

L^AT_EX 2_εまぐろの八衢

藤田 眞作

京都工芸繊維大学
工芸学部物質工学科

2004/07/30

(第 1 版)

LaTeX 2_ε まくろの八衢 ©(1995, 2004) 藤田 眞作

営利を目的とする転載はご遠慮ください。非営利の転載については、

1. 改変しないこと
2. 各ファイルやディレクトリー内容を分割しないこと

を条件に、原則として自由とします。ただし、転載した場合は、転載者、転載先（できれば、転載先のサービス範囲などの情報）、転載目的など、できる限りの情報ををメールでお知らせください。このドキュメントの内容の誤りなどによって起こった損害に対して、著者は一切の責任を負いません。

はしづみ

本オンラインマニュアルの旧版は、「 \LaTeX まくろの八衢」(第 1.00 版)として、1995/05/27 に、オンライン (@nifty (当時 NIFTY Serve) の FPRINT) で公表したものです。そのころ、著者は、富士写真フイルム(株)足柄研究所に勤務しており、論文執筆のために \LaTeX の利用を本格的におこないはじめた時期にあたります。 $\text{\TeX}/\text{\LaTeX}$ の発展史に沿っていえば、 $\text{\LaTeX}2.09$ の時代であり、当然旧版もこれに準拠していました。その後、 $\text{\LaTeX}2_\epsilon$ が主流となりましたが、本マニュアルはもともと \TeX の命令を中心に記述していますので、大部分の記述はそのままでも十分有効でした。著者の怠慢もありますが、これが旧版をそのままにしておいた第一の理由です。

$\text{\LaTeX}2_\epsilon$ の時代になり、さらに日本語版の $\text{p}\text{\LaTeX}2_\epsilon$ が発表されたあとは、本邦でも $\text{\TeX}/\text{\LaTeX}$ が広く使われるようになってきました。この状況をうけて、マクロの作成に関しても、たくさんの良書が刊行されています¹。本マニュアルが、その中で少しでも役に立つためのものでありつづけるには、最低限 $\text{\LaTeX}2_\epsilon$ に準拠するものには書き換える必要を感じはじめました。しかし、必要性を感じつつも、1997年に現職(京都工芸繊維大学)に転職したために、なかなか時間の余裕がなく、ここまで放置することになってしまいました。

初出より9年を経過し、さらには世の中も20世紀から21世紀に移り、これ以上放置できないところにきましたので、夏休みを利用して旧版を $\text{\LaTeX}2_\epsilon$ 準拠に書き換え、改訂をおこなうことにしました。改訂により大部になって読みにくくなってはもとも子もないので、 $\text{\LaTeX}2_\epsilon$ 準拠にすることを第一義としました。このため、内容については、旧版の大部分をそのまま受け継ぎ、 $\text{\LaTeX}2_\epsilon$ の仕様に基づいて最小限の変更をおこなったものになっています。

$\text{\LaTeX}2_\epsilon$ の出現により、本マニュアルの旧版で取り上げた $\text{\LaTeX}2.09$ の不十分であったところが改良されています。とくに、`\parbox` の定義が $\text{\LaTeX}2_\epsilon$ で変更されたことにより、 \LaTeX のロゴの出力異常がなくなったため、ロゴの改良マクロ自体の意味はなくなってしまいました。しかし、フォントに関する記述は歴史的な意味もあるので、 $\text{\LaTeX}2_\epsilon$ の現状に即した形に書き直して残してあります。ただし、フォントの選択に関しては、 $\text{\LaTeX}2_\epsilon$ が NFSS (New Font Selection Scheme) を採用したことに伴い、もうすこし厳密な説

¹ 著者自身も次のような解説書を刊行しています。参考にしていただければ幸いです。

- 藤田眞作 “化学者・生化学者のための \LaTeX —パソコンによる論文作成の手引”, 東京化学同人 (1993)
- 藤田眞作 “ \LaTeX まくろの八衢”, アジソン・ウェスレイ (1995)
- 藤田眞作 “ \LaTeX 本づくりの八衢”, アジソン・ウェスレイ (1996)
- 藤田眞作 “ $\text{\LaTeX}2_\epsilon$ 階梯”, アジソン・ウェスレイ (1996)
- 藤田眞作 “ $\text{\X}\text{\M}\text{\T}\text{\E}\text{\X}$ —Typesetting Chemical Structureal Formulas”, アジソン・ウェスレイ (1997)
- 藤田眞作 “続 $\text{\LaTeX}2_\epsilon$ 階梯・縦組編”, アジソン・ウェスレイ (1998)
- 藤田眞作 “ $\text{p}\text{\LaTeX}2_\epsilon$ 入門・縦横文書術”, ピアソンエデュケーション (2000)
- 藤田眞作 “ $\text{\LaTeX}2_\epsilon$ 階梯 (第2版)”, ピアソン・エデュケーション (2000)
- 藤田眞作 “ $\text{\LaTeX}2_\epsilon$ コマンドブック”, ソフトバンク (2003)

上記の著書はもちろん \LaTeX ($\text{\LaTeX}2_\epsilon$) で組版したものです。また、専門領域でも、 \LaTeX ($\text{\LaTeX}2_\epsilon$) で組版した著書を刊行しています。とくに最後にあげたものは、おびただしい数の化学構造式を含んでいますが、すべて $\text{\X}\text{\M}\text{\T}\text{\E}\text{\X}$ で組版したものです。ホームページのアドレスをあげておきましたのでご覧ください。

- S. Fujita “Symmetry and Combinatorial Enumeration in Chemistry”, Springer-Verlag (1991)
- S. Fujita “Computer-Oriented Representation of Organic Reactions”, Yoshioka Shoten (2001)
- S. Fujita “Organic Chemistry of Photography”, Springer-Verlag (2004)
- ホームページ <http://imt.chem.kit.ac.jp/fujita/fujitas/fujita.html>

明が必要だとも思います。これは、将来の課題として残っています。

改訂を機会に書名の $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ を $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ に改め、「 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ まくろの八衢」(第1版)としました。「八衢」は「やちまた」と読みますが、旧版の公表以後、むずかしすぎると不評でした。しかし、著者の思い出もありますので、新版でも残しました。

2004/7/30

藤田 眞作

旧版のはしづみ

L^AT_EX は、T_EX の制御綴 (control sequence) を組み合わせたマクロ集です。L^AT_EX の範囲内で、文書を書くのはきわめて簡単です。(j)article, (j)book, (j)report などのスタイルファイルを使えば、大抵は、満足できる仕上がりになります。しかし、L^AT_EX を長く使う人で、この段階で留まる人は少ないように思います。これは推測ですが、この段階で留まる人は、最初から「見たまま方式」のワードプロセッサに走るのではないのでしょうか。

L^AT_EX の命令に慣れてくると、「レイアウトを変えて見たい」、「こんな命令はないだろうか」などの希望がでできます。しかし、ちょっとしたレイアウトが気に入らないようなときでさえ、すぐに対処できるかという、必ずしも簡単にゆくとは限りません。こんなとき、多くのユーザーは、広く出まわっているスタイルファイルを漁り始めます。「こんな機能がないかなあ」と思うようなときには、NIFTY-Serve などの会議室で質問を出せば、すでに便利なスタイルファイルがあると教えてもらうことができます。

NIFTY-Serve の FSNOTE, FPRINT, FLABO, FEXT などのライブラリーには、このようなスタイルファイルがたくさん蓄積されています。論文などを書く道具としての使い方に徹する場合は、既存のスタイルファイルを上手に利用するという他力本願の姿勢は、まったく正しい姿です。そんなに凝ったことを望まなければ、この段階で大抵のことは済んでしまうでしょう。

ところが、完成品のマクロを使うだけでは、そのうちに満足できなくなります。また、まとまった学位論文や書籍などを書く場合や、投稿規定に特殊な様式が定められたものに投稿しなければならない場合などにも遭遇します。既存のスタイルファイルやマクロがない限り、自分で作成しなければならない事態になります。NIFTY-Serve などの会議室で質問を出せば、親切な方が、目的に合ったマクロを作ってくれることもあります。

そうこうするうちに、マクロの完成品を使うだけでは気がすまず、いろいろな便利なマクロがどうしてこのような機能をもっているのだろうかという疑問がわいてきます。そして、簡単なマクロを作るのから始めて、マクロ作成に凝りだし、既存のスタイルファイルを自分用にカスタマイズして楽しむようになります。

この他力本願から自力本願への変わり目のユーザーには、適当な参考書が少ないように思います。T_EXbook を読みこなすのは、大変むずかしいし、L^AT_EX のソースリストである latex.tex のオンライン解説 (英語) を読むのも面倒なものです。まずもって、どこから取り付いたらよいのやわからないというのが実情ではないのでしょうか。

マクロの作り方という面では、奥村本や磯崎本が有名です。かくいう私も、「化学者・生化学者のための L^AT_EX」(東京化学同人, 1993) というのを出版していますが、マクロの作り方について、かなりのスペースをとって説明しました。あらゆるレベルの読者を満足させる本というのではないでしょうから、できるだけ多様な参考書があれば、それだけ読者の選択肢が増えると思われれます。そんなわけで、この他力本願から自力本願への変わり目のユーザーに贈るのが、この「L^AT_EX まくろの八衢」(らてふ・まくろのやちまた) です。

ここにとりあげた章のいくつかは、NIFTY-Serve の FPRINT No. 12 の会議室にアップした同名の連載に少し手を入れたものです。そのほかに、NIFTY-Serve の T_EX 関係の会議室に出てきた質疑からヒントを得て書き下ろしたものが数章あります。この部分については、著作権の関係から、私が書き入れたコメントを中心に説明してありますが、これが必ずしも最良の解決策とは限りません。ぜひ、批判的精神で、

もっとよい解決方法があるという目で読んでくださいますよう。昨今の会議室の質疑では、単なるマクロの作成の範囲で済むというものばかりでなく、中間ファイルの取扱いに慣れたほうがよいとおもわれる事例がかなりあります。この理由から中間ファイルの利用のテクニックについては、とくに詳しく述べてあります。これに関しては、ほかにより参考書がないようにおもわれますので、本書の特徴として強調できるでしょう。のこりの幾許かは、上記拙著を書いたときにページ数の関係で削除したものを補筆して載せました。これは、拙著の該当のところを読まれてからのほうが理解しやすいかも知れません。

このタイトルは、本居春庭の有名な著書「詞の八衢」(ことばのやちまた)のもじりです。「はしぶみ」(序)というのもこの著書からとりました。八衢とは、ちまたの数のおおいところ、大勢の人のあつまる巷ということです。パソコン通信の会議室には実にふさわしいことばなので、借用させていただきました。

1995/5/27

藤田 眞作

目次

第 1 章	制御綴の引数	11
1.1	置き換えマクロ	11
1.2	引数一つのマクロの種々相	12
1.2.1	文字列を引数に	12
1.2.2	制御綴を引数に	12
1.2.3	引数の範囲の明示	13
1.2.4	特殊な引数範囲の指示	13
1.3	複数個の引数	14
1.3.1	通常の使い方	14
1.3.2	引数の範囲の明示	15
1.4	置き換えマクロの追加説明	15
第 2 章	文字列リスト処理の基礎	17
2.1	部分文字列の抽出	17
第 3 章	TeX 流カウンター指南	21
3.1	カウンターと足し算	21
3.2	カウンターの大小による分岐処理	23
第 4 章	L^AT_EX 2_ε流カウンター指南	25
4.1	カウンターと足し算	25
4.2	TeX 流カウンターとの違い	27
4.3	カウンターの大小による分岐処理	28
第 5 章	練習 その 1	31
5.1	部分文字列の取得	31
5.2	カウンター	33
第 6 章	カウンターの親子関係と一括リセット	35
6.1	カウンター値の参照と一括リセット命令	35
6.2	一括リセットの仕組み	37
6.2.1	可変命令を含むリスト	37
6.2.2	リストの作成命令	38
6.3	L ^A T _E X 2 _ε カウンターリスト	38
6.4	余計なこと	40

第 7 章	カウンターの親子関係の削除	41
7.1	ソースリスト	41
7.2	機能と使い方	42
7.3	マクロ解説	44
7.3.1	全体の方針	44
7.3.2	リストの作成	45
7.3.3	リセット命令の再定義	45
第 8 章	相互参照とリスト処理の応用	47
8.1	例題マクロ	47
8.2	部分文字列の取得	49
8.3	参照キーと参照番号の受け渡し	50
8.4	その他	51
第 9 章	補助ファイルの活用 (1)	53
9.1	補助ファイルへの書き込み	53
9.2	補助ファイルに書き込まれた命令	54
第 10 章	補助ファイルの活用 (2)	57
10.1	ページ番号の工夫	57
10.1.1	第一案	57
10.1.2	第二案	58
10.1.3	そのほかの方法	60
第 11 章	スタイルファイルの仕立て方	61
11.1	スタイルファイルの登録と使用	61
11.1.1	登録手順	61
11.1.2	自作のスタイルファイルの使用	62
11.2	複雑な例	63
第 12 章	ページごとの脚注番号	65
12.1	親子関係の更改—不十分	65
12.2	aux ファイルの活用	66
第 13 章	中間ファイルの新設	69
13.1	ソースリスト	69
13.2	機能と使い方	70
13.3	マクロ解説	73
13.3.1	全体の方針	73
13.3.2	中間ファイルのオープンとクローズ	73
13.3.3	中間ファイルへの出力	74
13.3.4	中間ファイルの読み込み	75
13.3.5	環境新設	75
13.4	その他の方法	75

第 14 章 目次用の中間ファイル	77
14.1 toc ファイルへの書き出し命令の作成	77
14.2 目次の出力の仕組み	78
14.2.1 aux ファイルへの書き出し	78
14.2.2 aux ファイルへから toc ファイルへ	79
14.2.3 toc ファイルの内容を出力	79
第 15 章 目次項目の追加	81
15.1 引数に関する注意事項	81
15.2 目次への項目追加	83
15.3 別立ての目次	83
第 16 章 目次の体裁の変更	85
16.1 指定できるパラメーター	85
16.2 リーダー	86
16.3 和書用の目次リーダー	87
第 17 章 章立て命令の種々相	89
17.1 浅いレベルでできること	89
17.2 内部マクロ細見	91
第 18 章 罫線を利用した数学・化学記号	95
18.1 標準をあらわす添え字	95
18.2 不等号	96
第 19 章 $\X\TeX$ のロゴ	99
19.1 \TeX のロゴの定義	99
19.2 $\X\TeX$ のロゴの定義	99
19.2.1 ロゴの由来	99
19.2.2 第一法	100
19.2.3 第二法	101
第 20 章 \LaTeX のロゴ細見	103
20.1 オリジナルの定義	103
20.2 改良版	104
20.3 さらに改良	105
20.4 $\LaTeX 2_{\epsilon}$ における定義	106
第 21 章 文字列の長さ	107
21.1 文字列の長さの測定	107
21.1.1 $\LaTeX 2_{\epsilon}$ 流測量術	107
21.1.2 \TeX 流測量術	108
21.2 文字の間隔と均等割	108
21.3 自動均等割	109

21.4 内部マクロの作成と利用	110
第 22 章 見出しの自動均等割	113
22.1 箇条書環境への応用	113
22.2 Jdescription 環境の使い方	115
22.3 Jdescribe 環境の使い方	116
22.4 マクロの説明	117
22.4.1 ラベル出力のマクロの解説	117
22.4.2 Jdescription 環境のマクロの解説	118
22.4.3 Jdescribe 環境のマクロの解説	118
22.5 節見出しの自動均等割	118
第 23 章 文字列の折り返し	121
23.1 文字列の抽出	121
23.2 文字列の折り返し	122
23.3 文字数の半分	122
23.4 読みのマクロ	123
第 24 章 練習その 2	125
24.1 文字列の長さの測定	125
24.2 文字数の勘定	126

第1章 制御綴の引数

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

TeX (L^AT_EX) では、`\def` という制御綴 (control sequence)²を定義する命令 (これも制御綴) があります。この命令で作成した制御綴は引数を取ることができますが、引数がどのように処理されるかを調べてみましょう。なお、L^AT_EX には、同様の目的で `\newcommand` と `\renewcommand` が用意されていますが、ここでは触れずに、TeX の `\def` を中心に話を進めます。

1.1 置き換えマクロ

`\def` のもっとも簡単な使い方は、置き換えマクロの作成です。control sequence という言葉の訳語として、「制御綴」を採用した場合、

```
\def\conseq{制御綴}
```

と定義しておけば、それ以後「`\conseq{}`」と入力すれば、「制御綴」と出力されるようになります。この場合、作成したマクロは引数を取りません。末尾の `{}` は、命令と地の文を区別するためにいれました。

定義の中には、命令 (制御綴) を含ませることができます。たとえば、下線を引く命令 `\underline` を使って、

```
\def\Conseq{\underline{制御綴}}
```

と定義しておけば、それ以後「`\Conseq{}`」と入力すれば、「制御綴」と出力されるようになります。次のように一度定義した `\conseq` を定義の中で使うこともできます。たとえば、

```
\def\ConSeq{$\overline{\mbox{\conseq}}$}
```

と定義しておけば、それ以後「`\ConSeq{}`」と入力すれば、「制御綴」と出力されるようになります。

この場合は、多少便利なことがあります。それは、`\conseq` の定義を変更することができることです。たとえば、

```
\ConSeq{};
{\def\conseq{コントロール・シーケンス}\ConSeq{};}
\ConSeq{}
```

¹◎藤田 眞作「L^AT_EX 2_ε まくろの八衢」。(00)「まくろのやちまた」と読みます。

²「せいぎょつづり」と読みます。文脈によって、同じものを「命令」といったり、「コマンド」といったりしますが、必ずしも明確な方針で使い分けしているわけではありません。

のように入力すると、次のような出力がえられます。

制御綴; コントロール・シーケンス; 制御綴

実際にこれらをテストするには、次のように L^AT_EX 2_ε の文書の定型部分を加えて、書式を整えておく必要があります。

```
\documentclass{jarticle}
\def\conseq{制御綴}
\def\Conseq{\underline{制御綴}}
\def\ConSeq{\overline{\mbox{\conseq}}}$}
\begin{document}
\conseq{,}
\Conseq{,}
\ConSeq{,}
\end{document}
```

1.2 引数一つのマクロの種々相

1.2.1 文字列を引数に

引数を取る命令 `\hikisu` を作ってみましょう。

```
\def\hikisu#1{\mbox{\langle $#1\rangle$}}
```

これを実際に使うには、引数を中括弧に入れて `\hikisu{寸法}` のように書きます。これによって、 \langle 寸法 \rangle と出力されます。区切りの \langle の後の空白が少し広すぎますが、説明だけのために作った命令ですから我慢してください。

引数を中括弧に入れずに処理したらどうなるか。実際に試してみましょう。半角の空白を明示するために、記号 `␣` を使って表記しておきます。

```
\hikisu␣寸法␣\hikisu␣ABCD
```

と入力してみます。 `\hikisu` の後の半角の空白はの場合必要ですので、ぜひ入れてください。そうしないと、L^AT_EX 処理のときに、エラーになってしまいます。この出力は次のようになります。

\langle 寸 \rangle 法 \langle A \rangle BCD

この場合は、最初の一文字のみが引数として認識されていることがわかります。このように引数を中括弧で囲まない場合と、上で示した囲んだ場合とで出力が異なりますので注意が必要です。

1.2.2 制御綴を引数に

今度は、実際の文字列でなく、置き換えマクロとして作成した制御綴 `\conseq` を、引数として指定した場合を試してみます。

```
\hikisu␣\conseq␣\hikisu␣制御綴
```

と入力することによって、次のような出力がえられます。

〈制御綴〉〈制〉御綴

このように、制御綴`\conseq`は、一まとまりの引数として認識されることがわかります。いいかえれば、制御綴`\conseq`は「展開」されていないのです。では、`\conseq`を文字列「制御綴」に展開したのちに、引数として処理したいときにはどうしたらよいのでしょうか。今度は、同じ`\hikisu`命令の前に`\expandafter`を用いて、

```
\expandafter\hikisu_\conseq_\hikisu_制御綴
```

