み込まれ、索引が出力されます。

例えば、次のような、1 ページ目に `\index{first}` という索引項目があり、3 ページ目に `\index{third}` という索引項目がある文書を作成して、その文書に対して上記の操作を行ってみてください。

```
\documentclass{article}
\usepackage{makeidx}
\makeindex
\begin{document}
first\index{first}\newpage
second \newpage
```

¹⁰⁷⁾ファイル `filename.tex` を処理した場合、拡張子を “`.idx`” に変えた `filename.idx` という名称のファイルに索引項目が書き出されます。

¹⁰⁸⁾例えば、Windows 環境では “コマンドプロンプト”（以前の “MS-DOS プロンプト”）の類になります。

¹⁰⁹⁾ \TeX 用の統合環境の類のメニューに追加しておくのもよいでしょう。ただし、 \TeX 関連ソフトウェアを用いる場合には、いずれ “複数のオプション指定を取り換えて” 使用するようになります。そのような場合のために CUI (character user interface) 操作にも慣れておくことをお勧めします。

¹¹⁰⁾ファイル `filename.idx` を処理した場合、拡張子を “`.ind`” に変えた `filename.ind` という名称のファイルに索引そのものが書き出されます。

¹¹¹⁾`theindex` 環境は、`jarticle.cls` などのクラスファイルの中で定義されている一種の簡条書きの環境です。

```
third\index{third}\newpage
\printindex
\end{document}
```

そうすると、4 ページ目に次のような索引が出力されるはずです。

Index

first, 1

third, 3

36.2. 索引項目と“読み方”の指定

前項の例からもわかるように、索引項目が英単語の類ならば、単に索引項目を `\index` のパラメータにして `\index{first}` のように記述することができます。一方、(漢字を含む) 日本語の索引項目の場合には、“読み方”を与えないと索引項目を正しく整列することができません¹¹²⁾。そのような場合には `\index{しぜん@自然}` のように、読み方を索引項目自身の前に文字 “@” で区切って追加することができます。一般には、並べ替えに用いる“読み方”を $\langle key \rangle$ とするような索引項目 $\langle entry \rangle$ は

```
\index{\langle key \rangle @ \langle entry \rangle}
```

のように記述するわけです。(もちろん、 $\langle entry \rangle$ 自身を $\langle key \rangle$ として用いる場合には $\langle key \rangle @$ の部分を省略できます。) なお、この形式からわかるように、文字 “@” は索引項目と“読み方”を区切るために用いる“mendex の特殊文字”です¹¹³⁾。mendex の特殊文字が索引項目や“読み方”に含まれる場合、例えば、文字列 “@” を索引項目にするつもりで `\index{@}` と記述してもうまくいきません。このような“mendex の特殊文字”を含むような索引項目の扱い方は 36.5 項で説明します。

また、索引項目に \LaTeX のコマンドが含まれていても構いません。例えば、

```
\index{\LaTeX @ \LaTeX}
```

のように記述すると、項目 “ \LaTeX ” を索引に載せることができます(また、並べ替えの際には文字列 “ \LaTeX ” として扱われます¹¹⁴⁾)。同様に、`\index{#@\#}` のように記述すると文字 “#” を (文字列 “#” を“読み方”として) 索引に追加できます。

Note: 開き括弧 “{” と閉じ括弧 “}” の対応がとれていないような索引項目を記述する場合には、`\{` の代わりに `\textbraceleft` を用い、`\}` の代わりに `\textbraceright` を用いてください。

¹¹²⁾また、複数の読み方がある単語 (例えば “自然” は “しぜん” と “じねん” とも読めます) に対しては“索引項目として用いるときの読み方”を指定する必要が生じます。なお、漢字を含む索引項目に対しては、漢字を含まない“読み方”を指定しないと mendex での処理時にエラーが生じます。(mendex 用の“辞書ファイル”を用意したような場合にはこの限りではありませんが、ここでは深入りしません。)

¹¹³⁾実は、ユーザ自身が mendex の特殊文字を変更することもできます。本稿ではその点には深入りせず、mendex の特殊文字についてはデフォルトの設定が用いられているものとして説明します。詳しくは mendex について解説した文献 (例えば [6] の 11.2 項) を参照してください。

¹¹⁴⁾このような用法があるので、文字 “@” の前に置く文字列は読み方というよりもむしろ、並べ替え用の“ソートキー”というべきでしょう。

注意 36.1

先程の `\index{#\#}` という例では $\text{T}_{\text{E}}\text{X}$ の特殊文字の “#” を “読み方” に含めています。このような $\text{T}_{\text{E}}\text{X}$ の特殊文字を含むような索引項目は、他のマクロのパラメータ（例えば、`\section` の見出し文字列）の中には記述できないことがあります。（ $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ のコマンドを含むような索引項目についても事情は同様です。） $\text{T}_{\text{E}}\text{X}/\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ の特殊文字やコマンドを含むような索引項目は、他のコマンドのパラメータの中（のような `\verb` が使えないところ）では用いないようにするのが無難です¹¹⁵。例えば、`\section` の見出し文字列に含まれる語句を索引項目にするには、`\section{...}` の直後に `\index` を記述しても構いませんし、`\parbox` は `minipage` 環境に書き換えることができます。このようなわけで、多くの場合には他のコマンドのパラメータの中で `\index` を用いなくても済むように書き換えることができます。

36.3. 索引項目の階層化

また、`mendex` は索引項目の “階層化” もサポートしています。ここで “階層化” というのは、例えば “画像ファイル” という項目の下位の項目として “bmp ファイル” とした項目を用意する、という具合にある項目の下位の項目を作成することです。具体的には、

画像ファイル

 bmp ファイル, 1

 eps ファイル, 2

のような索引を作ることができます。

このような索引項目は、上位の項目と下位の項目を文字 “!” で区切って記述します。例えば、上記の例の 2 個の索引項目 “bmp ファイル”、“eps ファイル” に対しては次のように記述します。

```
\index{がぞうファイル@画像ファイル!bmpファイル}
\index{がぞうファイル@画像ファイル!epsファイル}
```

この例は、項目 $\langle entry_2 \rangle$ を別の項目 $\langle entry_1 \rangle$ の下位の項目にして

```
\index{\langle key_1 \rangle @ \langle entry_1 \rangle ! \langle key_2 \rangle @ \langle entry_2 \rangle }
```

のように記述しました（もちろん $\langle key_1 \rangle @$ 、 $\langle key_2 \rangle @$ の部分は適宜省略できます）。さらに項目 $\langle entry_3 \rangle$ を $\langle entry_2 \rangle$ の下位の項目にして

```
\index{\langle key_1 \rangle @ \langle entry_1 \rangle ! \langle key_2 \rangle @ \langle entry_2 \rangle ! \langle key_3 \rangle @ \langle entry_3 \rangle }
```

のように記述することもできます。このように `mendex` は第 3 階層までの階層化をサポートしています。

¹¹⁵ 実は、`\index` コマンドの処理には、`\verb` でも用いられるある処理が含まれています。これは、 $\text{T}_{\text{E}}\text{X}$ の特殊文字なども “読み方”（ソートキー）の記述などに使えるようにするための措置でやむを得ないものですが、その結果 `\verb` と同様の制限が生じています。

注意 36.2

ある索引項目の上位の項目に“読み方”が与えられている場合には、その読み方も省略しないで記述してください。例えば、項目“ $\mathrm{T}_E\mathrm{X}$ ”を“`\index{TeX@TeX}`”のように（読み方を“TeX”にして）索引に載せているものとします。このとき、記述“`\index{TeX@TeX!LaTeX@LaTeX}`”は、項目“ $\mathrm{L}^A\mathrm{T}_E\mathrm{X}$ ”を“読み方がTeXである項目 $\mathrm{T}_E\mathrm{X}$ ”の下位の項目として扱います。一方、単に“`\index{\TeX!LaTeX@LaTeX}`”と記述すると、これは項目“ $\mathrm{L}^A\mathrm{T}_E\mathrm{X}$ ”を“読み方が\TeXである項目 $\mathrm{T}_E\mathrm{X}$ ”の下位の項目にします。その結果、最初の“`\index{TeX@TeX}`”による項目“ $\mathrm{T}_E\mathrm{X}$ ”とは別の場所に項目“ $\mathrm{L}^A\mathrm{T}_E\mathrm{X}$ ”が現れてしまいます。実際、`mendex`は読み方だけが異なるような項目も別の項目として扱います。

36.4. 索引でのページ番号の書式の指定

例えば、`\index{TeX@TeX|textit}`という記述で項目“ $\mathrm{T}_E\mathrm{X}$ ”を索引に載せてみます。すると、その項目に対応するページ番号がイタリック体で表記されます。このように、索引項目に“文字|+文字列”という形式の記述を追加するとページ番号の書式を変更することができます。

一般的には、索引項目 $\langle entry \rangle$ に文字列“| $\langle command \rangle$ ”（ $\langle command \rangle$ は文字列）を追加して

$$\backslash\mathrm{index}\{\langle entry \rangle|\langle command \rangle\}$$

のように記述すると、この項目に対応するページ番号（ $\langle page \rangle$ ）とは

$$\backslash\langle command \rangle\{\langle page \rangle\}$$

という形式で出力されます¹¹⁶⁾。もちろん、これまでの例と同様に $\langle entry \rangle$ は“読み方”を伴っていても構いませんし、階層化されていても構いません。例えば、先程の`\index{TeX@TeX|textit}`の場合には、ページ番号部分が“`\textit{\langle page \rangle}`”という形式で出力されるわけです。