と入力してみましょう。次のような出力がえられます。

〈制〉御綴〈制〉御綴

`\expandafter`命令は読んで字のごとく、それに続く制御綴の展開を後回しにする機能をもっています。つまり、この場合は、`\conseq`を展開した後で、`\hikisu`命令を展開して処理を行うのです。このテクニックは、応用範囲が広いのでおぼえておくとよいでしょう。

1.2.3 引数の範囲の明示

引数の範囲を限定するために、区切り記号を用いることもできます。たとえば、

```
\def\hikisua#1/{\mbox{${\langle$#1\rangle$}}
```

と定義したとしましょう。すると、

```
\hikisua 寸法/ \hikisua ABCD/
```

と入力することによって、次のような出力がえられます。

〈寸法〉〈ABCD〉

このように引数の範囲を区切り記号で明示することは、いろいろな局面で応用が効きます。区切り記号は、コンマ、ピリオド、セミコロン、コロンなど広く使えますので、実際にためしてみることをおすすめします。

1.2.4 特殊な引数範囲の指示

段落単位で引数とすることもできます。この場合は、`\par`を区切り記号として、制御綴を定義します。

```
\def\hikisuc#1\par{${\langle$#1\rangle$}
```

このようにすると、

```
\hikisuc_引数の範囲を段落単位にとることもできます。
```

これがその例です。

このように、`\hikisuc_`命令の引数の終わりは改行でもよいし、`{\tt_\backslash$par}`命令で明示してもかまいません。`\par`

と入力することによって、次のような出力がえられます。

〈引数の範囲を段落単位にとることもできます。これがその例です。〉このように、〈命令の引数の終わりは改行でもよいし、`\par` 命令で明示してもかまいません。〉

今度はすこし特殊なことをしてみましょう。半角の区切りを入れて、これを区切りとして、引数処理を行うことを試みましょう。次のように区切りに`^^I`を使います。

```
\def\hikisub#1^^I{\mbox{$\langle\!#1\rangle$}}
```

このような命令`\hikisub`を使ってみましょう。行の終わりに半角の空白を入れてありませんが、入れなくても、この場合は入れたのと同じ結果になります。

```
\hikisub 寸法 \hikisub ABCD
```

と入力することによって、次のような出力がえられます。

〈寸法〉(ABCD)

空白を`^^I`で処理する方法は、複数個の引数の場合には使えないようです。まあ、こんなこともできるという例程度に考えてください。

1.3 複数個の引数

1.3.1 通常の使い方

語の順番を変えるマクロを作ってみましょう。

```
\def\trikae#1#2{#2#1}
```

のように定義すると、「`\trikae{ABC}{DE}`」と入力することによって、「DEABC」が得られます。今度は、二番目の中括弧の引数を忘れたとしましょう。「`\trikae{ABCDE}`」と入力することによって、「」ABCDE が得られます。理由は、これまでに説明したことからおわかりでしょう。つまり、終わりカギ括弧が第 2 引数として認識されています。

内部に既に作成した`\trikae`を利用して、二番目の引数を忘れてしまったときにもうまくゆく命令を作成します。次のように`\Trikae` 命令を定義します。

```
\def\Trikae#1{\trikae#1\relax}
```

仕掛けは、`\relax` を最後尾におくことです。この命令は何もしないという機能をもっていますが、この場合は、二番目の引数に相当するものがないときに、`\relax` がその代わりをします。したがって、

```
\Trikae{A} and \Trikae{ABCDEF} and \Trikae{{ABC}{DEF}}
```

と入力することによって、次のような出力がえられます。

A and BACDEF and DEFABC

それぞれの出力を注意深く比較してください。理由は、これまでに説明したことからおわかりのことと思います。とくに、真ん中の例については、A が第 1 引数、B が第 2 引数として認識されていることに注意

してください。また、`\Trikae` の定義の中で `\relax` を削除してみて、同じ入力でどんなものがえられるかを調べるのもおもしろいでしょう。

さらに、`\def\AtoF{{ABC}{DEF}}` で定義された制御綴を、`\Trikae` の引数として与えた場合を考えてみます。この場合は、「`\Trikae{\AtoF}`」とすると、「`ABCDEF`」のようになります。せっかく中括弧で二重に区切ってもその効果がでないことがわかります。この理由は、制御綴 `\AtoF` が展開されていないためだというのは、第 1.2.2 項の説明から、すぐにわかります。そこで、このような場合にも対処できるように、定義を変更します。`\expandafter` 命令を内部で使います。

```
\def\Trikae#1{\expandafter\trikae#1\relax}
```

その上で

```
\Trikae{A}, \Trikae{ABCDEF}, \Trikae{{ABC}{DEF}} and \Trikae{\AtoF}
```

と入力することによって、次のような出力がえられます。今度は、中括弧で二重に区切った効果がでてきます。

A, BACDEF, DEFABC and DEFABC

1.3.2 引数の範囲の明示

引数の範囲を限定するために、区切り記号を用いることもできます。次のような定義で、マクロ `\Nengappi` を作成します。

```
\def\Nengappi#1/#2/#3/{#1年#2月#3日}
```

その上で、`\Nengappi1995/5/17/` と入力すると、1995年5月17日がえられます。`TEX` 内部では、現在の年・月・日が `\year`、`\month`、`\day` に保存されていますので、

```
\Nengappi\the\year/\the\month/\the\day/.
```

と入力することによって、次のように本日の日付が出力されます。すなわち、2004年7月20日。

ちなみに、`pLATEX 2ε` では、`\today` という命令が用意されているので、「`\today`」と入力すると、「平成16年7月20日」と出力されます。元号表示を西暦表示にするには、「`\西暦\today`」と入力します。たしかに「2004年7月20日」となります。`\西暦` は、元号表示を西暦表示に変更するスイッチになっています。したがって、`\西暦` の宣言以降は、すべて西暦表示になります。もとに戻す必要がある場合は、「`\和暦\today`」と入力すると、「平成16年7月20日」と変更することができます。`\和暦` は、西暦表示を元号表示に変更するスイッチです。

1.4 置き換えマクロの追加説明

`\documentclass` の引数として `jbook` を指定すると、文献リストの標題は「関連図書」と出力されます。これは、`jbook.cls` の中で、`\newcommand{\bibname}{関連図書}` と定義してあるためです。本書は、`jbook` クラスを使っていますので、ここで「`\bibname`」と入力すると、たしかに「関連図書」と出力されます。これを「参考文献」と変更したい場合は、文書の先頭などに

```
\def\bibname{参考文献}
```

と書けば目的が達せられます。ここで、「\bibname」と入力するとたしかに「参考文献」と出力されます。

この種の置き換えマクロは、文書クラスごとに用意されています。変更するには、ここで説明した\defを使うか、*L^AT_EX 2_ε*に従って\renewcommandを使います。

```
\renewcommand{\postpartname}{部}
\renewcommand{\contentsname}{目次}
\renewcommand{\listfigurename}{図目次}
\renewcommand{\listtablename}{表目次}
\renewcommand{\refname}{参考文献}
\renewcommand{\indexname}{索引}
\renewcommand{\figurename}{図}
\renewcommand{\tablename}{表}
\renewcommand{\appendixname}{付録}
\renewcommand{\abstractname}{概要}
```

第2章 文字列リスト処理の基礎

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

今回は、文字列やそのリストを分解して、部分文字列を取り出すにはどうすればよいかをかんがえてみましょう。とくに、 \LaTeX 2_ϵ の \@for と \@tfor の使い方を紹介します。

2.1 部分文字列の抽出

\LaTeX 2_ϵ は、 \TeX の制御綴 (control sequence) を組み合わせたマクロ集です。ユーザーが使うことのできる制御綴については、 \LaTeXbook などの使用説明書などに詳しく説明されています。通常の文書を書くには、これでほとんどの場合用が足りてしまいます。しかし、 \LaTeX 2_ϵ の内部には、マクロ集を作り上げるための多くの制御綴が隠されています。これらの内部制御綴 (隠しコマンド) は、自分でマクロを作り始めたユーザーにとって、きわめて便利なものです。今回とりあげる \@for と \@tfor は、そうした隠しコマンドの代表的なものです。

\LaTeX 2_ϵ の文書ファイルの冒頭の決まり文句として、

```
\documentclass[12pt,draft]{jbook}
```

などと書くことは \LaTeX 2_ϵ ユーザーにはおなじみでしょう。角括弧のなかにはオプションをコンマで区切って記入してゆきます。コンマで区切れば、指定するオプションの数は必要なだけ増やしてゆくことができます。 \LaTeX 2_ϵ では、このようなコンマで区切った文字列リストを処理するために、内部制御綴 (隠しコマンド) \@for が用意されています。また、文字列の一字ずつを分離して処理するための \@tfor という制御綴もあります。実際にためてみましょう。

```
\documentclass{article}
\begin{document}
\makeatletter
\@for\member:=orange,apple,banana\do{\member-} \par
\@tfor\member:=orange,apple,banana\do{\member-}
\makeatother
\end{document}
```

のように文書ファイルを作って処理してください。:=と \do とには含まれた orange,apple,banana が処理すべき文字列です。同じ文字列を \@for と \@tfor を用いて処理しています。この例文からは、

¹◎藤田 眞作「 \LaTeX 2_ϵ まぐろの八衢」.(01)「まぐろのやちまた」と読みます。

```
orange-apple-banana-
o-r-a-n-g-e,-a-p-p-l-e,-b-a-n-a-n-a-
```

のような出力が得られます。 \@for を使った例では、コンマで区切られた部分文字列を取得して、その文字列を \member という制御綴に代入します。もちろん、この制御綴の名前は任意です。ついで、取得した文字列 (\member に保存) を用いて、 \do に続く中括弧の中に書いた処理を行います。この例では、簡単のため「文字列を出力して末尾にハイフンを付ける」(ハイフン分割) という処理を行います。 \do{\member-} のように書けばよいことはわかりでしょう。

一方、 \@tfor を使った例では、一文字ずつを取り出して処理してゆきます。ついで、取得した文字列 (\member に保存) を用いて、 \do に続く中括弧の中に書いた処理を行うことは同じです。ここでは、同様に「文字列を出力して末尾にハイフンを付ける」(ハイフン分割) という処理を行っていますが、一文字ずつハイフンで区切られることになります。また、コンマも一文字として取り扱われています。

\@for や \@tfor を有効に使うためには、文字列を直接に代入するのでは限界があります。そこで、いったん文字列を制御綴 \Fruit に収めることを考えましょう。次のように、 \def を使います。

```
\documentclass{article}
\begin{document}
\makeatletter
\def\Fruit{orange,apple,banana}
\@for\member:=\Fruit\do{\member-} \par
\@tfor\member:=\Fruit\do{\member-}
\makeatother
\end{document}
```

出力を次に示します。 \@for の処理結果は、前と同じですが、 \@tfor の処理結果は、前とは異なってしまいました。なぜなのでしょう。

```
orange-apple-banana-
orange,apple,banana-
```

出力結果をよくみると、 \@tfor の処理では、コンマがそのままのこっており、ハイフンは最後尾のみに出力されています。つまり、 \@tfor では、 \Fruit がひとまとまりの文字として処理されています。 T_EX のことばでいいかえると、 \Fruit の文字列が、展開されないまま処理されています。このような場合に、 \@tfor などの制御綴の実行を後回しにして、後続の制御綴をさきに展開する(この場合は、 \Fruit の文字列をさきに展開する) 必要があります。このような処理に対処するため、 T_EX の制御綴として、 \expandafter が用意されていたことを思い出してください。さて、覚悟をきめて、上の \@tfor の例文に \expandafter を入れまくりましょう。

```
\makeatletter
\def\Fruit{orange,apple,banana}
\expandafter\@tfor\expandafter\member\expandafter:%
\expandafter=\Fruit\do{\member-}
\makeatother
```

面倒ですから、これ以降は、 \documentclass などの命令を省略して、必要な部分のみを書きます。出力は次の通りです。見事に所期の結果がえられました。

```
o-r-a-n-g-e,-a-p-p-l-e,-b-a-n-a-n-a-
```

さらに、応用するためには、制御綴の中で使うことが必要になります。とくに処理の対象が引数で与えられるようにすると、ぐんと適用範囲が広がります。簡単な例として、`\@for` と `\@tfor` の機能を用いて、引数をもつ制御綴 `\doFruit` を作ってみましょう。

```
\makeatletter
\def\doFruit#1{%
\@for\member:=#1\do{\member-} \par
\@tfor\member:=#1\do{\member-}}
\makeatother
(例)\doFruit{orange,apple,banana}
```

とします。引数 #1 で与えた文字列が、ハイフン分割の処理を行うべき文字列になります。この場合は、展開してからハイフン分割処理が行われます。

```
(例)orange-apple-banana-
o-r-a-n-g-e,-a-p-p-l-e,-b-a-n-a-n-a-
```

応用として、もう少しもっともらしいマクロを作ってみましょう。6/3/20 などの文字列をコンマで区切って作った文字列から、それぞれを取り出して、平成 6 年 3 月 20 日などとするマクロを作成します。このマクロの名前を `\nenngappi` とします。

```
\makeatletter
\def\@nenngappi#1/#2/#3/{平成#1年#2月#3日}
\def\nenngappi#1{%
\@for\member:=#1\do{\expandafter\@nenngappi\member/}}
\makeatother
(例)\nenngappi{6/3/20,7/5/27,7/6/1}
```

と定義しますと、次の出力が得られます。

```
(例)平成 6 年 3 月 20 日 平成 7 年 5 月 27 日 平成 7 年 6 月 1 日
```

念のために、この命令の成り立ちを説明しておきましょう。`\nenngappi` 命令の中での `\@for` の使い方は、これまでと同じです。`\do` に続く中括弧の中での処理に、`\@nenngappi` 命令を使っています。この命令は、スラッシュ(/) で区切った文字列をそれぞれに分離して、平成・年・月・日を付け加えて出力するものです。これは、`\member` に保存された文字列に作用させますが、あらかじめ `\member` を展開しておく必要があります。`\@nenngappi` 命令の直前の `\expandafter` はそのためのものです。

第3章 T_EX 流カウンター指南

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

今回は、T_EX のカウンターの説明を行います。L^AT_EX 2_εには別のカウンター体系がありますが、ここでは、混乱を避けるため T_EX の体系のみを説明します。

3.1 カウンターと足し算

T_EX では、`\newcount` という制御綴によって、新しいカウンターを定義します。例えば

```
\newcount\TestCount
```

と書きますと、それ以後 `\TestCount` というカウンターが使えるようになります。カウンター値は直接数字で代入することができます。

```
\newcount\TestCount
```

```
\TestCount=100
```

カウンター`\verb/\TestCount/`の値は `\the\TestCount` です。

とかけますと、出力は次のようになります。ここで `\the` はカウンターの値を出力するための T_EX の制御綴です。

カウンター`\TestCount` の値は 100 です。

これ以降は、カウンター`\TestCount` が定義済みとします。カウンターの足し算は、`\advance` を使います。by の後に加えるべき整数を書きます。by は省略可能です。

```
\TestCount=100
```

カウンター`\verb/\TestCount/`の値は `\the\TestCount` です。`\par`

```
\advance\TestCount by 25
```

カウンター`\verb/\TestCount/`の値は `\the\TestCount` です。`\par`

```
\advance\TestCount -15
```

カウンター`\verb/\TestCount/`の値は `\the\TestCount` です。

としますと、次のような出力が得られます。

カウンター`\TestCount` の値は 100 です。

¹©藤田 眞作「L^AT_EX 2_εまくろの八衢」(02)

カウンター\TestCount の値は 125 です。
 カウンター\TestCount の値は 110 です。

加える数は、次の例のようにカウンターで与えてもかまいません。この例では、\TestCount の古い値を加えて新しいカウンター値としています。

```
\TestCount=100
カウンター\verb/\TestCount/の値は \the\TestCount です.\par
\advance\TestCount by \TestCount
カウンター\verb/\TestCount/の値は \the\TestCount です.\par
\advance\TestCount \TestCount
カウンター\verb/\TestCount/の値は \the\TestCount です。
```

出力は次の通りです。

カウンター\TestCount の値は 100 です。
 カウンター\TestCount の値は 200 です。
 カウンター\TestCount の値は 400 です。

前の章で解説した\@for 命令と、カウンターの足し算を利用して、文字列リストの要素の数を数えるマクロ\itemNo を作成しましょう。定義を次に示します。

```
\makeatletter
\def\itemNo#1{\TestCount=0
  \@for\member:=#1\do{\advance\TestCount by 1}\the\TestCount}
\makeatother
```

\do に続く中括弧の中で\member が一回検出される度に\TestCount に 1 ずつ加えています。最後に\the 命令で、\TestCount の値を出力しています。実際に、\itemNo 命令を使ってみましょう。次のように書きます。

(例) リスト中の要素の個数を求める:

```
\itemNo{orange}; \itemNo{orange,apple};
\itemNo{orange,apple,banana}; \itemNo{1,2,3,4,5,6}
```

出力は次のようになります。やや、本格的なマクロらしくなってきました。

(例) リスト中の要素の個数を求める: 1; 2; 3; 6

方法としては、同じですが与えられた文字列に含まれる文字の個数を勘定してみましょう。今度は、前の章で解説した\@tfor 命令を使います。名前を\characterNo とします。定義を次のとおりです。

```
\makeatletter
\def\characterNo#1{\TestCount=0
  \@tfor\member:=#1\do{\advance\TestCount by 1}\the\TestCount}
\makeatother
```

何のことはない。要するに、\itemNo の定義の中で、\@for を \@tfor に変えただけです。実際に使ってみましょう。

(例) 文字列中の文字の個数を求める:

```
\characterNo{orange}; \characterNo{apple}; \characterNo{banana}
```

出力はつぎの通りです.

(例) 文字列中の文字の個数を求める: 6; 5; 6

3.2 カウンターの大小による分岐処理

カウンターの値を評価して、その大小によって処理を分岐したいことがあります。*T_EX*には、`\ifnum... \fi`という条件文が使えるようになっています。文字列の文字数が5文字以上になるとそれ以上は出力しないような機能をもつマクロを作ってみましょう。命令の名前を`\shortenchar`としましょう。上で定義した`\characterNo`命令を少し変えます。

```
\makeatletter
\def\shortenchar#1{\TestCount=0
  \@tfor\member:=#1\do{\advance\TestCount by 1
    \ifnum\TestCount<6 \member\fi}}
\makeatother
```

`\ifnum`と`\fi`で条件文を構成しています。この場合は、5文字までは`\member`として取得した文字をそのまま出力しますが、6文字以上になると、何もしないということになります。実際に使ってみましょう。

(例) 文字列中を5文字以下に約める:

```
\shortenchar{orange}; \shortenchar{apple};
\shortenchar{banana}; \shortenchar{plum}
```

とかきますと、次のような出力が得られます。

(例) 文字列中を5文字以下に約める: orang; apple; banan; plum

こんなことをして何の役にたつのだとお思いになる諸兄もいらっしゃるでしょう。ここで、これまでの知識を総合して、実用に耐えるマクロを作ってみます。`\andfor`というマクロは、`Fig. \andfor{A,B,C}`のように入力すると、「Fig. A, B and C」のように整形する機能をもつものです。定義と使い方を示しておきましょう。

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Analog to \@for by Shinsaku Fujita 1995/05/26
%
% \andfor{item1,item2,...}
%
\makeatletter
\def\andfor#1{\@tempcnta=\m@ne \@tempcntb=\z@%
  \@for\member:=#1\do{\advance\@tempcnta\@ne}% 文字列リストの要素の合計-1
  \@for\member:=#1\do{\advance\@tempcntb\@ne}%文字列リストの要素の個数
  \ifnum\@tempcntb<\@tempcnta %合計-1より小さいとき
    \ifnum\@tempcnta<1\relax \member %合計個数1ならそのまま出力
```

```

\else
  \ifnum\@tempcnta<2\relax \member\space %合計個数 2 ならば, 出力と空白
\else \member, %そのほかは, コンマ付きで出力
\fi\fi
\else %合計-1 以上, すなわち最後の二つの要素の処理
  \ifnum\@tempcntb=\@tempcnta \member\space and %最後から 2 つ目
  \else \member %最後のもの
\fi\fi}}
\makeatother

```

この命令を使うには, 次のように書きます.