ただし、文字列“| $\langle \rangle$ ”または文字列“|”を追加した場合は例外で、これらは索引項目の“ページ範囲”の指定に用います。具体的には、ある索引項目 $\langle entry \rangle$ について`\index{\langle entry \rangle| $\langle \rangle$ }`と`\index{\langle entry \rangle|}`を組にして用いると、ページ番号部分が（例えば“12–34”のように）“`\index{\langle entry \rangle| $\langle \rangle$ }`を用いた箇所のページ番号から`\index{\langle entry \rangle|}`を用いた箇所のページ番号まで”という範囲指定を行った形式で記述されます。

Note: 上述の文字列“| $\langle \rangle$ ”または“|”を用いた範囲指定の意味からわかるように、“| $\langle \rangle$ ”と“|”は必ず対にして、“| $\langle \rangle$ ”を先に用いなければなりません。

また、ある項目 $\langle entry \rangle$ に対して別の項目 $\langle entry' \rangle$ を参照させる場合には、

$$\backslash\mathrm{index}\{\langle entry \rangle|\mathrm{see}\{\langle entry' \rangle\}\}$$

のように記述することができます。例えば、`\index{LaTeX@LaTeX|see{\TeX}}`のように項目“ $\mathrm{L}^A\mathrm{T}_E\mathrm{X}$ ”を索引に載せると、

¹¹⁶⁾これも、`mendex`のデフォルトの設定を用いている場合の話です。

\LaTeX , *see* \TeX

のように出力されます。

ここで、“*see*”の部分を変更するには `\seename` というコマンドを再定義します。例えばブリアンプル (の `\usepackage{makeidx}` 以降の部分) に

```
\def\seename{${\Longrightarrow$}
```

という記述を入れると、`\index{LaTeX@LaTeX|see{\TeX}}` に対応する項目は

$\text{\LaTeX} \implies \text{\TeX}$

のように変わります。この場合、“ \LaTeX ”の直後のコンマを消したいところですが、そのような調整については 36.6 項で扱います。

36.5. mendex の特殊文字を含むような索引項目

これまでに “@”, “!”, “|” という mendex の特殊文字を紹介しましたが、ここでは、これらの文字を含むような索引項目の記述法について説明します。索引項目やその“読み方”に mendex の特殊文字が含まれる場合には、その文字に文字 “n” を前置します。例えば、文字列 “@” を索引項目にする場合には `\index{"@"}` のように記述します。もちろん、文字 “n” 自身も“特殊文字の意味を打ち消す¹¹⁷⁾”という処理に用いる mendex の特殊文字です。したがって、文字 “n” 自身が索引項目に含まれる場合には、`\index{"n"}` (文字列 “n” を索引項目にする場合) のように文字 “n” を重ねます。

もう少し複雑な例を考えてみます。例えば、“ $n!$ ” (数式なので、“ $\$n!\$$ ”と記述しています) という索引項目を “n!” という“読み方”で索引に載せる場合、“読み方”と索引項目をそれぞれ次のように記述します。

- “読み方”: $n!$
- 索引項目: $\$n!\$$

したがって、この場合は `\index{"n!"@n!\$}` のように記述すればよいわけです。

Note: ただし、アクセント記号のコマンド “\” が索引項目に含まれる場合には、そのまま “\” と記述すればよく、“\” のようにする必要はありません。これは、文字 “\” も mendex の特殊文字であることによります¹¹⁸⁾。なお、“\” に対しては念入りにエスケープして “\” のように記述することもできます。文字列 “@”, “!”, “|” が索引項目などに含まれる場合には、念入りにエスケープして “\”, “\”, “\” のように記述するとよいでしょう。

¹¹⁷⁾“エスケープする” といいます。

¹¹⁸⁾文字 “\” も文字 “n” と同様に、特殊文字のエスケープに用いられます。ただし、文字 “n” でエスケープした場合、文字 “n” 自身は消えます (例えば、索引項目の中の “n” は文字列 “@” として扱われます)。一方、文字 “\” でエスケープした場合には、文字 “\” 自身も残ります (例えば、索引項目の中の “\” はコマンド “\” として扱われます)。

36.6. 索引の体裁の（簡単な）カスタマイズ

再び、本節の冒頭の例（下記の記述）を考えます．

```
\documentclass{article}
\usepackage{makeidx}
\makeindex
\begin{document}
first\index{first}\newpage
second \newpage
third\index{third}\newpage
\printindex
\end{document}
```

また、下記の 3 行からなるファイル（ファイル名は sample.ist にしてください）を作成し、上記の内容をもったソースファイルと同じディレクトリ（フォルダ）に置いてください．

```
delim_0 "\\leaders\\hbox{$\\cdot$}\\hfill "
delim_1 "\\leaders\\hbox{$\\cdot$}\\hfill "
delim_2 "\\leaders\\hbox{$\\cdot$}\\hfill "
```

そして、idx ファイルを mendex で処理する際に、“-s sample” というオプションも用いてください．すなわち、ファイル filename.idx を処理する場合には、

```
mendex -s sample filename
```