```

(例) \andfor{orange}; \andfor{orange,apple};
\andfor{orange,apple,banana}; Fig. \andfor{1,2,3,4,5,6}

```

出力は次のようになります.

```

(例) orange; orange and apple; orange, apple and banana; Fig. 1, 2, 3, 4, 5 and 6

```

`\andfor` というマクロの内部でどんなことを行っているかを説明しておきましょう. まず, カウンターとして, `\@tempcnta` と `\@tempcntb` を使っています. これは, *L^AT_EX 2_ε* の中では, 一時的に使うカウンターとして宣言済みのものなので, `\newcount` で宣言せずとも使用できます. `\m@ne` は, `-1` と同義, `\z@` は `0` と同義です. `\@for` を 2 回使っていますが, 最初は, 文字列の要素の個数を調べて, 合計 `-1` としたものをカウンター `\@tempcnta` に格納しています. この方法は, `\itemNo` の手法と同じです. 二番目の `\@for` では, 文字列の要素の数をカウンター `\@tempcntb` に格納し, これと合計 `\@tempcnta` の値と比較しています. この例のように `\ifnum` は入れ子で多重に使用することができます. また `\else` 命令も注目してください.

(参考) `\andfor` の定義としては, 岩熊・古川「*L^AT_EX* のマクロとスタイルファイルの利用」の第 15.5 項に別のアプローチによるものが記載されています. 本項のものと比較すると, マクロの作成についてさらに理解が深まるでしょう.

第4章 L^AT_EX 2_ε流カウンター指南

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

今回は、L^AT_EX 2_εのカウンター体系です。前の章の T_EX カウンターを、そのまま L^AT_EX 2_εカウンターに置き換えてみましょう。比較することによって、理解が深まると思います。

4.1 カウンターと足し算

T_EX では、`\newcounter` という制御綴によって、新しいカウンターを定義します。例えば

```
\newcounter{TestCount}
```

と書きますと、それ以後 `TestCount` というカウンターが使えるようになります。ここで `TestCount` の先頭に `\`印がついていないことに注意してください。このことは、後程説明することになります。

カウンターに値を設定するには `\setcounter` を使います。カウンター値を取り出すには、`\value` を使います。

```
\setcounter{TestCount}{100}
```

カウンター `\verb/TestCount/` の値は `\the\value{TestCount}` です。

とかきますと、出力は次のようになります。ここで `\the` はカウンターの値を出力するための T_EX の制御綴で、前章にも出てきました。

カウンター `TestCount` の値は 100 です。

L^AT_EX 2_εでは通常カウンター値を出力するための体系的な命名を行っています。これはカウンター名の直前に `\the` を冠するという方法です。L^AT_EX 2_εに慣れた方ならば、`\thechapter` など多くの制御綴があることにすでにお気付きになっておられるでしょう。L^AT_EX 2_εでは、`\newcounter` 命令でカウンターを新設すると、カウンター値を出力する制御綴 (`\the` カウンター) が自動的に作成されるようになっています。これが `\newcount` と `\newcounter` の違いの一つです。したがって、今回の例では、`\theTestCount` とすれば、カウンター値が 100 のように出力されます。

これ以降は、カウンター `TestCount` が定義済みとします。カウンターの足し算は、`\addtocounter` を使います。この命令は、二つの引数を取りますが、後に加えるべき整数を書きます。

¹©藤田 眞作「L^AT_EX 2_εまくろの八衢」(03)

```
\setcounter{TestCount}{100}
カウンター\verb/TestCount/の値は \theTestCount です.\par
\addtocounter{TestCount}{25}
カウンター\verb/TestCount/の値は \theTestCount です.\par
\addtocounter{TestCount}{-15}
カウンター\verb/TestCount/の値は \theTestCount です.\par
```

としますと、次のような出力が得られます。

カウンター TestCount の値は 100 です。
 カウンター TestCount の値は 125 です。
 カウンター TestCount の値は 110 です。

加える数は、次の例のようにカウンターで与えてもかまいません。この例では、\TestCount の古い値を加えて新しいカウンター値としています。

```
\setcounter{TestCount}{100}
カウンター\verb/TestCount/の値は \theTestCount です.\par
\addtocounter{TestCount}{\the\value{TestCount}}
カウンター\verb/TestCount/の値は \theTestCount です.\par
\addtocounter{TestCount}{\the\value{TestCount}}
カウンター\verb/TestCount/の値は \theTestCount です.\par
```

出力は次の通りです。

カウンター TestCount の値は 100 です。
 カウンター TestCount の値は 200 です。
 カウンター TestCount の値は 400 です。

前の章で解説した \@for 命令と、カウンターの足し算を利用して、文字列リストの要素の数を数えるマクロ \itemNo を作成しましょう。定義を次に示します。

```
\makeatletter
\def\itemNo#1{\setcounter{TestCount}{0}
  \@for\member:=#1\do{\addtocounter{TestCount}{1}}\theTestCount}
\makeatother
```

\do に続く中括弧の中で \member が一回検出される度に TestCount に 1 ずつ加えています。最後に \theTestCount 命令で、TestCount の値を出力しています。実際に、\itemNo 命令を使ってみましょう。次のように書きます。

(例) リスト中の要素の個数を求める:

```
\itemNo{orange}; \itemNo{orange,apple};
\itemNo{orange,apple,banana}; \itemNo{1,2,3,4,5,6}
```

出力は次のようになります。やや、本格的なマクロらしくなってきました。

(例) リスト中の要素の個数を求める: 1; 2; 3; 6

方法としては、同じですが与えられた文字列に含まれる文字の個数を勘定してみましよう。今度は、前の章で解説した \@tfor 命令を使います。名前を \characterNo とします。定義を次のとおりです。

```
\makeatletter
\def\characterNo#1{\setcounter{TestCount}{0}
  \@tfor\member:=#1\do{\addtocounter{TestCount}{1}}\theTestCount}
\makeatother
```

何のことはない。要するに、`\itemNo` の定義の中で、`\@for` を `\@tfor` に変えただけです。実際に使ってみましょう。

(例) 文字列中の文字の個数を求める:

```
\characterNo{orange}; \characterNo{apple}; \characterNo{banana}
```

出力はつぎの通りです。

(例) 文字列中の文字の個数を求める: 6; 5; 6

4.2 T_EX 流カウンターとの違い

T_EX のカウンター新設命令 `\newcount` は、新しいカウンターを作成するという単機能の命令です。これに対して、L^AT_EX 2_ε のカウンター新設命令 `\newcounter` は、次のような複合機能をもっています。

1. `\newcounter{CTR}` と宣言すると、同名のカウンターがないかどうかをチェックします。あれば、警告を出して、処理を中断します。
2. カウンター CTR を新設します。ただし、T_EX の側からみると、カウンターは `\c@CTR` です。このように、`\c@` が頭についた制御綴が作成されます。
3. カウンター CTR の現在値を出力するための `\theCTR` 命令が自動的に作成されます。
4. オプションで上位のカウンター SUPCTR に従属させることができます。たとえば、
5. `\newcounter{CTR}[SUPCTR]` と宣言すると、カウンター SUPCTR が増えるたびにカウンター CTR が 0 のリセットされるようになります。これは、内部で新設された `\c1@CTR` によるものですが、ここでは説明を省略します。
6. `\label` による参照用に `\p@CTR` という空の制御綴が新設されます。このあたりもむずかしいので、ずっと後で機会があれば説明することにしましょう。

L^AT_EX 2_ε でのカウンター CTR が、T_EX の裸の命令では、`\c@CTR` であることは、つぎのようにすれば確かめられます。

```
\makeatletter
\setcounter{TestCount}{500}
\the\value{TestCount} = \the\c@TestCount \par
\c@TestCount=600
\the\value{TestCount} = \the\c@TestCount \par
\makeatother

500 = 500
600 = 600
```

4.3 カウンターの大小による分岐処理

カウンターの値を評価して、その大小によって処理を分岐したいことがあります。T_EX には、`\ifnum` という条件文が使えるようになっています。L^AT_EX 2_ε でも、条件文を使うときは、これを使わざるをえないとおもいます。文字列の文字数が5文字以上になるとそれ以上は出力しないような機能をもつマクロを L^AT_EX 2_ε 流で作ってみましょう。命令の名前を `\shortenchar` としましょう。上で定義した `\characterNo` 命令を少し変えます。

```
\makeatletter
\def\shortenchar#1{\setcounter{TestCount}{0}%
  \@tfor\member:=#1\do{\addtocounter{TestCount}{1}%
  \ifnum\c@TestCount<6 \member\fi}}
\makeatother
```

`\ifnum` と `\fi` で条件文を構成しています。カウンター `TestCount` は内部では、`\c@TestCount` として取り扱っていることに注意してください。この場合は、5文字までは `\member` として取得した文字をそのまま出力しますが、6文字以上になると、何もしないということになります。実際に使ってみましょう。

(例) 文字列中を5文字以下に約める:

```
\shortenchar{orange}; \shortenchar{apple};
\shortenchar{banana}; \shortenchar{plum}
```

とかきますと、次のような出力が得られます。

(例) 文字列中を5文字以下に約める: orang; apple; banan; plum

ここでちょっとしたご注意。上記の定義の `\shortenchar` 命令の定義の中で、行末ごとに%が宣言されていますが、この場合には必要です。実際にこれを除去して処理してご覧なさい。文字と文字の間に余分の空白が入ってしまうことがわかるとおもいます。このあたりの事柄は、ときとして出力の仕上がりの上で、致命的になることもあります。%の有無の違いの理由を考えることもおもしろいですよ。また、前の章で、T_EX 流に `\shortenchar` 命令を定義したときには、行末には%が入っていないことにも注意してください。その上で、L^AT_EX 2_ε と T_EX でこのような違いが出る理由も考えてみてください。

さて、本題に戻って、こんなことをして何の役にたつのだとお思いになる諸兄もいらっしゃるでしょう。ここで、これまでの知識を総合して、実用に耐えるマクロを作ってみます。`\Andfor` というマクロは、`Fig.\ \Andfor{A,B,C}` のように入力すると、「Fig. A, B, and C」のように整形する機能をもつものです。前章で述べた `\andfor` とは異なり、3項以上の列挙のときは、`and` の前にコンマを入れることにします。L^AT_EX 2_ε 流の定義と使い方を示しておきましょう。

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Analog to \@for by Shinsaku Fujita 1995/05/26
% Analog to \@for by Shinsaku Fujita 1995/07/30 改訂
%
% \andfor{item1,item2,...}
%
\makeatletter
\def\Andfor#1{\setcounter{enumi}{\m@ne} \setcounter{enumii}{\z@}%
```

```

\@for\member:=#1\do{\addtocounter{enumi}{\@ne}}% 文字列リストの要素の合計-1
\@for\member:=#1\do{\addtocounter{enumii}{\@ne}}%文字列リストの要素の個数
\ifnum\c@enumii<\c@enumi%合計-1 より小さいとき
  \ifnum\c@enumi<1\relax\else\member, \fi%最初の一個は, 以外コンマ付きで出力
\else%合計-1 以上, すなわち最後の二つの要素の処理
  \ifnum\c@enumi<2\relax%合計個数 2 以下
    \ifnum\c@enumii=1\relax \member%合計個数 2 以下の一番目
      \else \space and \member%合計個数 2 の二番目の出力
    \fi
  \else%合計個数 2 以上
    \ifnum\c@enumii=\c@enumi \member,\space and %最後から 2 つ目
      \else \member%最後のもの
    \fi\fi\fi}}
\makeatother

```

この命令を使うには, 次のように書きます.

```

(例) \Andfor{orange}; \Andfor{orange,apple};
\Andfor{orange,apple,banana}; Fig.\ \Andfor{1,2,3,4,5,6}

```

出力は次のようになります.

```

(例) orange; orange and apple; orange, apple, and banana; Fig. 1, 2, 3, 4, 5, and 6

```

\Andfor というマクロの内部でどんなことを行っているかを説明しておきましょう. まず, カウンターとして, enumi と enumii を使っています. これは, L^AT_EX 2_ε の箇条書環境の中で一時的に使うカウンターとして宣言済みのものなので, \newcounter で宣言せずとも使用できます. \m@ne は, -1 と同義, \z@ は 0 と同義です. \@for を 2 回使っていますが, 最初は, 文字列の要素の個数を調べて, 合計 -1 としたものをカウンター enumi に格納しています. この方法は, \itemNo の手法と同じです. 二番目の \@for では, 文字列の要素の数をカウンター enumii に格納し, これと合計 enumi の値と比較しています. 条件文の中では, 対応する制御綴 \c@enumii と \c@enumi を使っていることに注意してください. この例のように \ifnum は入れ子で多重に使用することができます. また \else 命令も注目してください.

第5章 練習 その1

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

これまで出てきたテクニックのおさらいです。解答を直後に書いてありますが、自分で解いてみてください。

5.1 部分文字列の取得

問題 5.1 `\takeinit{shinsaku}{fujita}`として、この命令で、S. Fujita という文字列がえられるようなマクロを定義しなさい。

解答

```
\makeatletter
\def\t@keinit#1#2/{\uppercase{#1}. \ }
\def\t@@keinit#1#2/{\uppercase{#1}#2}
\def\takeinit#1#2{\expandafter\t@keinit#1/
\expandafter\t@@keinit#2/}
\makeatother
\takeinit{shinsaku}{fujita}
```

とすれば、つぎの出力が得られます。 S. Fujita

問題 5.2 jbook スタイルでは、

```
\newcommand{\@chapapp}{\prechaptername}
\newcommand{\@chappos}{\postchaptername}
\newcommand{\prechaptername}{第}
\newcommand{\postchaptername}{章}
```

と定義されており、`\@chapapp\thechapter\@chappos` という文字列で、「第 1 章」などの章のタイトルが出力されるようになっています。この文字列をそのまま利用して、「Chapter 1」となるようにしなさい。

解答 これは、簡単です。

¹©藤田 眞作「L^AT_EX 2_εまぐろの八衢」(04)

```
\renewcommand{\prechaptername}{Chapter~}
\renewcommand{\postchaptername}{}
```

とすればよく、実際に

```
\makeatletter
\@chapapp\thechapter\@chappos
\makeatother
```

と入力すると、出力は次の通りです。Chapter 5

問題 5.3 `\tenjokyo{U.S.A.}`としたとき、この命令によって、USA のようにピリオドを除去するマクロ `\tenjokyo` を定義しなさい。

ヒント 二つの制御綴に格納されている文字が同じかどうかは、`\ifx... \fi` の条件文を使います。ピリオドを、`\def\tenn{.}` のように書いて、制御綴 `\tenn` に置き換えておきます。

解答

```
\makeatletter
\def\tenn{.}
\def\tenjokyo#1%
{\@tfor\member:=#1\do{\ifx\member\tenn\else\member\fi}}
\makeatother
\tenjokyo{U.S.A.} \tenjokyo{U. S. A.}
```

とすれば、次の出力が得られます。`\@tfor` の処理のときに、空白も除去されています。USA USA

問題 5.4 問題 5.3 で定義した `\tenjokyo` を、`U.S.A` でも `U. S. A.` でも `U.\ S.\ A.\` でも対処できるようにしなさい。

解答

```
\makeatletter
\def\tenn{.}
\def\skuhaku{\ }
\def\tenjokyo#1%
{\@tfor\member:=#1\do{\ifx\member\tenn\else%
\ifx\member\skuhaku\else\member \fi\fi}}
\makeatother
\tenjokyo{U.S.A.} and \tenjokyo{U. S. A. } and
\tenjokyo{U.\ S.\ A.\ } \par
```

とすれば、次の出力が得られます。USA and USA and USA

5.2 カウンター

問題 5.5 `\theequation` 命令を再定義して, 数式番号をローマ数字の大文字小文字の組み合わせ (たとえば VII-i) で出力するようにしなさい.

解答

```
\makeatletter
\def\theequation{\Roman{chapter}\mbox{-}\roman{equation}}
\makeatother
\begin{equation}
x=a+b
\end{equation}
```

とすれば, 次の出力が得られます.

$$x = a + b \quad (\text{V-i})$$

問題 5.6 `\元号{1995}` のように西暦年号を入力して, 「平成 7」のように出力する命令を作りなさい.

解答

```
% 藤田 眞作 著 『化学者・生化学者のための LaTeX』
% 東京化学同人, 東京 (1993) 付録
% nippon.sty <Jan 3 1993> by Shinsaku Fujita より引用
\makeatletter
\newcounter{seireki}
\def\元号#1{\c@seireki=#1
\ifnum\c@seireki>1988 \advance\c@seireki by -1988 平成\arabic{seireki}
\else
\ifnum\c@seireki>1925 \advance\c@seireki by -1925 昭和\arabic{seireki}
\else
\ifnum\c@seireki>1911 \advance\c@seireki by -1911 大正\arabic{seireki}
\else
\ifnum\c@seireki>1867 \advance\c@seireki by -1867 明治\arabic{seireki}
\fi\fi\fi\fi}
\makeatother
\元号{1995}, \元号{1900}
```

とすれば, 次の出力が得られます. 平成 7, 明治 33

第6章 カウンターの親子関係と一括リセット

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

今回は、 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X } 2_{\epsilon}$ のカウンター体系の上級編です。相互参照のときにどのようにカウンターに取り扱えばよいかを説明します。 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X } 2_{\epsilon}$ では、section カウンターと subsection カウンターのように従属関係にある組があります。この場合、親の section カウンターが1増えたときに、子の subsection カウンターは0にリセットしなければなりません。従属するカウンターが複数個存在する場合にも一括して対処するため、 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X } 2_{\epsilon}$ では、可変命令を含むリストをもっています。

6.1 カウンター値の参照と一括リセット命令

カウンターについては、拙著第15章に詳しい説明がありますので、参照してください²。ここでは、一括リセットに必要な項目を補足します。

カウンター値が`\label`と`\ref`で相互参照できることは、ご存じのとおりです。ここで、`\label{sec:a1}`を宣言しますと、章の番号が相互参照されます。`\ref{sec:a1}`と書けば、6.1のようにカウンター値が出力されます。現在の参照カウンターの値は、`\@currentlabel`の中に保存されています。たとえば、ここで、

```
\makeatletter
(例) 現在の参照カウンターの値 = \@currentlabel
\makeatother
```

のように書きますと、この値を取り出すことができます。(例) 現在の参照カウンターの値 = 6.1

これらのカウンターの値を更新するには、次の二つの命令が使われています。普通の $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X } 2_{\epsilon}$ の使い方では、滅多に使うことはありませんが、自作のマクロを作る場合には、使い方を理解しておく必要があります。

- `\stepcounter` 引数で指定したカウンターの値を1だけふやし、従属するカウンターの値を一括して0にリセット。
- `\refstepcounter` 引数で指定したカウンターの値を1だけふやし、従属するカウンターの値を一括して0にリセット。さらに、`\label{参照キー}`によって、更新されたカウンター値を参照できるようにする。このとき、カウンター値は、`\@currentlabel`に保存される。

¹©藤田 眞作「 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X } 2_{\epsilon}$ まくろの八衢」(4a)

²本書では、拙書として引用する場合は、次のものを差します: 藤田 眞作 著「化学者・生化学者のための $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ —パソコンによる論文作成の手引」, 東京化学同人 (1993), ISBN4-8079-0386-1.

このように、`\stepcounter` と `\refstepcounter` とに共通する機能として、大きく二つがあります。すなわち、(1) 引数で指定したカウンターの値を 1 だけふやすことと (2) 従属するカウンターの値を一括して 0 にリセットすることです。この節では第一の機能の概略を調べることにします。

第一の機能を実際に試してみましょう。`\stepcounter` の例です。次のように、入力します。カウンターとして `TESTCount` を使います。このカウンターの出力は、たとえば 6-1 のように、ハイフンを介して章の番号を先頭に付けるようにします。

```
\makeatletter
\newcounter{TESTCount}
\def\theTESTCount{\thechapter-\the\c@TESTCount}
\setcounter{TESTCount}{0}
第 1 項 \stepcounter{TESTCount} \theTESTCount [\the\c@TESTCount] \par
第 2 項 \stepcounter{TESTCount} \theTESTCount [\the\c@TESTCount] \par
第 3 項 \stepcounter{TESTCount} \theTESTCount [\the\c@TESTCount] \par
\makeatother
```

その結果、次の出力がえられます。この出力では、角括弧の中に、`TESTCount` カウンターの裸の値を併記しました。

```
第 1 項 6-1[1]
第 2 項 6-2[2]
第 3 項 6-3[3]
```

同じカウンターを引き続いて使い、`\refstepcounter` 実際に試してみましょう。次のように、入力します。`\label` と `\ref` による相互参照も行います。

```
\makeatletter
第 4 項 \refstepcounter{TESTCount} \theTESTCount [\the\c@TESTCount]
(\@currentlabel) \label{test:a} \par
第 5 項 \refstepcounter{TESTCount} \theTESTCount [\the\c@TESTCount]
(\@currentlabel) \label{test:b} \par
第 6 項 \refstepcounter{TESTCount} \theTESTCount [\the\c@TESTCount]
(\@currentlabel) \label{test:c} \par
相互参照の出力: \ref{test:a}, \ref{test:b}, \ref{test:c} \par
\makeatother
```

その結果、次の出力がえられます。この出力では、角括弧の中に、`TESTCount` カウンターの裸の値をしましました。また、丸括弧の中には、`\@currentlabel` の出力を併記しました。これは、対応する `\theTESTCount` の出力と同じであることを確かめてください。

```
第 4 項 6-4[4] (6-4)
第 5 項 6-5[5] (6-5)
第 6 項 6-6[6] (6-6)
```

相互参照の出力: 6-4, 6-5, 6-6

これが、`\label` と `\ref` による相互参照が可能なカウンターの基本的な使い方です。

6.2 一括リセットの仕組み

`\stepcounter` と `\refstepcounter` とに共通する機能のうち、第二のもの (従属するカウンターの値を一括して 0 にリセットすること) を、この節で調べることにします。

6.2.1 可変命令を含むリスト

カウンター CTR が作られたときに、`\c1@CTR` なる制御綴が自動発生することをすでに述べましたが、実は、これが、可変命令を含むリストです。この機能を説明するために、いくつかの要素技術にわけて解説します。

まず、可変命令を含むリストとは、どんなものかを説明しましょう。`\TESTlist` を次のように定義します。この中に含まれる `\tempCS` がここでいう可変命令です。

```
\def\TESTlist{\tempCS{CNTa}\tempCS{CNTb}\tempCS{CNTc}}
```

このようなリストを作ったのち、`\tempCS` に制御綴を代入してみます。ここでは、大文字を小文字に変換する `\lowercase` を `\let` 命令を用いて代入しましょう。そして、`\TESTlist` を実行してみます。

```
{\let\tempCS=\lowercase
(リストの実行) \TESTlist}
```

この結果は、次のように `\tempCS` の引数の文字列がすべて小文字になります。(リストの実行)cntacntbcntc

`\TESTlist` を使えば、こんなこともできます。`\csname... \endcsname` は、この間に挟んだ引数を含む制御綴を作成し実行する機能をもっています。たとえば引数が `CNTa` であると、`\CNTa` なる制御綴を作り実行します。

```
\def\CNTa{\alpha$} \def\CNTb{\beta$} \def\CNTc{\gamma$}
\def\toGreek#1{\csname #1\endcsname}
\def\toGreeks{\let\tempCS=\toGreek \TESTlist}
```

のように定義しておいて、`\toGreeks` を宣言しますと、 $\alpha\beta\gamma$ のように出力されます。

このように、`\TESTlist` のようなリストを作り上げることができれば、`\tempCS` にいろいろな制御綴を代入することにより、一括処理ができるわけです。カウンターなどの一括リセットもこの方法で行えます。たとえば、同じリスト `\TESTlist` から出発して、別のことを行ってみましょう。

```
\def\TLreset#1{\csname #1\endcsname=0\relax}
{\newcount\CNTa \newcount\CNTb \newcount\CNTc
\CNTa=100 \CNTb=150 \CNTc=170
(リストの実行前) \the\CNTa, \the\CNTb, \the\CNTc \par
\let\tempCS=\TLreset \TESTlist
(リストの実行後) \the\CNTa, \the\CNTb, \the\CNTc}
```

と入力すると、次のような結果がえられます。

```
(リストの実行前) 100, 150, 170
(リストの実行後) 0, 0, 0
```

`\TLreset` はカウンターの制御綴を作って、0 とおくという命令です。この命令を `\let` によって `\tempCS` に代入しています。この結果、`\TESTlist` による一括処理が行えるわけです。出力を調べるとわかるように、見事に三つのカウンターの一括リセットが行えました。

6.2.2 リストの作成命令

次にリストの作成方法を考えてみましょう。空のリスト `\TESTlist` から出発して順次内容を入れてゆきます。このとき文字列としての `\tempCS` と将来引数となるべき文字列を入れてゆきます。このとき、`\tempCS` を `\relax` (なにもしない命令) に置き換えて無効にしておきます。次のように定義します。

```
\def\TESTlist{}
\def\TLset#1#2{\begingroup\let\tempCS=\relax
  \xdef#1{#1\tempCS#2}\endgroup}
\def\ADDtoTL#1{\TLset\TESTlist{#1}}
```

この中で、`\ADDtoTL` が順次内容を入れてゆくための命令です。

```
\ADDtoTL{CNTa} \ADDtoTL{CNTb} \ADDtoTL{CNTc}
```

として、`\TESTlist` に内容を入れてゆきます。このときは、上と同じリストがえられます。すなわち、

```
\tempCS{CNTa}\tempCS{CNTb}\tempCS{CNTc}
```

が、得られた `\TESTlist` の内容です。このことは、内部の変換命令 `\tempCS` に `\relax` を代入して、`\tempCS` の引数の文字列をそのまま出力してみれば一目瞭然です。すなわち、

```
\def\typeTL{\let\tempCS=\relax\TESTlist}
(リストの内容表示) \typeTL
```

と入力しますと、次のように出力されます。(リストの内容表示) CNTaCNTbCNTc

また、上の `\lowercase` を使った例をそのまま試してみることもよっても、`\TESTlist` に格納された内容を確認することができます。

```
{\let\tempCS=\lowercase
(リストの実行) \TESTlist}
```

この結果は、次のようになります。(リストの実行) cntacntbcntc

本項で得られた `\TESTlist` の内容は、前項で述べたものと同じですから、内部の変換命令 `\tempCS` の引数をカウンター名であるとみなせば、一括リセットにそのまま利用できることは明らかです。前項の最後の部分をご覧ください。

6.3 *L^AT_EX 2_ε* カウンターリスト

L^AT_EX 2_ε では、親のカウンターが 1 あがるたびに、子のカウンターの値がすべて 0 にリセットされます。前節で準備ができましたので、いよいよ、この一括リセットの仕組みの説明に取りかかりましょう。

```
\newcounter{oya}
\newcounter{musuko}[oya]
\newcounter{musume}[oya]
```

のように、宣言しますと、`oya` カウンターを親として、`musuko` カウンターと `musume` カウンターを従属させることができます。この従属関係は、`\cl@oya` というリストに順次保存されてゆきます。このときは、`\cl@oya` の内容は、次のようになっています。`\@elt` が、上でのべた可変命令です。

```
\@elt{musuko}\@elt{musume}
```

このことは、可変命令 `\@elt` に適当な命令を代入して、`\cl@oya` を出力させるとわかります。ここでは、`\@elt` に `\uppercase` を代入し、文字列を大文字にして出力してみましょう。すなわち、

```
{\makeatletter
\let\@elt=\uppercase
(リストの内容) \cl@oya
\makeatother}
```

のように書きますと、次のよう出力されます。(リストの内容) MUSUKOMUSUME

マクロの中では、`\csname... \endcsname` という制御綴の作り方もよく行われますので、上と同じことをこれを使って書いておきましょう。

```
{\makeatletter
\expandafter\let\csname @elt\endcsname=\uppercase
(リストの内容) \csname cl@oya\endcsname
\makeatother}
```

のように書きますと、次のよう出力されます。(リストの内容) MUSUKOMUSUME

実際に、カウンターがリセットされる様子確かめてみましょう。次のように、各カウンターに値を与えておきます。

```
\setcounter{oya}{15} \setcounter{musuko}{5} \setcounter{musume}{6}
```

このようにしたのち、`\cl@oya` を実行してみましょう。このとき、上で作ったリセットの命令 `\TLreset` を少し改変して、可変命令 `@elt` に代入します(カウンターは、`\c@musuko` と `\c@musume` であることに注意)。

```
{\makeatletter
\def\TLreset#1{\csname c@#1\endcsname=0\relax}
(リストの実行前) \themusuko; \themusume \quad
\let\@elt=\TLreset \cl@oya
(リストの実行後) \themusuko; \themusume
\makeatother}
```

のように書きますと、次のよう出力されます。(リストの実行前) 5; 6 (リストの実行後) 0; 0

ここに説明した `\TLreset` 命令は、 \LaTeX 2_ϵ の `\refstepcounter` 命令の一括リセットの機能の本質的な部分を取り出したものといえます。いいかえると、`\refstepcounter` を使うことによって、このようなりセットが内部で行われているわけです。ついでに、この命令を使った例を示しておきます。

```
\setcounter{oya}{15} \setcounter{musuko}{5} \setcounter{musume}{6}
(リストの実行前) \themusuko; \themusume \quad
(親カウンターを1進める) \refstepcounter{oya} \quad
(リストの実行前) \themusuko; \themusume \quad
```

のように書きますと、次のように出力されます。(リストの実行前) 5; 6 (親カウンターを1進めて出力)
16 (リストの実行前) 0; 0

6.4 余計なこと

`\newcounter{CTR}`の説明のときに、自動的に`\p@CTR`なる制御綴が発生するといいました。これは、箇条書環境などでの、親カウンターを記憶しておく目的をもっています。これを `enumerate` 環境で調べてみましょう。各階層(天地人)に`\label`を書き入れて、そのあとに

```
ラベル: \@currentlabel;
カウンター: \the\c@enumi; \the\c@enumii; \the\c@enumiii;
           \the\c@enumiv;
親カウンター: \p@enumi; \p@enumii; \p@enumiii; \p@enumiv;
```

を入れてあります。次のような出力がえられます。制御綴の空のところは何も出力されていませんが、セミコロンで区切ってありますので、判断がつくとおもいます。ここで、`\c@enumi`などはカウンターですが、`\p@enumi`などは文字列になっていることに注意してください。このため、カウンターの場合は出力するためには、`\the`が必要です。

1. 天一 ラベル: 1; カウンター: 1; || 6; 0; 0; 親カウンター: ; || 1; 1(f); 1(f);
 - (a) 地一 ラベル: 1a; カウンター: 1; 1; || 0; 0; 親カウンター: ; 1; || 1(a); 1(a);
 - i. 人一 ラベル: 1(a)i; カウンター: 1; 1; 1; || 0; 親カウンター: ; 1; 1(a); || 1(a)i;
 - ii. 人二 ラベル: 1(a)ii; カウンター: 1; 1; 2; || 0; 親カウンター: ; 1; 1(a); || 1(a)ii;
 - (b) 地二 ラベル: 1b; カウンター: 1; 2; || 2; 0; 親カウンター: ; 1; || 1(b); 1(b)ii;
 - (c) 地三 ラベル: 1c; カウンター: 1; 3; || 2; 0; 親カウンター: ; 1; || 1(c); 1(c)ii;
 - (d) 地四 ラベル: 1d; カウンター: 1; 4; || 2; 0; 親カウンター: ; 1; || 1(d); 1(d)ii;
2. 天二ラベル: 2; カウンター: 2; || 4; 2; 0; 親カウンター: ; || 2; 2(d); 2(d)ii;

これをみると、`\@currentlabel`の出力は、親カウンター + 現カウンターとして出力されていることがわかります。これは、`\@currentlabel`の定義によって確かめてみてください。たとえば、地二のところでは、親`\p@enumii = 1`、現`\c@enumii = b`(カウンター値は2)で、その結果1aと出力されています。人一のところでは、親`\p@enumiii = 1(a)`、現`\c@enumiii = i`(カウンター値は1)で、その結果1(a)iと出力されています。現階層より下のもの(||で表示)は、旧の値が残っていますので無効です。

第7章 カウンターの親子関係の削除

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

L^AT_EX 2_εのカウンター体系をさらに調べてみましょう。カウンターの親子関係を削除するにはどうしたらよいかを説明します。この章は、かなりむずかしいので、とばしてもかまいません。cntreset.sty というスタイルファイルに仕立ててありますので、これをオプションとすることによって\delfromreset 命令が使えます。

7.1 ソースリスト

まず、オプションスタイル cntreset のソースリストを示します。使うときは、ファイル名を cntreset.sty として格納してください。例文まで、いっしょにコピーするようにしてください。

```
% cntreset.sty
% Copyright (C) 1995 May 27 by Shinsaku Fujita. All right reserved
%
% \delfromreset{CTR}{SUPCTR}
%
% Cancel the relationship between CTR and SUPCTR that is set by
% \newconter command
%
\typeout{Option Style 'cntreset' 1995 May 27 by Shinsaku Fujita}
\def\Comm@{,}
\def\doubleq{==}
\def\delfromreset#1#2{%
\edef\@Tempa{\csname del@parent\endcsname}
\expandafter\gdef\csname del@parent\expandafter\endcsname%
{\@Tempa\Comm@#2\doubleq #1}}

\def\dl@prnt{} \def\dl@son{}
\def\del@parent{#1}=={#2}}
\def\p@rntson#1==#2/{\gdef\dl@prnt{#1}\gdef\dl@son{#2}}
```

¹©藤田 眞作「L^AT_EX 2_εまぐろの八衢」(4b)

```

\newcount\TestCount

\def\@stpelt#1{\TestCount=0 \def\@Tempb{#1}
\ifx\@Tempa\empty \else
\@for\member:=\@Tempa\do{\expandafter\p@rntson\member/\relax
\ifx\PCNT\dl@prnt\relax \ifx\@Tempb\dl@son \relax
\global\advance\TestCount\@ne \fi\fi}\fi
\ifnum\TestCount=0 \global\csname c@#1\endcsname \z@ \fi\relax}

\def\stepcounter#1{\xdef\PCNT{#1} \xdef\@Tempa{\del@parent}%
\global\expandafter\advance\csname c@#1\endcsname \@ne
{\let\@elt\@stpelt \csname cl@#1\endcsname}}

\endinput
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\documentclass[a4j]{jarticle}
\usepackage{cntreset}
\begin{document}
(例文の本体は省略)
\end{document}

```

7.2 機能と使い方

`cntreset.sty` はオプションファイルになっていますが、本章では、この箇所直前に、命令の定義を書き込んであります。このオプションで使える `\delfromreset` は二つの引数をとります。

```
\delfromreset{CTR}{SUPCTR}
```

CTR は SUPCTR カウンターの子として既に宣言してあるカウンターとします。この宣言で、CTR カウンターと SUPCTR の親子関係は削除されます。`\@addtoreset{CTR}{SUPCTR}` 命令とちょうど逆の働きもっています。

たとえば、ここで `oya` カウンターをつくり、`musuko`, `musume`, `mei`, `mago` のカウンターを従属させてみましょう。次のように書きます。

```
% the setting of counters
\newcounter{oya}
\newcounter{musuko}[oya] \newcounter{musume}[oya]
\newcounter{mei}[oya] \newcounter{mago}[oya]
```

これで、親子関係が設定されました。それぞれのカウンターに値をいれます。次のように書きます。

```
% the setting of counter values
\setcounter{oya}{50}
\setcounter{musuko}{15} \setcounter{musume}{20}
\setcounter{mei}{30} \setcounter{mago}{2}
```

つぎに、この値を出力したのちに、`\stepcounter` で oya カウンターの値を一つ増やしてみましょう。次のように入力します。

```
% original values
(もとのカウンターの値)
\theoya, \themusuko, \themusume, \themi, \themago \par
% step counter
\stepcounter{oya}
% reset counter values
(親子関係の解除後, リセット)
\theoya, \themusuko, \themusume, \themi, \themago
```

当然のことながら、従属するすべてのカウンターは 0 にリセットされます。

```
(もとのカウンターの値) 50, 15, 20, 30, 2
(親子関係の解除後, リセット) 51, 0, 0, 0, 0
```

次に、`\delfromreset` の機能の実行例です。ここで、`\setcounter` 命令で、もとの値に戻しておきます。その上で、親子関係をはずしてみましょう。ここでは、oya カウンターから、mago, mei カウンターを切り離します。このため、

```
\delfromreset{mago}{oya} \delfromreset{mei}{oya}
```

と宣言します。つぎに、まえと同様に、もとの値を出力したのちに、`\stepcounter` で oya カウンターの値を一つ増やしてみましょう。変わったところは、一部の親子関係を削除してあるところです。次のように入力します。

```
% original values
(もとのカウンターの値)
\theoya, \themusuko, \themusume, \themi, \themago \par
% step counter
\stepcounter{oya}
% reset counter values
(親子関係の解除後, リセット)
\theoya, \themusuko, \themusume, \themi, \themago
```

出力は、次の通りです。oya カウンターから、mago, mei カウンターが切り離されていますので、これらの値は`\stepcounter{oya}`を宣言したのちも、リセットされません。ほかの、musuko, musume カウンターは、oya に従属したままですから、リセットされています。

```
(もとのカウンターの値) 50, 15, 20, 30, 2
(親子関係の解除後, リセット) 51, 0, 0, 30, 2
```

`\delfromreset` 命令は、たとえば jarticle スタイルで、subsection カウンターを section カウンターから切り離すようなときに使えます。これによって、項番号 (subsection カウンター) は、節番号 (section カウンター) とは無関係に連続して振られるようになります。たとえば、

```

\documentstyle[a4j]{jarticle}
\usepackage{cntreset}
\begin{document}
\delfromreset{subsection}{section}
\def\thesubsection{\Alph{subsection}}
\def\thesection{\arabic{section}}
\section{節の表題}
\subsection{項の表題}
\subsection{項の表題}
\subsection{項の表題}
\section{節の表題}
\subsection{項の表題}
\subsection{項の表題}
\subsection{項の表題}
\end{document}

```

のようにしますと、次のようになります。ここでは、項の出力を大文字のアルファベットで行うように `\thesubsection` 命令を再定義しています。

```

1      節の表題
      A 項の表題
      B 項の表題
      C 項の表題
2      節の表題
      D 項の表題
      E 項の表題
      F 項の表題

```

7.3 マクロ解説

7.3.1 全体の方針

ある親カウンター `SUPCTR` が設定されたときには、子にあたるカウンターをリスト形で `\c1@SUPCTR` に保存してあることは、すでに述べました。親子関係を削除するためには、このリストの中から該当の子カウンターに関する記述を削除すればよいのですが、これがなかなかむずかしい²。

そこで、本章では、別の方法を採用することにしました。まず、削除すべき親子関係を `\del@parent` というリストに保存しておきます。`\stepcounter` の実行時に (具体的には `\@elt` 命令に `\@stpelt` 命令を代入する際に)、このリストを参照します。リストに含まれていれば、該当のカウンターのリセット処理を行わないようにする方法です。このためには、

1. 削除すべき親子関係を保存するリスト `\del@parent` の保存形式の決定
2. この方針によって、削除すべき親子関係を保存するリスト `\del@parent` を作成するための命令として、`delfromreset` を作成

²この方向での解決がいちばんすっきりすると思うのですが、どなたかよいアイデアはありませんか。

3. \stepconter 命令およびその内部命令\@stpelt の再定義

を行う必要があります。

7.3.2 リストの作成

リストの形式は、\@for 命令が使えるように、コンマで区切ったものとします。各部分は SUPCTR==CTR のように親子のカウンターを==でさらに区切っておきます。リストを作るとき、これらの区切り記号を直接に扱うのはやりづらいので、\def\Comm@{,}と\def\doubleq{==}として、制御綴に収めておきます。面倒ですから、\def\del@parent{#{}=={}}として、初期化しておきます。これは、\@for 命令のときに、最初の 1 回は無駄なループを行うことになるのですが、簡便のためにこのようにしました。このような前準備をすると、\delfromreset の定義は、比較的簡単です。

```
\def\delfromreset#1#2{%
\edef\@Tempa{\csname del@parent\endcsname}
\expandafter\gdef\csname del@parent\expandafter\endcsname%
{\@Tempa\Comm@#2\doubleq #1}}
```