のように（コマンドプロンプトに対して）入力します．

索引の作成操作を以上のように変更すると、索引の体裁が次のように変わることがわかります．

Index

```
first ..... 1
third..... 3
```

この例からわかるように、mendex によって生成される索引の体裁は、ここで用いた sample.ist のような外部ファイル（および mendex のオプション指定）を用いて変更することができます¹¹⁹）。また、*filename* という名称の外部ファイルを用いて索引を作成する場合には、mendex を用いる際に “-s *filename*” というオプションを使用すればよいわけですから（なお、*filename* の拡張子が “.ist” のみである場合には拡張子を省略できます）。なお、先程の sample.ist のような、索引の体裁の指定に用いるファイルを ist ファイルといいます．ist ファイルの拡張子は、“.ist” にするのが通例です．

例えば、先程の sample.ist で設定している delim_0 というのは、最上位の索引項目とページ番号の区切りとして用いられる文字列です．同様に、delim_1、delim_2 は第 2

¹¹⁹もちろん、複数の索引を作成しようとしたり複雑な体裁の索引を作成しようとする、マクロ作成に関する知識をそれなりに必要とします．

階層または第 3 階層の索引項目とページ番号の区切りとして用いられる文字列です¹²⁰⁾。本項の例では、`delim_0` などとして、文字列 “`\leaders\hbox{\cdot}\hfill`” を用いています。ファイル `sample.ist` の中では文字 “`\`” を重ねていますが、これは一般的なプログラミング言語における文字列の表記法に準じているからです¹²¹⁾。

なお、`ist` ファイルで設定できる項目は多岐にわたるので、詳しくは `mendex` について解説した文献を参照してください。ここでは、典型的な例のいくつかを扱うにとどめます。また、以下の説明では、`version 2.5` 以降の `mendex` を用いているものとして説明します。古い版の `mendex` では以下の説明で用いている項目のいくつかがサポートされていないので、最新の `mendex` を用いることをお勧めします。

最初に、36.4 項の末尾で提示した、“`|see{...}`”を用いたときに、索引項目の直後のコンマを消す”という問題を考えます。これに対しては、`\seename` を

```
\def\seename{\unskip\ $\Longrightarrow$}
```

のように定義し、さらに、索引作成時に上述の `sample.ist` を用いるとよいでしょう。

また、下記のような行を含む `ist` ファイルを用いると、索引項目の頭文字が変わるごとに頭文字を見出しにすることができます (十分に多くの索引項目を含むソースファイルを作って実験してみてください)。

```
lethead_flag 1
```

この指定を用いると、見出しとして出力される頭文字のうち、アルファベットは大文字になっています。一方、“`lethead_flag 1`” の数値 1 を `-1` に変更すると、見出しのアルファベットは小文字になります。また、和文項目の頭文字はカタカナになっていますが、これをひらがなにするには、

```
letter_head 2
```

という行を `ist` ファイルに追加します。

なお、`ist` ファイル中の `lethead_flag` 項目の値が 0 でない場合、数字や記号で始まる索引項目の見出しは “`Symbols`” または “`symbols`” という文字列になります (`lethead_flag` の符号によって決まります)。これを変更するには、`ist` ファイル中で `symhead_positive` (`lethead_flag` の値が正の場合) または `symhead_negative` (`lethead_flag` の値が負の場合) を設定します。例えば、(`lethead_flag` の値が正の場合) `ist` ファイルに

```
symhead_positive "記号"
```

という行を追加すると、数字や記号で始まる項目の見出し文字列が “`記号`” になります。

索引中の見出し (頭文字) 部分の体裁を変更したい場合は、`ist` ファイルの中で `lethead_prefix` (見出しに前置する文字列)、`lethead_suffix` (見出しに後置する

¹²⁰⁾ `ist` ファイルの方では 0 から数えているので、階層の番号と `delim_0` などの番号がずれていることに注意してください。

¹²¹⁾ 同様に、`ist` ファイル中の文字列において改行文字・タブ文字を表すには、それぞれ “`\n`”、“`\t`” と記述します。

文字列)を設定します。例えば,(lethead_flag を 0 でない値に設定した)ist ファイルに

```
lethead_prefix "\n\\centerline{\\bfseries  "
lethead_suffix " }\\par\\nobreak"
```

という 2 行を追加すると,見出しを “ ” で挟んで太字で表記したものを行の中央に置くことができます。書体の指定や飾りに用いた文字などは読者自身で適宜変更するとよいでしょう。

また,和文項目の見出しとしては,デフォルトでは“ア”,“イ”,...のように 50 音すべてが用いられます。これを“ア”,“カ”,...のように“行”ごとに見出しを出すように変更するには,mendex の -g オプションを用います。