この定義では、旧リスト \@Tempa に仮格納し、このあとにコンマ、引数 #2 (親)、記号 ==、引数 #1 (子) を順次つなげて、新しいリストとしています。

7.3.3 リセット命令の再定義

まず、\@for で取り出した文字列 SUPCTR==CTR から、==を区切りにして、SUPCTR と CTR を取り出して、それぞれ、\dl@prnt と \dl@son に収めます。分離する命令 \p@rntson を作ります。

```
\def\p@rntson#1==#2/{\gdef\dl@prnt{#1}\gdef\dl@son{#2}}
```

この命令は、制御綴の引数処理の常法を利用しています。

親 \P@CNT と削除リスト \@Tempa (\del@parent を入れてある) は、あとで述べるように別のところで設定してあります。 \@stpelt の再定義は次の通りです。

```
\def\@stpelt#1{\TestCount=0 \def\@Tempb{#1}
\ifx\@Tempa\empty \else
\@for\member:=\@Tempa\do{\expandafter\p@rntson\member/\relax
\ifx\P@CNT\dl@prnt\relax \ifx\@Tempb\dl@son \relax
\global\advance\TestCount\@ne \fi\fi}\fi
\ifnum\TestCount=0 \global\csname c@#1\endcsname \z@ \fi\relax}
```

再定義した \@stpelt の引数 #1 は、子カウンターの名称 (\@Tempb に格納) です。 \@for 命令により、削除リスト (\@Tempa) を調べてゆき、親カウンターの名称 (\@Tempb) と子カウンターの名称 (\P@CNT) が、分離した \dl@prnt と \dl@son それぞれ一致するかどうかを調べます。一致するとカウンター \TestCount の値を 1 だけ進めます。この判断は、\ifx... \fi の条件文を用いています。最後に、削除すべき親子関係かどうかの判断は、カウンター \TestCount が 0 のままかどうかで行っています。この判断は、\ifnum... \fi の条件文を用いています。

この`\stpelt` 命令は、次に示す`\stepcounter` の定義の中で`\@elt` 命令に代入しています。この処理のときに必要な親`\PCNT` と削除リスト`\@Tempa` (`\del@parent` を入れてある) の設定は、この命令の中で行っています。オリジナルの`\stepcounter` の定義と比較するとわかりますが、異なるのはこの設定の部分です。

```
\def\stepcounter#1{\xdef\PCNT{#1} \xdef\@Tempa{\del@parent}%  
\global\expandafter\advance\csname c@#1\endcsname \@ne  
\let\@elt\@stpelt \csname cl@#1\endcsname}}
```

第8章 相互参照とリスト処理の応用

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

旧@nifty の FPRINT Mes 11 で、`\ref` で参照するカウンター値を数値化したいという質問がありました。このために作成した`\REF` 命令を、前回までに説明して来た事柄を踏まえて、解説をしてみます。

8.1 例題マクロ

私が書き込んだコメントを次に示します (ただし、説明の都合上、一部変更してあります)。

```
01666/01666 HBH00445 藤田眞作 RE:\ref の文字を数値化
(11) 95/05/22 17:06 01663 へのコメント
%#01663 讃岐さん、こんにちは。
%次のはどうですか(^_^)。数値になっていることを確かめるために足し算をするマ
%クロを作ってあります。もしも、\ref の値を使うときは、\tempref がその値です。
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% refno の足し算 by S. Fujita 1995/05/22
% \REF{key}
% \Writerefno
%
% \tempref --- net counter: use this value for your application
%
% 1, 2 ... ==>そのまま
% 1.1, 1.2, ...==> 1, 2, ...
% 1.2.1, 1.2.2... ==>1, 2, ...
%
\makeatletter
%ピリオドで分離するための命令
\def\@periodfor#1:=#2\do#3{\edef\@fortmp{#2}\ifx\@fortmp\@empty \else
\expandafter\@pforl@p#2.\@nil.\@nil\@#1{#3}\fi}
\def\@pforl@p#1.#2.#3\@#4#5{\def#4{#1}\ifx #4\@nnil \else%
#5\def#4{#2}\ifx #4\@nnil \else#5\@piforl@p #3\@#4{#5}\fi\fi}
```

¹©藤田 眞作「*L^AT_EX 2_ε まくろの八衢*」(05)

```

\def\@piforl@p#1.#2\@#3#4{\def#3{#1}\ifx #3\@nnil%
  \let\@nextwhile=\@fornoop \else%
  #4\relax\let\@nextwhile=\@piforl@p\fi\@nextwhile#2\@#3{#4}}
\newcount\tempref%カウンター新設
\newcount\tmpref%カウンター新設
\def\@tempref#1{\global\tempref=#1\relax}%文字列をカウンターに設定
\def\@tempref#1#2\@nil{\gdef\tttempa{#1}}%文字列分離
\def\Cancelrefno{\global\tmpref=0\relax}%カウンターをリセット
\def\Writerefno{\the\tempref \global\tempref=0\relax}%出力
\def\REF#1{\@ifundefined{r@#1}%
  {\bf ??}\@warning{Reference ‘#1’ on page \thepage \space undefined}}%
  {\edef\@tempa{\@nameuse{r@#1}}}%
  {\expandafter\@tempref\@tempa \@nil\null}%
  \@periodfor\member:=\tttempa\do{\xdef\tttemp{\member}}}%
  {\expandafter\@tempref\tttemp}%
  \expandafter\@car\@tempa \@nil\null%
% \the\tempref %net counter; use this value for your application
  \advance\tmpref\tempref}}
\Cancelrefno
\makeatother %end S.Fujita 1995/05/22

```

これらを、プリアンブルに宣言して、次の例文を処理します。

(例文はじまり)

```

\begin{eqnarray}
x = a + b \label{eq:a1} \\
y = c + d \label{eq:a2} \\
z = e + f \label{eq:a3} \\
x = a + b \label{eq:b1} \\
y = c + d \label{eq:b2} \\
z = e + f \label{eq:b3} \\
x = a + b \label{eq:c1} \\
y = c + d \label{eq:c2} \\
z = e + f \label{eq:c3} \\
x = a + b \label{eq:d1} \\
y = c + d \label{eq:d2} \\
z = e + f \label{eq:d3}
\end{eqnarray}
\verb/\REF/で一続きに書けば足してゆきます。 \verb/\Writerefno/で書き出し
\par
\REF{eq:a1} + \REF{eq:d3} = \Writerefno;
\REF{eq:a1} + \REF{eq:a2} + \REF{eq:a3} + \REF{eq:d3} = \Writerefno;
\REF{eq:b1} + \REF{eq:b2} + \REF{eq:b3} + \REF{eq:d3} = \Writerefno;

```

```
\REF{eq:c1} + \REF{eq:c2} + \REF{eq:c3} + \REF{eq:d3} = \Writerefno
```

(例文おわり)

得られる出力は次の通りです .

(例文はじまり)

$$x = a + b \tag{8.1}$$

$$y = c + d \tag{8.2}$$

$$z = e + f \tag{8.3}$$

$$x = a + b \tag{8.4}$$

$$y = c + d \tag{8.5}$$

$$z = e + f \tag{8.6}$$

$$x = a + b \tag{8.7}$$

$$y = c + d \tag{8.8}$$

$$z = e + f \tag{8.9}$$

$$x = a + b \tag{8.10}$$

$$y = c + d \tag{8.11}$$

$$z = e + f \tag{8.12}$$

\REF で一続きに書けば足してゆきます . \Writerefno で書き出し

8.1 + 8.12 = 13; 8.1 + 8.2 + 8.3 + 8.12 = 18; 8.4 + 8.5 + 8.6 + 8.12 = 27; 8.7 + 8.8 + 8.9 + 8.12 = 36(例文おわり)

8.2 部分文字列の取得

上記の \REF の定義のポイントは、\@periodfor 命令を内部で使っていることです . これは、 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ の \@for 命令の定義 (latex.tex) 中のコンマをピリオドに変えただけのものです . 作成した命令を有効にするためには、そのほかに、\@pforl@p と \@piforl@p を作成する必要がありますが、これも \@for 命令の下位命令を今回の目的に合うように小変更したものです . これらの命令のメカニズムは理解するのはむずかしいですが、区切り記号を変更するだけなら、変更すべきところはすぐに見つかります .

実際に \@periodfor 命令を直接に試してみましょう . 処理する文字列は 1.2.44 とします . 部分文字列 (数字) を取り出して、いったん \member に収めます . \do の次の中括弧の処理 (カウンター \tempref に設定) 行います . この場合は簡単で、単に等号で結べばよいのです .

```
\makeatletter
\@periodfor\member:={1.2.44}\do{\tempref=\member}
(例) カウンター\verb/\tempref/の値 (=the\tempref)
\makeatother
```

途中の処理の値は捨てていますので、\@periodfor 命令のくり返し処理が終わった時点では、最後の値 (この場合は 44) が、カウンター \tempref に残っているということになります . この値は、\the 命令により出力することができます . 実際の出力は次のとおりです .

(例) カウンター\tempref の値 (=44)

8.3 参照キーと参照番号の受け渡し

\REF は、L^AT_EX 2_εの\ref を変更したのですが、本質的なものはすべて受け継いでいます。両者に共通な参照キーと参照番号の受け渡しの方法は、L^AT_EX 2_εのこの種のテクニックの特徴がよく現れています。ここで、徹底的に理解しておきましょう。

1. L^AT_EX 2_εの処理時には、aux ファイルが出力されていますが、この中には、

```
\newlabel{eq:d3}{8.12}{3}
```

などの記述があります。これが上記の数式の label{eq:d3}に対応する情報の出力です。すぐにわかるように、eq:d3 が参照キー、8.12 が参照番号、3 がページ番号になっています。ただし、このマニュアルでは、参照番号は 8.12、ページ番号は 49 となっています。

2. L^AT_EX 2_ε処理を複数回行うと、そのたびに aux ファイルが読み込まれます。このときに、上記の \newlabel が実行されます。これによって、\r@eq:d3 なる制御綴 (数字や記号を含んでいますのでこのままのかたちではありませんが、説明の方便でこのようにしています) が発生し、この内容が文字列{8.12}{3}となるようになっていきます。

参考のため、実際に\r@eq:d3 をここで出力してみましょう。数字や記号が入った制御綴は通常では使えませんので、@nameuse を使います。@nameuse というのは引数の文字列の頭に\を付けた制御綴を作成する命令です。

```
\makeatletter
```

```
(例) \verb/\r@eq:d3/に保存されている参照キー = \@nameuse{r@eq:d3}.
```

```
\makeatother
```

と入力しますと、次のような出力がえられます。(例) \r@eq:d3 に保存されている参照キー = 8.1249. 出力のときに、中括弧が消えてしましますが、8.12... は{8.12}{...}から T_EX の出力処理にしたがって出力されたものです (このマニュアルでは、... のところは実際のページ番号 49 が出力されます)。T_EX の仕様上仕方のないことですので、我慢してください。

3. \REF の処理の最初に

```
\edef\@tempa{\@nameuse{r@#1}}%
```

と定義されています。edef というのは、引数を展開してから設定するための命令です。上で説明したように、@nameuse というのは引数の文字列の頭に\を付けた制御綴を作成する命令です。この場合の引数#1 は、\REF 命令の引数 eq:d3 と同じになっていますので、\r@eq:d3 という制御綴 (\r@eq:d3 の内容は、{8.12}{3}) が作成されます。これが、edef で展開されたのちに、\@tempa に設定されるのですから、結局、\@tempa の内容が{8.12}{3}となります。

4. `\@tempref` は 2 個の引数を分離して最初のものだけを残して, `\tttempa` に文字列として格納する働きをもつものです. これを `\@tempa` に作用させるわけです. この過程をみるため, 擬似的に `\@tempa` に文字列を収めて,

```
\makeatletter
\edef\@tempa{{8.12}{3}}
\expandafter\@tempref\@tempa \@nil\null
抽出された番号 (= \tttempa).
\makeatother
```

なる処理を行ってみましょう. 出力は次の通りです. 抽出された番号 (=8.12).

5. この `\tttempa` (文字列は 8.12) に `\@periodfor` を働かせて, `\tttemp` の内容として, 文字列 12 を残します. この処理の過程は, 前節で詳しく説明しました.
6. `\@tempref` 命令によって, `\tttemp` の内容 (文字列 12) を数値化して, カウンター `\tempref` に収めます.

8.4 その他

上記の `\REF` 命令の定義の中には, `\@ifundefined`, `\@warning`, `\@car`, `\@nameuse` など, $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ の有用な隠しコマンドがふくまれています. これらは, 別の機会に説明することにします.

第9章 補助ファイルの活用 (1)

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

L^AT_EX 2_εを使っていると、処理の際にいろいろなファイルが自動発生していることに気がついていらっしゃるでしょう。この中で、aux ファイル (補助ファイル) が、相互参照などを行うために使われているファイルです。今回は、aux ファイルの活用を解説します。

9.1 補助ファイルへの書き込み

aux ファイルを有効に使うためには、L^AT_EX 2_ε処理の際に発生するカウンターの値などの出力手段を知っておく必要があります。これは、実に簡単です。`\@auxout` に aux ファイルの出力ストリーム番号が保存されているので、

```
\makeatletter
\immediate\write\@auxout{\string\relax\string\relax\string\relax}
\makeatother
```

のように指定するだけです。`\immediate` はその場で、すぐに出力するという意味です。`\write` が出力させるための命令です。中括弧の中に出力させたい文字列を書きます。aux ファイルは、L^AT_EX 2_ε処理中に読み込まれるので、ここでは、何もしない命令`\relax`を3個続けて出力するようにします。`\string`は、後続の制御綴を文字列のまま出力する命令です。

この命令を書いたファイルが `test.tex` であるとします。ファイル名が `test.tex` なので、処理後に、`test.aux` が発生しています。これを、エディターを使って覗いてみてごらんください。tex ファイルの内容にもよりませんが、

```
\relax
\relax\relax\relax
```

のように出力されているはずですが、この2行目が、上記の`\write`命令で書き込んだものです。aux ファイルに書き込んだ命令は、aux ファイルが読み込まれたときに、実行されます。しかし、`\relax`は何もしない命令ですから、当然何も起こりません。

これでは、おもしろくありませんから、aux ファイルに書き込んだ命令が aux ファイルが読み込まれたときに実行される場合を試してみましょう。L^AT_EX 2_ε処理のときにエラーになるといけないので、無害の`\typeout`命令を書き込んでみます。

¹©藤田 眞作「L^AT_EX 2_εまくろの八衢」(06)

```
\makeatletter
\immediate\write\@auxout{\string\typeout{試験出力．ただ今テスト中．}}
\makeatother
```

`\typeout` 命令は、ディスプレイに引数に指定した内容を表示する命令です。L^AT_EX 2_ε 処理中に「試験出力．ただ今テスト中．」がディスプレイに表示されるのを確かめてください。

今度も、発生した `test.aux` を、エディターを使って覗いてみてごらん下さい。tex ファイルの内容にもよりますが、

```
\relax
\relax\relax\relax
\typeout{試験出力．ただ今テスト中．}
\@writefile{toc}{\string\contentsline\space {chapter}
  {\string\numberline\space {7}補助ファイルの活用}{1}}
\@writefile{lof}{\string\addvspace\space {10pt}}
\@writefile{lot}{\string\addvspace\space {10pt}}
\@writefile{toc}{\string\contentsline\space {section}
  {\string\numberline\space {7.1}補助ファイルへの書き込み}{1}}
\gdef\CurrPage{2}
\@writefile{toc}{\string\contentsline\space {section}
  {\string\numberline\space {7.2}補助ファイルに書き込まれた命令}{2}}
```

などの出力がえられています (これは、本文書ファイル `timata06.tex` の `aux` ファイルの出力。紙面の都合上、適当に行を折り返してあります)。この中に (3 行目)、確かに、

```
\typeout{試験出力．ただ今テスト中．}
```

が出力されています。

もう一度 L^AT_EX 2_ε 処理をしてください。今度は、処理中に「試験出力．ただ今テスト中．」が 2 回ディスプレイに表示されます。`\typeout` 命令は、`test.aux` が読み込まれたときに有効になります。つまり、この例から、`test.aux` が 2 回読み込まれるのがわかります。ディスプレイに表示されるログの中にも、(`test.aux`) が 2 回読み込まれていることが示されていますので注意してください。

ちなみに、上記の `aux` ファイルの出力のうち、`\@writefile{toc}` は、目次の項目を `toc` ファイル (この場合は `test.toc`) に出力するためのものです。これは、`\tableofcontents` 命令を宣言したときにのみ有効になります。これらについては、別の機会に説明します。

9.2 補助ファイルに書き込まれた命令

もうすこし、実用的なことをしてみましょう。L^AT_EX 2_ε では、`\label` や `\ref` で、相互参照を行っていることは、ご存じの通りです。どうして、こんな複雑なことができるのだろうか、一度は疑問をもたれたことでしょう。すでに、前の章にも一部説明しましたが、この説明を聞いても実際は理解することは、むずかしいとおもいます。

本節では、相互参照のメカニズムの基本的な事項を知るために、実例を示しながら詳しく調べてゆくことにします。まず、次のような命令を、文書の適当なところに書いておきましょう。

```

\makeatletter
\@ifundefined{CurrPage}{\gdef\CurrPage{??}}{}
\immediate\write\@auxout{\string\gdef\string\CurrPage{\number\count0}}
\makeatother

```

`\@ifundefined{A}{B}{C}`なる命令は、`\A`なる制御綴が定義されていないときには処理 B を行い、定義されているときには処理 C を行う機能をもっています。ここでは、`\CurrPage`なる命令の有無をチェックしています。`\write`命令の書式は、すでに説明しました。初出の命令は、`\number`です。これは、カウンターレジスター (カウンターを保存するためのレジスター) の値をアラビア数字で出力する機能をもっています。`\count0`はカウンターレジスターで、 \TeX では、現在のページを保存している特別なものです。

ここで、

(例) 現在のページ番号 = `\CurrPage`

と書いてみましょう。次の出力がえられます。

(例) 現在のページ番号 = ??

1 回目は`\CurrPage`が定義されていないので、このような出力がえられることは、おわかりと思います。処理後に発生した `aux` ファイルを、エディターを使って覗いてみてごらんください。出力の中に、次のような 1 行があるはずですが (ページ番号は、 \LaTeX 2_ϵ 処理によって、異なることもあります)。上の`\write`文の引数の内容と比較してみれば、一層理解が深まります。

```
\gdef\CurrPage{55}
```

ここに書き込まれた`\gdef`は、`\global\def`の略で、`aux`ファイルのそとでもこの定義が有効になるようにする機能をもっています。この結果、2 回目の \LaTeX 2_ϵ 処理以降は、`aux`ファイルのこの命令が実行されますので、次の出力がえられます。

(例) 現在のページ番号 = 55

第10章 補助ファイルの活用 (2)

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

10.1 ページ番号の工夫

10.1.1 第一案

通常のスタイル `jarticle` などでは、ページ番号は $1, 2, \dots$ のように出力されます。これを総ページ (たとえば 25 ページとする) を添えて、 $1/25, 2/25, \dots$ のように出力したいことがあります。この場合、総ページの情報を獲得する必要があります。これは、`aux` ファイルへの書き込みの手法で解決することができます。次に完成したマクロと例文を示します。

```
%test1a.tex by Shinsaku Fujita 1995/05/27
\documentclass{jarticle}
\makeatletter
\@ifundefined{LastPage}{\gdef\LastPage{??}}{}
\def\writeLastPage{\if@files\immediate\write\@auxout{%
  \string\gdef\string\LastPage{\number\count0}}\fi}
\let\origenddocument=\enddocument
\def\enddocument{\writeLastPage\origenddocument}
\def\thepage{\arabic{page}/\LastPage}
\makeatother
%例文
\begin{document}
a \clearpage b \clearpage c \clearpage d \clearpage
e \clearpage f \clearpage g \clearpage h \clearpage i
\clearpage j \thepage
\end{document}
```

これを `test1a.tex` と名付けて、実際に $\text{\LaTeX} 2_{\epsilon}$ 処理してみてください。 $\text{\LaTeX} 2_{\epsilon}$ を処理したときに、`aux` ファイルをみて、何が書き込まれているか確かめるようにしてください。

上記のマクロのを説明しましょう。`\writeLastPage` 命令が肝心なものですが、これは、前回説明したことの応用です。この中には、`\write` 命令が使われています。その引数に書いてあることのうち、 \TeX では、現在のページ番号を `\count0` というカウンターレジスターに保存していることは、前に述べました。これを `\number` 命令で出力すると、ページ番号がアラビア数字で出力されます。`\if@filesw\dots\fi` という条件

¹©藤田 眞作「 $\text{\LaTeX} 2_{\epsilon}$ まくろの八衢」(07)

文は、補助ファイル類を発生させるのスイッチになっています。これについては、のちほど触れることにします。

現在のページ番号を出力する方法は、`\CurrPage` 命令の作成の項で説明しました。この類推ですぐわかるとおもいますが、`\end{document}` が実行されたときに、`\writeLastPage` 命令が実行されるようにすれば、最終ページが `aux` ファイルに書き込まれるようになります。内部の `\write` 命令の引数をみるとわかるように、たとえば、総ページ数が 25 ページの場合は、`aux` ファイルに

```
\gdeg\LastPage{25}
```

のように出力されることがわかります。実際に、`test1a.tex` を一度 L^AT_EX 2_ε 処理したあとで、出力された補助ファイル (`test1a.aux`) をみて確かめること。これで、`aux` ファイルが読み込まれたときに、`\LastPage` の内容として、25 が保存されることになります。

さて、`\end{document}` が実行されたときに、`\writeLastPage` 命令が実行されるようにするにはどうしたらよいのでしょうか。まず、元の `\end{document}` (これは、`\enddocument` として定義されていることに注意) を一時保存しておきます。

```
\let\origenddocument=\enddocument
```

その上で、`\writeLastPage` 命令を追加して、次のように、再定義します。

```
\def\enddocument{\writeLastPage\origenddocument}
```

この再定義の手法は、応用範囲の広いものですから、おぼえておくと役に立ちます。

上記の方法では、`\thepage` の出力は、`\thepage` を再定義してありますので、もとの `jarticle` スタイルのものと同じではありません。たとえば、文中での参照は「`\thepage` ページ」としたときに「1/25 ページ」などと出力されてしまいます。上記の例文の最終ページに `\thepage` としてありますので、出力がこの形式になっていることを確かめてください。また、`\tableofcontents` を宣言した場合も、出力される目次のページ番号は、やはりこの形式になってしまいます。

10.1.2 第二案

柱(ヘッダー)や脚(フッター)へのページ番号の出力は、「1/25」のようにしておいて、文中での参照は「`\thepage` ページ」としたときに「1 ページ」などとし、また、目次への出力も該当ページのみ出力とした場合には、どうすればよいのでしょうか。これが、普通の方法のようにおもわれますので、すこしむずかしいですが、挑戦してみましよう。方針は、`\thepage` の定義はオリジナルの L^AT_EX 2_ε ののままにして、他の部分で変更を実現することです。次に完成したマクロと例文を示します。

```
%test2.tex by Shinsaku Fujita 1995/05/27
%\documentclass{jarticle}
%\documentclass{jbook}
\documentclass{jreport}
\makeatletter
\@ifundefined{LastPage}{\gdef\LastPage{??}}{}
\def\writeLastPage{\if@filesw\immediate\write\@auxout{%
\string\gdef\string\LastPage{\number\count0}}\fi}
```

```

\let\origenddocument=\enddocument
\def\enddocument{\writeLastPage\origenddocument}
%
\ifx\@oddfoot\empty\else
  \def\@oddfoot{\rm\hfil\thepage/\LastPage\hfil}\fi
\ifx\@oddhead\empty\else
  \def\@oddhead{\rm\hfil\thepage/\LastPage}\fi
\ifx\@evenhead\empty\else
  \def\@evenhead{\rm\thepage/\LastPage\hfil}\fi
\makeatother
%例文
\begin{document}
a \clearpage b \clearpage c \clearpage d \clearpage
e \clearpage f \clearpage g \clearpage h \clearpage i
\clearpage j \thepage
\end{document}

```

これを test2.tex と名付けて、実際に L^AT_EX 2_ε 処理してみてください。L^AT_EX 2_ε を処理したときに、aux ファイルをみて、何が書き込まれているか確かめるようにしてください。

さて、マクロの解説です。writeLastPage のところは前項と同じですので、後半の部分のみを説明しましょう。柱（ヘッダー）や脚（フッター）に出力するときのみ「1/25」などとする方法を探りますが、どこを直したらよいのか調べる必要があります。このようなときに肝心なのは、(1) jarticle.sty などのスタイルファイルで該当のところを見つけることと (2) 目星をつけた制御綴の定義を latex.tex (L^AT_EX の基本マクロ集。通常は \tex\macros ディレクトリーの中にある。L^AT_EX 2_ε では、latex.ltx が該当) で見つけることです。これには、かなり経験が必要です。まあ最初は試行錯誤でいろいろと試みるしかないでしょう。

ケーススタディーとして、具体的に、この手順を行ってみましょう。スタイルファイルをエディターで開いて、最後尾をみてください。ps@plain (jarticle.sty と jreport.sty) や ps@heading (jbook.sty) が宣言されているはずですが、これが、柱や脚の出力を決めるための命令なのです。ps@plain の定義は、latex.tex というマクロファイルの中で宣言されています。

```

\def\ps@plain{\let\@mkboth\@gobbletwo
  \def\@oddhead{}\def\@oddfoot{\rm\hfil\thepage
  \hfil}\def\@evenhead{}\let\@evenfoot\@oddfoot}

```

jarticle スタイルなどでは、脚（フッター）にページ番号（ノンブル）が中央揃えで出力されます。この出力の仕様は、\@oddfoot に収められていることがわかります（中央揃えの決まり文句、すなわち、二つの \hfil で \thepage を挟んでいるところに注目）。したがって、この部分を直せばよいことがわかります。

jarticle.sty だけでなく、ほかのスタイルでも使えるようにするには、\@oddfoot が、空 (\empty) でないかどうかを判断して、処理を分岐させます。これは、\ifx 文を利用して実現しています。中央揃えの決まり文句のところを

```

\ifx\@oddfoot\empty\else
  \def\@oddfoot{\rm\hfil\thepage/\LastPage\hfil}\fi

```

のように変えます。これで、中央揃えでノンブルを 1/25 のように出力できることがわかりでしょう。ノンブルが柱(ヘッダー)に現れるときも同様の方法をとります。すなわち、この場合は、`\@oddhead` と `\@evenhead` について、空でないときに再定義するわけです。

スタイルファイルを新しく作成する場合は、このような一時凌ぎの方法ではなくて、正式に `\ps@plain` に相当する命令を作るのがよいと思います。たとえば、次の例が参考になるでしょう。

```
%test3.tex by Shinsaku Fujita 1995/05/27
\documentclass{jarticle}
\makeatletter
%スタイルファイルに入れる
\@ifundefined{LastPage}{\gdef\LastPage{??}}{}
\def\writeLastPage{\if@files\immediate\write\@auxout{%
  \string\gdef\string\LastPage{\number\count0}}\fi}
\let\origenddocument=\enddocument
\def\enddocument{\writeLastPage\origenddocument}
%
\def\ps@totalpagestyle{\let\@mkboth\@gobbletwo
  \def\@oddhead{} \def\@oddfoot{\rm\hfil\thepage/\LastPage
  \hfil} \def\@evenhead{} \let\@evenfoot\@oddfoot}
\ps@totalpagestyle
\makeatother
%例文
\begin{document}
a \clearpage b \clearpage c \clearpage d \clearpage
e \clearpage f \clearpage g \clearpage h \clearpage i
\clearpage j \thepage
\end{document}
```

新しく `\ps@totalpagestyle` 命令を定義しておき、これをスタイルファイルの最後尾などで宣言するわけです。

10.1.3 そのほかの方法

総ページ数の取得に関しては、`\label` と `\ref` を使う方法も報告されています。たとえば岩熊・古川の「L^AT_EX のマクロやスタイルファイルの利用」の第 3.4 項に詳しく説明されていますので、参照してください。本項の方法と比較検討すると、マクロ作成の技術をさらに高めるのに役に立つとおもいます。

第11章 スタイルファイルの仕立て方

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

L^AT_EX 2.09 の時代には、オプションスタイルと呼ばれていましたが、L^AT_EX 2_ε になってからパッケージファイルと呼ばれるようになっていきます。ここでは、拡張子.sty にちなんで、スタイルファイルと呼ぶことにします。いろいろな機能をもったスタイルファイルが出回っています。自作のマクロでも、汎用性がありそうなものは、スタイルファイルとしておけば便利に使えるようになります。手順は、きわめて簡単です。

11.1 スタイルファイルの登録と使用

11.1.1 登録手順

前の章で作ったマクロ (test2.tex) をスタイルファイルに仕立ててみましょう。

1. 名称は、何でもよいのですが、拡張子を必ず.sty とします。ここでは、clpage.sty (current/lastpage) とでもしておきましょう。
2. tex ファイルの中では、\makeatletter と\makeatother の宣言が必要ですが、sty ファイルの中では不要です。
3. 例文を添えると、あととのために便利です。別ファイルとする方法もありますが、短いときはマクロの記載が終わったところで \endinput 命令を書き、それ以降に例文を載せるのが簡便です。
4. 是非とも必要というわけではありませんが、L^AT_EX 2_ε 実行時に、ディスプレイにこのオプションファイルが使われたことを表示しましょう。 \typeout 命令で行います。
5. 作者、日付、バージョンなどをコメントとして書いておきます。これは、後日の保守に役に立ちます。もしも、ほかのファイルを改変したのなら、その旨も記載しておきましょう。

このようにして作ったオプションファイルの例を次に示します。

```
%
%clpage.sty (version 1.00) by Shinsaku Fujita 1995/05/27
%
\typeout{Option Style 'clpage' (v. 1.00) Shinsaku Fujita 1995/05/27}
```

¹©藤田 眞作「L^AT_EX 2_εまぐろの八衢」(08)

```

\@ifundefined{LastPage}{\gdef\LastPage{??}}{}
\def\writeLastPage{\if@filesw\immediate\write\@auxout{%
  \string\gdef\string\LastPage{\number\count0}}\fi}
\let\origenddocument=\enddocument
\def\enddocument{\writeLastPage\origenddocument}
%
\ifx\@oddfont\empty\else
  \def\@oddfont{\rm\hfil\thepage/\LastPage\hfil}\fi
\ifx\@oddhead\empty\else
  \def\@oddhead{\rm\hfil\thepage/\LastPage}\fi
\ifx\@evenhead\empty\else
  \def\@evenhead{\rm\thepage/\LastPage\hfil}\fi
\endinput%これ以下は無視される

```

%例文

```

%test2.tex by Shinsaku Fujita 1995/05/27
%\documentclass{jarticle}
%\documentclass{jbook}
\documentclass{jreport}
\usepackage[clpage]
\begin{document}
a \clearpage b \clearpage c \clearpage d \clearpage
e \clearpage f \clearpage g \clearpage h \clearpage i
\clearpage j \thepage
\end{document}
%the end of clpage.sty by Shinsaku Fujita 1995/05/27

```

複雑なスタイルファイルもまったく同じことです。clpage.sty を \TeX の標準入力ディレクトリー (普通は、 $\text{\tex}\text{\inputs}$ など) に格納します。

11.1.2 自作のスタイルファイルの使用

通常のオプションスタイルファイルと同じで、 \usepackage で読み込みます。

```

\documentclass[a4j]{jarticle}
\usepackage[clpage]

```

このようにするだけで、簡単に目的の命令や機能が使えるようになります。 $\text{\LaTeX} 2.09$ では、 \documentstyle のオプション引数の中に書き込んでいましたが、 $\text{\LaTeX} 2_{\epsilon}$ では、上記のように変更されていますので、注意が必要です。

11.2 複雑な例

実例として, FPRINT MES 11 (#1687) にアップしたものを載せます. 次の手順で $\text{\LaTeX} 2_{\epsilon}$ 処理をしてください.

%手順

%(1) この書き込み全体を, ファイル名 pageblck.sty として格納してください.

% 例文以下も切り離さずにそのまま入れてください.

%(2) 例文以下を test.tex などの名前にして格納し, latex 処理 (2 回以上) してください.

%%%

% pageblck.sty by Shinsaku Fujita 1995/05/27

% \setFirstPage{102} : 最初の見かけページを設定 (e.g. 102)

% \LastPage: 総ページ数

% \theNetPage: 実ページ/総ページ (e.g. 2/10) のように出力

% \theNetpage: 実ページ (e.g. 2) のように出力

% \thepage: 見かけページ

% \placeblocks: ページ数に応じてブロックを出力

%%%

%%\makeatletter%スタイルファイルとするときはこれをコメントアウト

\@ifundefined{LastPage}{\gdef\LastPage{??}}{}

\newcount\NetPage \NetPage=0

\newcount\FirstPage \FirstPage=1

\def\setFirstPage#1{\setcounter{page}{#1}\global\FirstPage=#1}

\def\writeLastPage{%

\NetPage=\c@page \advance\NetPage-\FirstPage

\iffiles\immediate\write\@auxout{%

\string\gdef\string\LastPage{\the\NetPage}\fi}

\def\theNetPage{\NetPage=\c@page \advance\NetPage-\FirstPage

\advance\NetPage\@ne \the\NetPage/\LastPage}

\def\theNetpage{\NetPage=\c@page \advance\NetPage-\FirstPage

\advance\NetPage\@ne \the\NetPage}

\def\placeblocks{\theNetPage (\thepage) \hskip10pt

\ifcase\NetPage

\or

\or

\or

\or

\or

\or

\or

\or

\or

\or

第12章 ページごとの脚注番号

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

aux ファイルの活用の好例として、脚注番号をページごとリセットする方法を解説します。これに関しては、@nifty (NIFTY-Serve) FSNOTE の会議室 13 で、活発な質疑がありました。4883 番 (「脚注について教えてください」) の書き込みから始まるコメントチェーンは、皆さんがそれぞれのアイデアを出して、いろいろ工夫したマクロが提案した過程が示されていて、たいへんおもしろくよめます。本章は、その質疑を参考にして、書いたものです。

12.1 親子関係の更改—不十分

脚注の番号をページごとに独立に振りたいということがあります。この目的には、footnote カウンターを page カウンターに従属させればよいと考えるのが普通でしょう。まず、これを試してみましよう。

このとき、page カウンターや footnote カウンターはすでに定義済みのものなので、\newcounter 命令は使えません。拙著の mathchem.sty には、\renewcounter 命令を作成してありますが、本質的な部分は \@addtoreset 命令のみです。次の例を試してください。

```
\documentclass[a4j]{jarticle}
\makeatletter
\@addtoreset{footnote}{page}
\makeatother
\section{節番号}
\begin{document}
(本文)\footnote{脚注 A} \par
(本文)\footnote{脚注 B} \par
(本文)\footnote{脚注 C} \par
(本文)\footnote{脚注 D} \par
\clearpage
\section{節番号}
(本文)\footnote{脚注 E}
(本文)\footnote{脚注 F}
(本文)\footnote{脚注 G}
```

¹©藤田 眞作「L^AT_EX 2_ε まくろの八衢」(8a)

```
(本文)\footnote{脚注 H}
\end{document}
```

これでうまくいったような気がします。ところが、これはたまたまなのであって、実は不十分なのです。この場合は、`\clearpage` でページの分割を明示したのでうまくいったのです。実際に、次のようなファイルを作って L^AT_EX 2_ε 処理をしてみますと、

```
\documentclass[a4j]{jarticle}
\makeatletter
\@addtoreset{footnote}{page}
\makeatother
\begin{document}
(本文)\footnote{脚注をたくさん} \par
(これを 2 ページ以上になるように繰り返す)
\end{document}
```

2 ページ目以降で、`footnote` カウンターのリセット異常が起きていることがわかります。

T_EX のページ分割処理では、ファイルを先読みしておいて、適当な分割点をさがして分割するようになっています。このため、ページの終わりに近いところに脚注があった場合に、分割点より前になるか後になるかはページ整形をして始めてわかるわけです。したがって、運の悪いときには、`footnote` カウンターのリセットされないまま次ページ送りになってしまうという事態がおこるのです。ページの先頭で脚注番号が異常になるのはこのためです。

12.2 aux ファイルの活用

このためには、L^AT_EX 2_ε 処理したときに、全部の脚注が実際にどのページに現れているか、`aux` ファイルに書き出して、この情報をもとに 2 回目以降の L^AT_EX 2_ε 処理で、正しい脚注番号を振るようにします。完成したマクロの定義を次に示します。適当に改行して、要所に注釈行を入れました。

```
\documentclass[a4j]{jarticle}
\makeatletter
% Page-Dependent Footnotes by S. Fujita 1995/06/06
\newcount\@fn@total \@fn@total\z@ \newcount\CurrP@ge \CurrP@ge\z@
\def\ftnt@page#1#2{\@tempcnta=#2 \ifnum\@tempcnta>\CurrP@ge
\expandafter\gdef\csname @ftnt@#1\endcsname{#2}\CurrP@ge=\@tempcnta\fi}
\def\Footnote#1{%ページごとの脚注番号
\global\advance\@fn@total\@ne%総脚注番号を 1 だけふやす
\ifundefined{@ftnt@\the\@fn@total}%\@ftnt**が定義されているか
{\footnote{#1}}%定義されていないときは footnote 命令そのまま
{\c@footnote\z@ \footnote{#1}}%定義済みのときは footnote カウンター
%をリセットしたのち、footnote 命令を使う
\edef\the\@fn@total{\the\@fn@total}%総脚注番号を展開後\the.. に設定
{\let\the\z@%これは、\the 命令の一時無効化
\edef\next{\write\@auxout{%出力命令\next を定義
```

```

\string\ftnt@page{\the@fn@total}{\the@c@page}}%出力の内容
\next}}%aux ファイルへ出力: \next 実行しておこなう .
\makeatother
\begin{document}
脚注番号をページごとにリセット\Footnote{ページごとの脚注番号}\par
(たくさん繰り返し)
脚注番号をページごとにリセット\Footnote{ページごとの脚注番号}\par
(たくさん繰り返し)
\end{document}

```

マクロの定義の説明をしましょう。まず、aux ファイルに、`\ftnt@page{F}{P}`の形で出力するようにします。F は脚注番号、P はページ番号です。たとえば、次のように出力されるようにします。

```

\relax
\ftnt@page{1}{1}
\ftnt@page{2}{1}
(途中省略)
\ftnt@page{26}{1}
\ftnt@page{27}{2}
\ftnt@page{28}{2}
(後略)

```

このような aux ファイルへの出力を実現する方法の説明をまず行います。`\Footnote` 命令の中に含まれる `\write` 命令のあとの中括弧のなかの記述がこれにあたることはわかりでしょう。`\let\the\z@` の手法は、 \TeX book 問題 21.10 述べられていることの応用です。この場合は、ページ番号の出力 (`\the@c@page`) をページ整形後の送り出し (`\shipout`) 時まで抑制する目的で使っています。

```
\edef\the@fn@total{\the@fn@total}%総脚注番号を展開後\the.. に設定
```

の定義は、上の `\let\the\z@` による抑制機構が働かないようにするためです (`\next` の定義時に `\the` を展開済みにしてある)。これは、`\Footnote` 命令の定義から、この記述をはずし、`\the@fn@total` のかわりに `\the@fn@total` を用いたもの (`\` が途中にあるかないかに注意)、すなわち、

```

\def\Footnote#1{\global\advance\@fn@total\@ne
\@ifundefined{ftnt@\the@fn@total}{\footnote{#1}}
{\c@footnote\z@ \footnote{#1}}
% \edef\the@fn@total{\the@fn@total}%削除
{\let\the\z@\edef\next{\write\@auxout{\string\ftnt@page
{\the@fn@total}{\the@c@page}}\next}}%下と入れ換え
% {\the@fn@total}{\the@c@page}}\next}}

```

をためしてみればわかります。aux ファイルに打ち出されたものをもとのものと比較してごらん下さい (すこし、aux ファイルの出力の見かけはわるいですが、この定義でも多くの場合うまくゆきます)。以上の説明を大雑把にまとめると、次のようになります。

<code>\the@fn@total</code> の定義時	<code>\the\@fn@total</code> より, 総脚注番号が入る
<code>\next</code> の定義時	<code>\let\the\z@</code> の抑制により, <code>\the\c@page</code> の出力なし. すでに入っている総脚注番号はそのまま
<code>\next</code> の実行時 (送り出し時)	<code>\the\c@page</code> を実行し, ページ出力.