補遺 A. amssymb パッケージが提供する記号

ここでは,amssymb パッケージによって定義されている各種の数式用記号を紹介します。

A.1. 2 項演算子

amssymb パッケージが提供する 2 項演算子の表です。

<code>\dotplus</code>	→ $\dot{+}$	<code>\smallsetminus</code>	→ \smallsetminus	<code>\centerdot</code>	→ \cdot
<code>\Cap</code>	→ \cap	<code>\Cup</code>	→ \cup	<code>\intercal</code>	→ \intercal
<code>\barwedge</code>	→ $\bar{\wedge}$	<code>\veebar</code>	→ \veebar	<code>\doublebarwedge</code>	→ $\overline{\wedge}$
<code>\boxdot</code>	→ \boxdot	<code>\boxminus</code>	→ \boxminus	<code>\circleddash</code>	→ \odot
<code>\boxplus</code>	→ \boxplus	<code>\boxtimes</code>	→ \boxtimes	<code>\circledast</code>	→ \otimes
<code>\ltimes</code>	→ \ltimes	<code>\rtimes</code>	→ \rtimes	<code>\circledcirc</code>	→ \odot
<code>\leftthreetimes</code>	→ \leftthreetimes	<code>\rightthreetimes</code>	→ \rightthreetimes	<code>\divideontimes</code>	→ \div
<code>\curlywedge</code>	→ \curlywedge	<code>\curlyvee</code>	→ \curlyvee		

A.2. 関係演算子

amssymb パッケージが提供する(通常の)関係演算子の表です。

<code>\leqq</code>	→ \leq	<code>\leqslant</code>	→ \leq	<code>\eqslantless</code>	→ \leq
<code>\lessssim</code>	→ \lesssim	<code>\lessapprox</code>	→ \lesssim	<code>\lessdot</code>	→ \leq
<code>\lll</code>	→ \lll	<code>\lessgtr</code>	→ \leq	<code>\lesseqgtr</code>	→ \leq
<code>\lesseqqgtr</code>	→ \leq	<code>\vartriangleleft</code>	→ \triangleleft	<code>\trianglelefteq</code>	→ \trianglelefteq
<code>\geqq</code>	→ \geq	<code>\geqslant</code>	→ \geq	<code>\eqslantgtr</code>	→ \geq
<code>\gtrsim</code>	→ \gtrsim	<code>\gtrapprox</code>	→ \gtrsim	<code>\gtrdot</code>	→ \geq
<code>\ggg</code>	→ \ggg	<code>\gtrless</code>	→ \geq	<code>\gtreqless</code>	→ \geq
<code>\gtreqqless</code>	→ \geq	<code>\vartriangleright</code>	→ \triangleright	<code>\trianglerighteq</code>	→ \trianglerighteq
<code>\subteqq</code>	→ \subseteq	<code>\Subset</code>	→ \subseteq	<code>\sqsubset</code>	→ \sqsubset

<code>\supseteq</code>	\supseteq	<code>\Supset</code>	\supsetneq	<code>\sqsupset</code>	\sqsupset
<code>\preccurlyeq</code>	\preccurlyeq	<code>\curlyeqprec</code>	\preccurlyeq	<code>\precsim</code>	\precsim
<code>\precapprox</code>	\precapprox	<code>\blacktriangleleft</code>	\blacktriangleleft	<code>\therefore</code>	\therefore
<code>\succcurlyeq</code>	\succcurlyeq	<code>\curlyeqsucc</code>	\succcurlyeq	<code>\succsim</code>	\succsim
<code>\succapprox</code>	\succapprox	<code>\blacktriangleright</code>	\blacktriangleright	<code>\because</code>	\because
<code>\approx</code>	\approx	<code>\backsim</code>	\backsim	<code>\backsimeq</code>	\backsimeq
<code>\thicksim</code>	\sim	<code>\thickapprox</code>	\thickapprox	<code>\varpropto</code>	\propto
<code>\risingdotseq</code>	\risingdotseq	<code>\doteqdot</code>	\doteqdot	<code>\fallingdotseq</code>	\fallingdotseq
<code>\bumpeq</code>	\bumpeq	<code>\Bumpeq</code>	\Bumpeq	<code>\smallsmile</code>	\smallsmile
<code>\eqcirc</code>	\eqcirc	<code>\circeq</code>	\circeq	<code>\smallfrown</code>	\smallfrown
<code>\triangleq</code>	\triangleq	<code>\shortmid</code>	\shortmid	<code>\shortparallel</code>	\shortparallel
<code>\vDash</code>	\vDash	<code>\Vdash</code>	\Vdash	<code>\Vvdash</code>	\Vvdash
<code>\between</code>	\between	<code>\backepsilon</code>	\backepsilon	<code>\pitchfork</code>	\pitchfork