このことを頭に入れて, 次ページ送りになった脚注を考えてみましょう. 次ページ送りになった時点で, 総脚注番号 (脚注の通し番号) は保持されていますが, ページ番号は不定になっています. 後者は, 次の `\next` の実行時に (次ページの送り出し時に) 入れられることとなります.

さて, aux ファイルに打ち出された `\ftnt@page` 命令がどのように働くか, その定義を調べてみましょう. これは簡単で, 二番目の引数 (ページ数) が 1 だけ増えたときに, この時の第一引数を含んだ制御綴を作ります. 上に載せた aux ファイルの出力では, `\ftnt@page{1}{1}` から `\@ftnt@1` (内容は 1) および `\ftnt@page{27}{2}` から `\@ftnt@27` (内容は 2) が作られます. `\ifnum` を使った条件文はそんなにむずかしいことをやっているわけではありませんから, 容易に理解できるとおもいます.

再び, `\Footnote` の定義に戻りましょう.

```
\ifundefined{@ftnt@\the\@fn@total}%\@ftnt**が定義されているか
```

は, `\@ftnt*` が定義されていないときには, 次の中括弧の処理 (ここでは `\footnote` 命令の実行) を行い, 定義されているときは, さらに次の中括弧の処理 (footnote カウンターをリセットしたのち, `\footnote` 命令の実行) を行っています. これで, ページごとの脚注番号の出力が実現できました.

第13章 中間ファイルの新設

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

これまでは、 $\text{\LaTeX} 2_{\epsilon}$ 既存の補助ファイル (.aux ファイル) を活用する方法を述べてきました。この範囲でもかなりのことができることがわかりいただけたとおもいます。さらに、その上級編として、特別な目的をもった中間ファイルの新設すれば、見通しよく解決できる例を示します。今回は、表の下に注 (表注) を便利に入れるための `tblnote` 環境を作ってみましょう。このときに、`.tbn` という拡張子をもった中間ファイルを新設します。`tblnotes.sty` というスタイルファイルに仕立ててありますので、これをオプションに指定することによって、`\TNlabel`、`\TNref`、`\TNtex` などの命令が使えます。

13.1 ソースリスト

まず、オプションスタイルファイル `tblnotes.sty` のソースリストを示します。

```
% tblnotes.sty
% 1995/06/13 by Shinsaku Fujita
%
% \TNlabel{KEY}:      KEY is a reference key
% \TNref{KEY}:       KEY is a reference key
% \TNtext{KEY}{TEXT}: KEY is a reference key. TEXT is the text of
%                   a table note
%
\typeout{Option Style 'tblnotes' 1995/06/13 by Shinsaku Fujita}
% create and open .tbn file
\def\thetblnote{\begingroup\makeatletter
\if@files\newwrite\tf@tblnt\relax
\immediate\openout\tf@tblnt\jobname.tbn\relax\fi
\global\@nobeakfalse\endgroup}
%TNlabel, TNref, TNtext
\newcounter{TableNote}\setcounter{TableNote}{0}
\def\TNlabel#1{\refstepcounter{TableNote}\label{#1}$^{\*\theTableNote}$}
\def\TNref#1{$^{\*\ref{#1}}$}
```

¹©藤田 眞作「 $\text{\LaTeX} 2_{\epsilon}$ まくろの八衢」(8b)

```

\def\TNote#1#2{\@tempcntb=#1\if@filesw\immediate\write\tf@tblnt{%
  \string\tbnttext{\the\@tempcntb}{#2}}\fi}
\def\TNtext#1#2{\@ifundefined{r@#1}%
{\@tempcnta=0}{\edef\@tempa{\@nameuse{r@#1}}%
\edef\@tempb{\expandafter\@car\@tempa \@nil\null}}%
\@tempcnta=\@tempb}\TNote{\@tempcnta}{#2}\ignorespaces}
% \TNoikomi, switch for an oikomi format
\newif\ifTNoikomi \TNoikomifalse
\def\tbnttext#1#2{\ifTNoikomi $\^{*#1}$~#2\hskip1.5em
\else\noindent$\^{*#1}$\hskip1em#2\par\vskip.5ex\fi}
% close .tbn file --- print tablenotes
\def\theTblnote{\if@filesw\immediate\closeout\tf@tblnt\fi
\begingroup\footnotesize
\par\noindent\input{\jobname.tbn}\endgroup}
% \begin{tblnote}... \end{tblnote}
\newdimen\TNdimen
\def\tblnote{\@ifnextchar[{\@tblnote}{\@tblnote[\textwidth]}}
\def\@tblnote[#1]{\TNdimen=#1\thetblnote}
\def\endtblnote{\par\vskiplex
\minipage{\TNdimen}\baselineskip=0.5ex \theTblnote \endminipage}
\endinput
%(Examples)
\documentclass[a4j]{jarticle}
\usepackage{tblnotes}
\begin{document}
(省略)
\end{document}
% the end of tblnotes.sty (1995/06/13 by Shinsaku Fujita)

```

13.2 機能と使い方

`tblnotes.sty` は、オプションスタイルファイルになっていますが、本章では、この箇所の上に、命令の定義を書き込んであります。使えるおもな命令を次に示します。

`\TNlabel{KEY}`: 表注の初出のところに書く。KEY は参照キーをあらわす。

`\TNref{KEY}`: 既出の表注の番号を参照して、その番号を出力する。KEY は対応するものを指定。

`\TNtext{KEY}{TEXT}`: キーは対応する初出のものを入れる。表注の内容を第 2 引数に書き入れる。

これらの命令は、表組全体を

```
\begin{tblnote}[WD] ... \end{tblnote}
```


表 13.1: 表注の入れ方—列挙形式

項目 A						
項目 1	*1	×	*1	*2	×	
項目 2	*2		*3	×	*4	*3
項目 3	*1	*1	×	×	*1	×
項目 B						
項目 1	*5	*5	×	×	*6	×
項目 2	×	*7	×	*7	*7	×

*1 脚注は参照キーによって参照します。脚注の内容はまとめて、ここに示すように列挙します。

*2 脚注 2

*3 脚注 3

*4 脚注 4

*5 脚注 5

*6 脚注 6

*7 脚注 7

この結果，表 13.1 のような出力がえられます。処理の内部では，`\label/\ref` の仕組みを応用していますので，正しい参照番号を得るためには，*L^AT_EX 2_ε* 処理は 2 回以上必要です。

表注の出力を追い込み形式にしたいときは，`\TNoikomittrue` というスイッチを用意してありますので，`\TNoikomittrue` と宣言します。もとの列挙形式に戻すときは，`\TNoikomifalse` とします。

```
\begin{table}
\caption{表注の入れ方---追い込み形式}
\label{tt:12a2}
\begin{center}
\begin{tblnote}[8cm]
\TNoikomittrue
\begin{tabular}{|c|c|c|c|c|c|c|c|}
(省略)
\end{tabular}
\Tntext{Aa1}{脚注は参照キーによって参照します。
脚注の内容はまとめて、ここに示すように列挙します。}
(省略)
\end{tblnote}
\end{center}
\end{table}
```

この結果，表 13.2 のような出力がえられます。表注の番号は，前の表からの続き番号にしてあります。もしも，リセットしたいときは，`\setcounter` 命令を使ってください。

表 13.2: 表注の入れ方—追い込み形式

項目 A						
項目 1	*8	×	*8	*9	×	
項目 2	*9		*10	×	*11	*10
項目 3	*8	*8	×	×	*8	×
項目 B						
項目 1	*12	*12	×	×	*13	×
項目 2	×	*14	×	*14	*14	×

*8 脚注は参照キーによって参照します。脚注の内容はまとめて、ここに示すように列挙します。 *9 脚注 2 *10 脚注 3 *11 脚注 4 *12 脚注 5 *13 脚注 6 *14 脚注 7

13.3 マクロ解説

13.3.1 全体の方針

表注の内容は、中間ファイル (.tbn ファイル) に格納し、出力のときは、これを読み込んでそのまま出力するようにします。相互参照は、`\label` と `\ref` を使うものを処理の内部で使うようにします。

次に上のソースリストから関係の部分を取り出して示します。まず `\newcounter` 命令で、TableNote カウンターを作成します。カウンターの数を増やすのは、`\TNlabel` 命令の中で、`\refstepcounter` を利用しています。`\label` は別に宣言するのではなくて、`\TNlabel` 命令の内部で用いるようにしました。表注番号の出力もこの命令で行います。`\TNref` は、`\ref` の出力を調整したものです。

```
%TNlabel, TNref, TNtext
\newcounter{TableNote}\setcounter{TableNote}{0}
\def\TNlabel#1{\refstepcounter{TableNote}\label{#1}$^{*}\theTableNote{}}
\def\TNref#1{$^{*}\ref{#1}}$}
```

13.3.2 中間ファイルのオープンとクローズ

中間ファイルの名前を `\jobname.tbn` としましょう。`\jobname` は現在処理している文書ファイル (tex ファイル) の名称です。`\newwrite` 命令で、ファイル出力のストリーム番号を新たに宣言します。任意ですが、`\tf@tblnt` とすることにしましょう。この番号を、実ファイルに割り当てて、オープンするのが、`\openout` という命令です。`\immediate` は、すぐさまオープンするということを表しています。次の部分が、このような処理を記述しています。

```
% create and open .tbn file
\def{\begingroup\makeatletter
\if@files%補助ファイル出力のスイッチが true のときは
\newwrite\tf@tblnt\relax%番号\tf@tblnt を用意
\immediate\openout\tf@tblnt\jobname.tbn\relax%これを\jobname.tbn に割当
```

```
\fi
\global\@nobreakfalse\endgroup}
```

書き込みが終ったファイルは、出力する前にクローズします。`\theTblnote` 命令の中にある `\closeout` がその命令です。この部分は、`\openout` と同じ構文ですから、容易にわかります。次に、`tbn` ファイルに書き込まれた内容を出力するために、`\input` 命令で読み込んでいます。

```
% close .tbn file --- print tablenotes
\def\theTblnote{\if@filesw\immediate\closeout\tf@tblnt\fi
\begingroup\footnotesize
\par\noindent\input{\jobname.tbn}\endgroup}
```

13.3.3 中間ファイルへの出力

中間ファイル (`tbn` ファイル) への書き出しは、`\TNote` 命令が受け持っています。内部で、`\write` 命令を使っています。これに続く中括弧の中がその内容です。

```
\def\TNote#1#2{\@tempcntb=#1\if@filesw\immediate\write\tf@tblnt{%
\string\tbnttext{\the\@tempcntb}{#2}}\fi}
```

引数#1 は、TableNote カウンターの値です。引数#2 は、表注の内容が格納されています。このことによって、たとえば、表 13.2 を作ったときの `tbn` ファイルの内容は次のようになります。

```
\tbnttext{8}{脚注は参照キーによって参照します。脚注の内容はまとめて、
ここに示すように列挙します。}
\tbnttext{9}{脚注 2}
\tbnttext{10}{脚注 3}
\tbnttext{11}{脚注 4}
\tbnttext{12}{脚注 5}
\tbnttext{13}{脚注 6}
\tbnttext{14}{脚注 7}
```

この `\TNote` 命令は、`\TNtext` 命令の中で使用されています。この命令は、`aux` ファイルが読み込まれたときに発生する相互参照の制御綴 `r@#1` (引数#1 は相互参照のキー) がすでにあるかどうかを判定し、あればその中に格納されている情報のなかから、TableNote カウンターの値を取り出します。これは、L^AT_EX 2_ε の隠しコマンド `\@car` を利用しています。取り出した TableNote カウンターの値 (文字列) は、`\@tempb` に一時保存しています。この文字列をカウンター `\@tempcnta` に移して数値化しています。このような処理の後、上述の `\TNote` 命令を実行しているわけです。この説明で、次に示す `\TNtext` 命令の定義は容易にたどれるはずですが。

```
\def\TNtext#1#2{\@ifundefined{r@#1}%
{\@tempcnta=0}{\edef\@tempa{\@nameuse{r@#1}}}%
\edef\@tempb{\expandafter\@car\@tempa \@nil\null}}%
\@tempcnta=\@tempb}\TNote{\@tempcnta}{#2}\ignorespaces}
```

`\TNtext` の定義の最後に `\ignorespaces` を宣言してあります。これは、後続の半角の空白を取り除くものです。`\TNtext` 命令をたくさん連ねて、指定してゆきますので、この宣言がないと表注の出力位置が狂ってしまいます。

13.3.4 中間ファイルの読み込み

上述の `\theTblnote` 命令は、`tbn` ファイルを読み込む働きももっています。`\input` 命令の部分がそれです。`tbn` ファイルが読み込まれますと、書き込まれた `\tbnttext` が実行されることとなります。これは、引数 #1 の表注番号と引数 #2 の内容を整形して、実際に出力するための命令です。この際、追い込み形式にするか列挙形式にするかを、`\TNoikomi` スイッチで判断して、処理を分岐させています。

```
% \TNoikomi, switch for an oikomi format
\newif\ifTNoikomi \TNoikomifalse
\def\tbnttext#1#2{\ifTNoikomi $^{\#1}$~#2\hskip1.5em
\else\noindent$^{\#1}$\hskip1em#2\par\vskip.5ex\fi}
```

ここで、`\newif` 命令がでてきましたので、これを説明しておきましょう。`\newif\ifTNoikomi` と書きますと、`\TNoikomi` なるスイッチが作成されます。この実態は、真のときの `\TNoikomitrue` という制御綴、偽のときの `\TNoikomifalse` という制御綴、さらに条件判断のための `\ifTNoikomi` という制御綴が作成されます。上の例では、デフォルトとして、`\TNoikomifalse` を宣言してあるので、普通は列挙形式の表注になるわけです。これまでにてできた `\@filesw` などの $\text{\LaTeX 2}_{\epsilon}$ のスイッチも同じように `\newif` 命令で定義されているわけです。ある条件が真か偽で、処理を分岐させる常法ですので、記憶しておくとう便利です。

13.3.5 環境新設

これを総合して、`tblnote` 環境として整えましょう。この目的は、表注の出力幅を指定できるようにすることにあります。オプション引数として、獲得した寸法 (表注の出力幅) を新しく作成した寸法 `\TNdimen` に格納します。この後、上述の `\thetblnote` が実行されます。

```
% \begin{tblnote}...\end{tblnote}
\newdimen\TNdimen
\def\tblnote{\@ifnextchar[{\@tblnote}{\@tblnote[\textwidth]}}
\def\@tblnote[#1]{\TNdimen=#1\thetblnote}
```

`tblnote` 環境の終りは、`\minipage` 環境を利用して、出力幅を制御しています。内部で、`\theTblnote` を実行しています。`\baselineskip=0.5ex` は行間隔を調整しています。

```
\def\endtblnote{\par\vskip1ex
\minipage{\TNdimen}\baselineskip=0.5ex\theTblnote \endminipage}
```

13.4 その他の方法

出回っているスタイルファイルの中に、`endnotes.sty` があります。これを使えば、表注 (というより後注) が実現できます。このスタイルファイルの `\endnote` 命令そのものは、`\label/\ref` による相互参照ができません。そこで、少し変更して、

```
\documentclass[a4j]{jarticle}
\usepackage{endnotes}
\makeatletter
\def\endnote{\@ifnextchar[{\@xendnote}{\refstepcounter
  {endnote}\xdef\@theenmark{\theendnote}\@endnotemark\@endnotetext}}
\makeatother
\begin{document}...
```

のようにプリアンプルに書いておきます。これによって、`\label/\ref` による相互参照が可能になります。ただ、`$$\ref{foo}$$` のように出力を整えるなど、いくつかの細かい手直しが必要です。

第14章 目次用の中間ファイル

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

目次の項目が出力されるファイルが `toc` ファイルです。普段は、このファイルは発生しませんが、特別に `\tableofcontents` を宣言したときにはじめて発生するものです。今回は、情報の交換という点に注目して、このさまざまな使い方を説明しましょう。

この章は、拙著「化学者・生化学者のための L^AT_EX」の第6章とあわせて読まれると理解が深まると思います。

14.1 `toc` ファイルへの書き出し命令の作成

まず、説明のために、`\tocoutput` 命令を作ります。目次を作成するために、`\tableofcontents` 命令を文書の最初に宣言することはご存じのことと思います。この `\tableofcontents` 命令は、次の二つの働きもっています。

1. 目次項目の `toc` ファイルへの出力
2. 目次の出力

ある文書を分割して、L^AT_EX 2_ε処理をしたいときに、各部分ごとに `\tableofcontents` 命令を宣言すると、目次が出力されるためページ数が狂ってしまいます。手動でやることもできますが、ここでは、上記第一の機能だけを取り出してみよう。

```
\def\tocoutput{\begingroup\makeatletter\if@files\newwrite\tf@toc
\openout\tf@toc\jobname.toc\relax\fi \global\@nbreakfalse \endgroup}
```

このようにすれば、`toc` ファイルへの情報出力だけを行うことが可能になります。長い文書は、分割処理をすることが普通ですので、この命令は意外に便利です。

この命令の成り立ちを説明しておきましょう。とくに、ファイルの新設についての命令が含まれていますので、おぼえておくといでしょう。`\begingroup` と `\endgroup` は、中括弧と同じ働きをします。これらは、中括弧と異なって、別の命令の定義の中に含まれていても構いません。この場合は、同じ命令の中で使っています。これは、`\makeatletter` 命令をこの範囲に制限するために入れています。この命令の中で、

```
\newwrite\tf@toc
\openout\tf@toc\jobname.toc
```

¹©藤田 眞作「L^AT_EX 2_εまくろの八衢」(09)

という記述があります。 `\newwrite` 命令はファイルの新設するための命令です。これによって、新しい出力ストリームの番号 `\tf@toc` が用意されます。このファイルを開くには、`\openout` 命令を使います。上の 2 行目の宣言によって、現在の `tex` ファイル名 (`\jobname` に保存) に `.toc` を拡張子として付けたファイルを開きます。

`\if@filesw` というのは、前にもでてきました。ここで正式に説明しておきましょう。`\@filesw` は、ファイルを出力するかどうかのスイッチです。これは、通常は `\@fileswtrue` の状態にありますが、`\nofiles` が宣言されたときに `\@fileswfalse` の状態になります。`\if@filesw` というのは、この二つの状態のどちらにあるかを判断する条件文です。`\@fileswtrue` の状態にあれば、それに引き続き命令群を実行します。

`\@nobreakfalse` (`\@nobreaktrue` と対) は、`\label` が宣言されたときに、節の変わり目などでページが変わらないようにするためのスイッチですが、今回は余り重要に考えなくともよいでしょう。容易に類推できるように、このスイッチは、`\if@nobreak... \fi` の構文で判断します。

14.2 目次の出力の仕組み

14.2.1 aux ファイルへの書き出し

L^AT_EX 2_ε 処理の際に、目次の項目は `toc` ファイルにかき出された情報によるのですが、それまでにいくつかの手順を踏んでいます。これを理解することが本節の目的です。

目次に必要な項目は、まず補助 (`aux`) ファイルへの書き出されます。これは、実は `\section` などの章立て命令の働きによります。この段階では、`\tableofcontents` 命令 (あるいは前節の `\tocoutput` 命令) は、必ずしも宣言されていなくともよいのです。