次は，“否定の”関係演算子の表です．

<code>\nless</code>	\nless	<code>\nleq</code>	\nleq	<code>\nleqslant</code>	\nleqslant
<code>\nleqq</code>	\nleqq	<code>\lneq</code>	\lneq	<code>\lneqq</code>	\lneqq
<code>\lvertneqq</code>	\lvertneqq	<code>\lnsim</code>	\lnsim	<code>\lnapprox</code>	\lnapprox
<code>\ngtr</code>	\ngtr	<code>\ngeq</code>	\ngeq	<code>\ngeqslant</code>	\ngeqslant
<code>\ngeqq</code>	\ngeqq	<code>\gneq</code>	\gneq	<code>\gneqq</code>	\gneqq
<code>\gvertneqq</code>	\gvertneqq	<code>\gnsim</code>	\gnsim	<code>\gnapprox</code>	\gnapprox
<code>\nprec</code>	\nprec	<code>\npreceq</code>	\npreceq	<code>\precnsim</code>	\precnsim
<code>\precnapprox</code>	\precnapprox	<code>\ntriangleleft</code>	\ntriangleleft	<code>\ntrianglelefteq</code>	\ntrianglelefteq
<code>\nsucc</code>	\nsucc	<code>\nsucceq</code>	\nsucceq	<code>\succnsim</code>	\succnsim
<code>\succnapprox</code>	\succnapprox	<code>\ntriangleright</code>	\ntriangleright	<code>\ntrianglerighteq</code>	\ntrianglerighteq
<code>\nsubseteq</code>	\nsubseteq	<code>\subsetneq</code>	\subsetneq	<code>\varsubsetneq</code>	\varsubsetneq
<code>\nsupseteq</code>	\nsupseteq	<code>\supsetneq</code>	\supsetneq	<code>\varsupsetneq</code>	\varsupsetneq
<code>\nsubseteqq</code>	\nsubseteqq	<code>\subsetneqq</code>	\subsetneqq	<code>\varsubsetneqq</code>	\varsubsetneqq
<code>\nsupseteqq</code>	\nsupseteqq	<code>\supsetneqq</code>	\supsetneqq	<code>\varsupsetneqq</code>	\varsupsetneqq
<code>\nmid</code>	\nmid	<code>\nparallel</code>	\nparallel	<code>\nsim</code>	\nsim
<code>\nshortmid</code>	\nshortmid	<code>\nshortparallel</code>	\nshortparallel	<code>\ncong</code>	\ncong
<code>\nvDash</code>	\nvDash	<code>\nVDash</code>	\nVDash		
<code>\nVdash</code>	\nVdash	<code>\nVDash</code>	\nVDash		

A.3. 矢印類

各種の矢印類の表です．

<code>\dashrightarrow</code>	\dashrightarrow	<code>\circlearrowright</code>	\circlearrowright	<code>\looparrowright</code>	\looparrowright
<code>\dashleftarrow</code>	\dashleftarrow	<code>\circlearrowleft</code>	\circlearrowleft	<code>\looparrowleft</code>	\looparrowleft
<code>\curvearrowright</code>	\curvearrowright	<code>\Rrightarrow</code>	\Rrightarrow	<code>\rightrightarrows</code>	\rightrightarrows
<code>\curvearrowleft</code>	\curvearrowleft	<code>\Lleftarrow</code>	\Lleftarrow	<code>\leftleftarrows</code>	\leftleftarrows

<code>\rightleftarrows</code>	\rightleftarrows	<code>\rightleftharpoons</code>	\rightleftharpoons	<code>\upharpoonright</code>	\upharpoonright
<code>\leftrightarrows</code>	\leftrightarrows	<code>\leftrightharpoons</code>	\leftrightharpoons	<code>\upharpoonleft</code>	\upharpoonleft
<code>\twoheadrightarrow</code>	\twoheadrightarrow	<code>\rightarrowtail</code>	\rightarrowtail	<code>\downharpoonright</code>	\downharpoonright
<code>\twoheadleftarrow</code>	\twoheadleftarrow	<code>\leftarrowtail</code>	\leftarrowtail	<code>\downharpoonleft</code>	\downharpoonleft
<code>\Rsh</code>	\Rsh	<code>\rightsquigarrow</code>	\rightsquigarrow	<code>\upuparrows</code>	\upuparrows
<code>\Lsh</code>	\Lsh	<code>\leftrightsquigarrow</code>	\leftrightsquigarrow	<code>\downdownarrows</code>	\downdownarrows
<code>\nrightarrow</code>	\nrightarrow	<code>\nRightarrow</code>	\nrightarrow	<code>\nleftrightarrow</code>	\nleftrightarrow
<code>\nleftarrow</code>	\nleftarrow	<code>\nLeftarrow</code>	\nleftarrow	<code>\nLeftrightarrow</code>	\nLeftrightarrow

A.4. その他の記号

一般の数式用記号の表です。(ただし、§27 で紹介したものは省略しています。)

<code>\vartriangle</code>	\triangle	<code>\triangledown</code>	\triangledown	<code>\square</code>	\square
<code>\blacktriangle</code>	\blacktriangle	<code>\blacktriangledown</code>	\blacktriangledown	<code>\blacksquare</code>	\blacksquare
<code>\lozenge</code>	\lozenge	<code>\blacklozenge</code>	\blacklozenge	<code>\bigstar</code>	\bigstar
<code>\measuredangle</code>	\measuredangle	<code>\sphericalangle</code>	\sphericalangle	<code>\eth</code>	\eth
<code>\nexists</code>	\nexists	<code>\Finv</code>	\Finv	<code>\Game</code>	\Game
<code>\hslash</code>	\hslash	<code>\Bbbk</code>	\Bbbk	<code>\backprime</code>	\backprime
<code>\varnothing</code>	\varnothing	<code>\complement</code>	\complement	<code>\circledS</code>	\circledS
<code>\diagup</code>	\diagup	<code>\diagdown</code>	\diagdown	<code>\multimap</code>	\multimap
<code>\digamma</code>	\digamma	<code>\varkappa</code>	\varkappa		

補遺 B. テキストの色付け

ここでは、“色”に関する処理を行うために用いる color パッケージについて、基本的なことを説明します。なお、本節で説明している機能を用いるには、プリアンブルに

```
\usepackage[dvips]{color}
```

のような color パッケージを読み込む指定を入れる必要があります。ここで、オプションの“dvips”の部分はお使いの dviware に応じて変更してください¹²²⁾。また、色の指定を“ForestGreen”(■という色に対応します¹²³⁾)のような多彩な色名を用いた指定を行いたい場合には、usenames オプションも追加するとよいでしょう。

ここで、color パッケージに関しては (graphicx パッケージの場合と同様に) “color パッケージで提供されている機能は、T_EX 自身の機能ではない”という点に注意してください。本節で説明するコマンドたちは、色の取り扱いに関する記述を dvi ファイルに埋め込み、それを dviware に解釈させます。したがって、お使いの dviware によっては、この節で説明している例の一部(場合によっては全部)が正しく表示できないことがあります(これも graphicx パッケージの場合と同様です)。

¹²²⁾概ね dviware の名称になりますが、“dvips”にすれば PostScript 経由で表示・印刷ができます。

¹²³⁾ただし、言うまでもないことですが、同じ色指定を行ったとしても実際に出力される色は表示・出力環境によって変わってしまうことがあります。(これは、T_EX を用いる場合に限ったことではありません。)

B.1. 色指定の方法

色をつける対象 $\langle object \rangle$ に、 $\langle color \rangle$ という名称の色をつける場合には、`\color` または `\textcolor` を用いて

`{\color{\langle color \rangle}\langle object \rangle}` または `\textcolor{\langle color \rangle}{\langle object \rangle}`

のように記述します。例えば、

`{\color{blue}\rule{1em}{1ex}blue}` and `\textcolor{green}{green}`

という記述からは

blue and green

という出力が得られます。この例からわかるように、色をつける対象は文字列以外の罫線のようなものであっても構いません¹²⁴⁾。

今の例では、色の名称を用いましたが、`rgb` 値を直接指定するような場合には、

`{\color[\langle model \rangle]{\langle parameters \rangle}\langle object \rangle}` または
`\textcolor[\langle model \rangle]{\langle parameters \rangle}{\langle object \rangle}`

(ただし、 $\langle model \rangle$ はカラーモデルの名称、 $\langle parameters \rangle$ は色を表すパラメータ) という形式で指定することができます。例えば、

`{\color[gray]{.7}light-gray}` and `\textcolor[rgb]{.4,0,1}{violet}`

という入力に対しては

light-gray and violet

という出力が得られます¹²⁵⁾。

なお、パラメータの与え方については、次の点に注意してください。

- `rgb` 指定のように複数の値を必要とする場合には、個々の値をコンマで区切って並べます。
- 各パラメータは 0 と 1 の間の実数値で記述します。例えば、`rgb` 指定の場合には各パラメータを 0 から 255 の範囲の整数値で考えることもありますが、`\color` または `\textcolor` のパラメータとして用いるときには 255 で割って 0 と 1 の間の値にします。(先程の例の後半は $\text{red} = 0.4 \times 255 = 102$, $\text{green} = 0$, $\text{blue} = 1 \times 255 = 255$ に相当します。)

¹²⁴⁾ただし、白黒(あるいはグレースケール)の画像を `\includegraphics` で取り込んだときに、その `\includegraphics` 部分に色指定を行っても画像の色は変わりません。画像の色を変えたい場合には画像自身を編集してください。

¹²⁵⁾この例では、`gray` カラーモデルを用いた場合、パラメータの値が 1 に近づくほど明るくなることに注意するとよいでしょう。

一般的なカラーモデルには `rgb`, `cmyk`, `gray` などがありますが, どのカラーモデルがサポートされるかはもちろん `dviware` に依存します¹²⁶).

また, ある色を適用している範囲に別の色を適用して色を取り換えることもできます. 例えば,

```
\textcolor{blue}{\underline{\textcolor{black}{patapata}}}
```

のような記述を行うと,

patapata

のように青い下線を引くことができます (内側の `\textcolor{black}{...}` の指定がなければ下線をつける文字列も青色で出力されることに注意してください).

B.2. 背景色の指定

文字列の背景色を指定するには, `\colorbox` コマンドを次のように用います.

```
\colorbox{<color>}{<text>}
```

ここで, `<color>` は背景色, `<text>` は背景色を変更した部分に記述するテキストです. また, カラーモデル `<model>` とパラメータ値 `<parameters>` による色指定を行って

```
\colorbox[<model>]{<parameters>}{<text>}
```

のように記述することもできます. 例えば,

```
\colorbox{cyan}{cyan},
\textcolor{blue}{[\colorbox{cyan}{cyan}]} and
\colorbox[gray]{.5}{\textcolor{white}{\sffamily gray}}
```

という記述からは

cyan, cyan and gray

という出力が得られます. なお, 今の例のうちの最初の 2 つのものからわかるように, `<text>` 部分の色は `\colorbox` の周囲で用いられている色が用いられます. したがって, “白抜き” のようなことをしたい場合には, 第 3 の例のように `<text>` 部分そのものにも色指定を適用します.

また, `\colorbox` での出力では `<text>` 部分よりも広い領域に背景色が設定されていますが, それは `<text>` 自身が占める領域を上下左右に `\fboxsep` だけ膨らませた領域になっています. したがって, `\fboxsep` の値を調節すると, 背景色の部分の広さを調整できます. 実際,

```
\colorbox{green}{green} and
{\setlength{\fboxsep}{1pt}\colorbox{green}{green}}
```

¹²⁶`\dvips(k)` を用いる場合には, これらのモデルはサポートされます. なお, `cmyk` モデルを用いる場合には, `cyan`, `magenta`, `yellow`, `black` のそれぞれの値を (やはり 0 と 1 の間の) 実数値で指定します.

という記述からは

 and 

という出力が得られます．

さらに，`\fbox` の拡張として，色つきの枠と（枠とは異なった）色の背景をもったテキストを作成するコマンド `\fcolorbox` も用意されています．これは，

```
\fcolorbox{<border color>}{<background color>}{<text>}
```

（`<border color>` は枠の色，`<background color>` は背景色，`<text>` は枠の中のテキスト）のように用います．カラーモデル `<model>` と枠の色のパラメータ `<border params>` と背景色のパラメータ `<background params>` による指定を

```
\fcolorbox[<model>]{<border params>}{<background params>}{<text>}
```

のように行うこともできます¹²⁷．例えば，

```
\fcolorbox{blue}{yellow}{\textcolor{red}{yellow}} and
{\setlength{\fboxrule}{1pt}\fcolorbox[gray]{.6}{.9}{gray}}
```

という記述からは

 and 

という出力が得られます．この例からもわかるように，`\fboxrule` の値を変更すると枠の太さが変わります．また，`\fbox` の場合と同様に `\fboxsep` の値を変更すると枠の中身と枠との間隔が変わります．

Note: ここでは，色に関する処理の基本的なことを扱いました．一方，“表の個々のセルに色をつける”といったことは `color` パッケージの機能ではできません．必要があれば `colortbl` パッケージなどを用いてください（“`LATEX` グラフィックスコンパニオン” [5] などが参考になるでしょう）．

参考文献

まず，“バイブル”的なものを挙げます．

- [1] Donald E. Knuth（鷲谷好輝訳），“改訂新版 `TEX` ブック”，アスキー（1992）．
- [2] Leslie Lamport，“*L^AT_EX: A Document Preparation System*”（2nd ed.），Addison-Wesley（1994）．

`LATEX 2ε` の定評のある入門書です．

- [3] 奥村晴彦，「改訂第 3 版」`LATEX 2ε` 美文書作成入門”，技術評論社（2004）．

¹²⁷この場合，カラーモデル `<model>` は枠の色と背景色の両方に適用されます．枠の色と背景色に別々のカラーモデルを用いることはできませんが，そのような指定を行う必要はないでしょう．

L^AT_EX 自身および各種のパッケージの総合的な解説です．なお，原書は第 2 版が出ています．

- [4] Michel Goossens, Frank Mittelbach, Alexsander Samarin (アスキー書籍編集部監訳), “The L^AT_EX コンパニオン”, アスキー (1998).

L^AT_EX 文書における“視覚的表現”に詳しいテキストです．

- [5] Michel Goossens, Sebastian Rahtz, Frank Mittelbach (鷺谷好輝訳), “L^AT_EX グラフィックスコンパニオン”, アスキー (2000).

L^AT_EX におけるマクロ作成の“教科書”です．

- [6] ページ・エンタープライゼズ株式会社, “L^AT_EX 2_ε【マクロ&クラス】プログラミング基礎解説”, 技術評論社 (2002).
(本田知亮氏と筆者が執筆し, 筆者が組版を担当したものです.)