`\section` 命令が、目次に関する項目を `aux` ファイルにかき出すメカニズムはかなりむずかしいのですが、L^AT_EX 2_ε の特徴がよくあらわれています。`art10.sty` ファイルなどに `\section` 命令が定義されていますが、この段階を調べても、この命令の `aux` ファイルへの書き出し機能について、ソースリストの中には直接に手がかりになるものはありません。例として、`\section` 命令の定義を `art10.sty` ファイルの中から抜き出します。

```
\def\section{\@startsection {section}{1}{\z@}{-3.5ex plus -1ex minus
-.2ex}{2.3ex plus .2ex}{\Large\bf}}
```

したがって、内部命令 `\@startsection` の機能をさらに調べなければなりません。この命令はいろいろな機能をもっていますが、その中で、目次の打ち出しの機能に限って追跡することにしましょう。ファイルを遡って追跡してゆくと、この `\@startsection` 命令は、`\tex\macros` ディレクトリーの `latex.tex` の中で定義されています。さらに、`\@startsection` の内部命令として、下請け命令 `\@sect` があります。この `\@sect` の中で、`\addcontentsline` を使っています。この `\addcontentsline` 定義の中に、`aux` ファイルへの書き出しを担当している部分が含まれています。

L^AT_EX 2_ε を起動すると (`\tableofcontents` の宣言の有無にかかわらず)、`aux` ファイル `jobname.aux` が作られます。その中には、他の情報とともに、目次に関するものが書き出されています。たとえば、

```
(jobname.aux の内容の一部)
\relax
\@writefile{toc}{\string\contentsline\space
{section}{\string\numberline\space {40}目次}{1}}
```

```
\@writefile{toc}{\string\contentsline\space
{subsection}{\string\numberline\space {40.1}目次の出力}{1}}
\@writefile{toc}{\string\contentsline\space
{subsection}{\string\numberline\space {40.2}目次の出力の仕組み}{1}}
```

14.2.2 aux ファイルへから toc ファイルへ

この段階で、文書中の目次を出力すべき箇所に、

```
\tableofcontents
```

と宣言します。この命令の機能の一つは、上述のように、aux ファイルへから toc ファイルへの情報変換を行うことです。`\tableofcontents` 命令の定義は、スタイルファイルの中で行われているので覗いてみましょう。

```
\def\tableofcontents{\section*{Contents\@mkboth{CONTENTS}{CONTENTS}}
\@starttoc{toc}}
```

ただし、日本語の場合 “Contents” のところは「目次」となります。この中の`\@starttoc{toc}`は、拡張子として.tocをもつファイルが存在すればそれを使い、無ければ、新たに toc ファイルを作る機能をもちます。実は、この機能を取りだして、前節で`\tocoutput`を作成したのです。

L^AT_EX 2_ε処理の最後に、aux ファイルが読み込まれます。(これは、`\enddocument` 命令の中で行われます。L^AT_EX 2_ε処理の際に、都合 2 回 aux ファイルが読み込まれます)。このとき、上記の`\@writefile` 命令が実行されます。それによって、toc ファイルへの書き込みが行われるわけです。

`\@writefile` による書き込みの際に、`\string` が前に付いた制御綴は、実行されずに、そのまま文字列として扱われることはすでに述べたとおりです。したがって、`\@writefile` 命令によって、toc ファイルに書き込まれたものは、`\contentsline` と `\numberline` になっています。一方、`\string` が付いていない `\space` 命令は実行されて、半角の空白を生じます。上記の aux ファイルから生じた toc ファイルの内容を次に示します。

(jobname.toc の内容)

```
\contentsline {section}{\numberline {40}目次}{1}
\contentsline {subsection}{\numberline {40.1}目次の出力}{1}
\contentsline {subsection}{\numberline {40.2}目次の出力の仕組み}{2}
```

`\tableofcontents` 命令を宣言した後の最初の L^AT_EX 2_ε処理においては、題字「目次」(“Contents”)が出力されているだけです。toc ファイルが存在しないので、目次の内容は出力されません。しかし、L^AT_EX 2_ε処理の内部では、上記のような toc ファイルが作られているのです。

14.2.3 toc ファイルの内容を出力

再度、L^AT_EX 2_εを起動すると、今度は、toc ファイルが存在するので、これが読み込まれます。この読み込みは、`\tableofcontents` 命令のもう一つの機能ですが、実際には、さらに下位の内部命令`\@starttoc{toc}`

が分担しています。このことにより、`toc` ファイルに書き込まれている `\contentsline` 命令が実行されま
す。これは、実際に目次を出力する命令です。

目次

40	目次	1
40.1	目次の出力	1
40.2	目次の出力の仕組み	2

`\contentsline{SEC}`命令は、*L^AT_EX 2_ε*の内部では、`\l@SEC`命令と同義です。SEC は章建ての命令を表
しています。たとえば、`\contentsline{section}`命令は、`\l@section`命令と同義です。`\l@SEC`命令は、
スタイルファイル中で定義されています。目次の体裁を変更したい場合には、これらの命令を再定義すれば
よいわけです。

第15章 目次項目の追加

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

15.1 引数に関する注意事項

LaTeX 2_εでは、相互参照を行っているため、これに関わる制御綴 (コントロールシーケンス) の引数の中にさらに制御綴が含まれると、面倒が起こることがあります。これを防ぐために、`\protect` 命令を使います。この`\protect` 命令が何のためにあるのか、なかなか理解できなくて悩ましいものです。ここでは、この命令が何をしているのかを理解するために、実験を行ってみましょう。

ソースリストには、`\chapter` 命令の引数として「`\TeX{}` および `\TeX/`」と書いて、`\protect` 命令の有無によって、結果がどう違うかを追跡してみます。次に示す `test.tex` ファイルを作って、それから発生する補助ファイルを調べましょう。`tex` ファイル ⇒ `aux` ファイル ⇒ `toc` ファイルの順で情報が移動するのは、上で示したとおりです。生じたファイル内容を次に示します。ただし、見やすいように適当に改行してあります。このときの `dvi` ファイルをプレビューすると、章の見出しは「`\TeX` および `\TeX`」となっています。

```
% *****
% *(test.tex の内容)*
% *****
\documentclass{jbook}
\begin{document}
\setcounter{chapter}{5}
\addtocounter{page}{50}
\tableofcontents
%%%\protectの有無%%%\protect
\chapter{\TeX{ }および\protect\TeX{ }}
%%%\人為的な追加%%%\protect
\addcontentsline{toc}{section}{\protect\numberline{6.1}{\protect\LaTeXe}}
\section*{EXERCISES} %*を付けると目次に出力されない。
\addtocontents{toc}{\protect\medskip} %目次に垂直方向の空白を追加。
\addcontentsline{toc}{section}{Excercises}%番号を付けずに目次に追加。
\end{document}

% *****
% *(test.aux の内容)*
% *****
```

¹©藤田 眞作「LaTeX 2_εまぐろの八衢」(10)

```

\relax
\@writefile{toc}{\string\contentsline\space
{chapter}{\string\numberline\space {6}
T\kern -.1667em\lower .5ex\hbox {E}\kern -.125emX{}}および
\string\TeX\space {}}{53}}
\@writefile{lof}{\string\addvspace\space {10pt}}
\@writefile{lot}{\string\addvspace\space {10pt}}
\@writefile{toc}{\string\contentsline\space
{section}{\string\numberline\space {6.1}{\string\LaTeXe\space }}{53}}
\@writefile{toc}{\string\medskip\space }
\@writefile{toc}{\string\contentsline\space {section}{Exercices}{53}}

% *****
% *(test.toc の内容)*
% *****
\contentsline {chapter}{\numberline {6}
T\kern -.1667em\lower .5ex\hbox {E}\kern -.125emX{}}および\TeX {}}{53}
\contentsline {section}{\numberline {6.1}{\LaTeXe }}{53}
\medskip
\contentsline {section}{Exercices}{53}

```

このファイルの変換の中で、二つの \TeX がどういう変化を辿っているのかを抜き出したのが、次の表です。

tex	$\backslash\text{\TeX}\{}$	$\backslash\protect\text{\TeX}\{}$
↓	↓	↓
aux	T\kern -.1667em\lower .5ex\hbox {E}\kern -.125emX{}	$\backslash\string\text{\TeX}\space \{$
↓	↓	↓
toc	T\kern -.1667em\lower .5ex\hbox {E}\kern -.125emX{}	$\backslash\text{\TeX} \{$

ちなみに、 \TeX のロゴを作る命令は、

```
\def\TeX{T\kern -.1667em\lower .5ex\hbox {E}\kern -.125emX}
```

となっています。したがって、 $\backslash\protect$ 命令を付けない $\backslash\text{\TeX}\{}$ は、もとの定義に展開されて、toc ファイルに書き込まれていることとなります。一方、 $\backslash\protect\text{\TeX}\{}$ は、 $\backslash\text{\TeX}\{}$ を文字列として扱って、そのままの形で、toc ファイルに出力していることになっているわけです。ここで、 $\backslash\string$ 命令は、そのままの形で出力する機能をもつ制御綴です。

$\backslash\text{\TeX}\{}$ という命令は展開されても、たまたま $\backslash\@writefile$ 命令がうまく働きます。しかし、引数の中に含まれるマクロ命令を展開したときに $\backslash\@writefile$ 命令がうまく働かない場合には、エラーになってしまいます。したがって、このような展開を抑制する目的で $\backslash\protect$ 命令を使うわけです。

試みに、toc ファイルに書き込まれた内容を次にコピーして、そのまま $\text{\LaTeX} 2_{\epsilon}$ 処理にかけてみましょう。次に示すように、目次の中の該当行が出力されます。

6	T_EX および T_EX	53
6.1	L ^A T _E X 2 _ε	53
	Excercises	53

15.2 目次への項目追加

上記の tex ファイルの中には、目次の中に項目を人為的に追加するための命令として、`\addcontentsline` と `\addtocontents` の使用例も示しておきました。

```
\addtocontents{toc}{\protect\medskip} %目次に垂直方向の空白を追加。
```

なる命令において、第 1 引数 `toc` は、出力すべきファイルが `toc` ファイルであることを示します。第 2 引数が出力すべき内容です。この場合は `\protect` 命令を付けた `\medskip` を書いているので、`toc` ファイルに `\medskip` という文字列が出力されるわけです。このため、2 回目以降の L^AT_EX 2_ε 処理にかけると、目次の中に `\medskip` 命令による垂直方向の空白が取られます。

```
\addcontentsline{toc}{section}{Excercises}%番号を付けずに目次に追加。
```

という命令は、`toc` ファイル (第 1 引数) に、`section` (第 2 引数) の項目として、`Excercises` (第 3 引数) なる文字列を書き込めということを表しています。この結果、2 回目以降の L^AT_EX 2_ε 処理にかけることによって、目次の中に “Excercise” という項目が節に相当する箇所に追加されます。この命令は、上記の例のように、`\section*` 命令と組合せて使うと便利です。ここでは、章末に練習問題を付けて、そこには節番号を付けないが、目次には追加しておきたいという場合などに有効に応用できます。これらの命令の機能についても、その結果を tex ファイル ⇒ aux ファイル ⇒ toc ファイルの順に追跡してゆけば、上の説明からの類推で理解できるでしょう。

15.3 別立ての目次

注意が必要なのは、`\tableofcontents` 命令を宣言して目次を出力すると、ページ数が狂ってしまうことです。小さな文書のときは、この変化が目次に反映されるまで、L^AT_EX 2_ε 処理を繰り返せば、たいした手間にはならないでしょう。

しかし、書籍の場合などの場合には、この処理の手間は馬鹿にならないし、また、索引などにも影響するので、目次を別立てにしておくことが普通に行われます。`\tableofcontents` と本文先頭との間に (`\clearpage`) `\setcounter{page}{0}` などと宣言しておくのが一つの手です。しかし、もう少しよい方法はないものでしょうか。

もっと簡単な方法として、新設した `\tocoutput` 命令が使えます。この命令は、`toc` ファイルを作ってその中に目次の内容を出力するが、本文中には印刷しないというものです。これは、拙著の中で、オプションファイル `toc_out` としてあります。これを用いるには、

```
\documentclass[12pt,a4j]{jarticle}
\usepackage{toc_out}
```

のように、オプションで `toc_out.sty` を指定し、さらに、プリアンブルに `\tocoutput` を宣言しておきます。この結果、L^AT_EX 2_ε 処理によって、必要な `toc` ファイルが発生するようになります。

もう一つの方法として、`lablst.tex` というユーティリティを使うものがあります。これは、 \LaTeX 2 ϵ に大抵用意されている便利なユーティリティです。このユーティリティは、`aux` ファイル (`jobname.aux`) の内容のうち、目次と引用文献の部分を選んで、`dvi` ファイル (`lablst.dvi`) に変換するという機能をもっています。この際、同時に `lablst.lab` というファイルが出力されます。これは、相互参照に関わるデータを集めたものになっています。`labst.lab` ファイルは、テキストファイルなので、適当なエディターで編集できます。したがって、目次に必要な項目だけを残せばよいわけです。

`lablst` ユーティリティを使って別立ての目次を作るための具体的な手順を示しましょう。

1. まず、`jobname1.tex` という文書ファイルを作っているとします。これを \LaTeX 2 ϵ 処理することにより、`jobname1.aux` という補助ファイルを得ます。
 2. コマンドラインに `jlatex lablst` と打ち込んで、ユーティリティ `lablst` を起動します。
 - まず、`\filename=` と `aux` ファイルの名前を聞いてくるので、`jobname1` と打ち込みます。
 - 次に、`\docsty=` と文書スタイルを聞いてくるので、`jarticle` と打ち込みます (もちろん、他のスタイルを使っているときは、それに応じて名前を入力します)。
- このようにすると、処理が行われて、`lablst.dvi` ファイルが生じます。必要ならば、コマンドラインに `dvi2 lablst` と打ち込んで、プレビューで出力を見てもよいでしょう。
3. 同時に `lablst.lab` というファイルが発生しているので、これを `jobname2.toc` にコピーします。
 4. `jobname2.toc` ファイルをエディターで開いて、先頭に `\contentsline` 命令が宣言されている行だけを残します。これで、 \LaTeX 2 ϵ 処理で得られる `toc` ファイルに相当するものが得られることになります。
 5. 別立ての部分を作る文書ファイルとして、`jobname2.tex` を作ります。ファイル出力を抑制するため、この中のプレアンブル `\nofiles` の宣言をいれ、さらに、目次を出力する箇所に `\tableofcontents` を宣言します。
 6. `jobname2.tex` を \LaTeX 処理にかけます。

第16章 目次の体裁の変更

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

目次では、題目とページ番号の間にリーダーをおくことが多いのは、手元にある書籍をみれば容易に確かめられます。jbook スタイルなどは、book スタイルなどの英文のスタイルをもとにしているので、目次用のリーダーも、もとの英文用のままになっています。注意してご覧になると気付くことですが、 $\text{\LaTeX} 2_{\epsilon}$ で書いたと思われる書籍は、大抵は英文標準になっています。拙著「化学者・生化学者のための \LaTeX 」を書いたときに、和書によく使われている三点リーダーを連ねたものを使ってみました。たしかに、このようにすると和書らしい雰囲気がでできます。ここでは、 \TeX におけるリーダーの出力の仕方の説明をかねて、三点リーダーを連ねた目次リーダーを作ってみましょう。

16.1 指定できるパラメーター

jarticle.sty などの中の `\def\@dotsep{4.5}` が、目次のリーダーの間隔を変えるためのパラメーターです。この標準設定では、 4.5mu の二倍の間隔のリーダーが出力されます。

1	章のタイトル	3
1.1	節のタイトル	3
1.2	節のタイトル	4

この設定を 1.5mu に変更してみます。ついでに、`\def\@pnumwidth{1.55em}` で初期設定されているページ番号を出す箇所の幅も、標準のから、 1zw に狭めてみます。

```
\makeatletter \def\@dotsep{1.5} \def\@pnumwidth{1zw} \makeatother
```

この結果は、次のようになります。

1	章のタイトル	3
1.1	節のタイトル	3
1.2	節のタイトル	4

このようにすると、点の間が詰まって、やや和書らしくなるのですが、文字の底の線に沿って並ぶため、かえって見苦しく感じられるようになります。これを文字の高さの半分にもってくるのは、単にパラメーターを変えるだけでは、うまくゆきません。

¹©藤田 眞作「 $\text{\LaTeX} 2_{\epsilon}$ まぐろの八衢」(10a)

16.3 和書用の目次リーダー

LaTeX 2_ε の目次のリーダーの出力は、`\@dottedtocline` 中の次のような記述に基づいています。これと、三点リーダーを使ったものを比較します。

```
\Line{%
初\leaders\hbox{$\m@th \mkern \@dotsep mu.\mkern \@dotsep mu$}\hfill 終}
\Line{初\leaders\hbox to0.33333zw{\hss\raise.4zh\hbox{.}\hss}\hfill 終}
```

`\@dotsep` を標準の 4.5 に再設定して、この定義によるリーダーを出力してみましょう。次のようになります。

```
初 . . . . . 終
初 ..... 終
```

さて、いよいよ `\@dottedtocline` の再定義です。`\leaders` 命令を上第 1 のものから第 2 のものに入れ換えるだけです。その他の箇所は変える必要はありません。

```
\makeatletter
\def\@dottedtocline#1#2#3#4#5{\ifnum #1>\c@tocdepth \else
  \vskip \z@ plus .2pt
  {\leftskip #2\relax \rightskip \@tocrmarg \parfillskip -\rightskip
   \parindent #2\relax\@afterindenttrue
   \interlinepenalty\@M
   \leavevmode
   \@tempdima #3\relax \advance\leftskip \@tempdima \hbox{ }\hskip -\leftskip
   #4\nobreak\leaders\hbox to0.33333zw{\hss\raise.4zh\hbox{.}\hss}\hfill
   \nobreak \hbox to\@pnumwidth{\hfil\rm #5}\par}\fi}
\makeatother
```

理解を助けるために、ここで、`\@dottedtocline` の各引数の説明をしておきましょう。`latex.tex` から、引用しますと、

```
\@dottedtocline{LEVEL}{INDENT}{NUMWIDTH}{TITLE}{PAGE}
```

で与えられ、引数の意味は次の通りです。

- LEVEL: LEVEL > `\c@tocdepth` ならば、リーダーを出力せず。たとえば、`jbook` などでは、章以上は、リーダーを出力せず。
- INDENT: 左マージンからの字下げ
- NUMWIDTH: `\numberline` 命令があるときの、章番号などの出力幅
- TITLE: タイトル
- PAGE: ページ番号

ついでに、`\def\@pnumwidth{1.55em}` で初期設定されているページ番号を出す箇所の幅も、標準のから、`1zw` に狭めてみます。`\numberline` も `3zw` 幅になるように設定します。このように、再定義をしますと、次のようなレイアウトになります。

1	章のタイトル	3
1.1	節のタイトル	3
1.2	節のタイトル	4

第17章 章立て命令の種々相

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

拙著第6章で、章立ての仕様変更をいくつかのべてあります。`\section` 命令などの内部で使われている `\@startsection` 命令の6個の引数の意味についても解説しました。これらの引数を変えることによって、かなり多彩な仕様を実現できることも述べました。今回は、泥縄ではあるが、役に立つ修正法を解説することにします。

17.1 浅いレベルでできること

`\@startsection` 命令の6番目の引数に目くらしの制御綴を忍ばすことができます。ここでは、このような特殊な手法を述べてみましょう。

article などでは、節番号は「6 タイトル」、項番号は「6.1 タイトル」のように数字の後にはピリオドがありません。ところが、節番号を「6. タイトル」、項番号を「6.1. タイトル」のようにピリオドをつける形式を要求される論文誌がかなりあります。次の定義はこれを実現するものです。拙著の付録のフロッピーディスクに含まれる `jacs.sty` から該当の部分を引用します。

```
\documentclass[a4j]{jarticle}
\makeatletter
%1992 July 8 S. Fujita%%Section Numbering 1 ==> 1. %%%
\def\section{\@startsection {section}{1}{\z@}{-3.5ex plus -1ex minus
-.2ex}{2.3ex plus .2ex}{\edef\@svsec{\thesection.\hskip 1em}\Large\bf}}
\def\subsection{\@startsection{subsection}{2}{\z@}{-3.25ex plus -1ex minus
-.2ex}{1.5ex plus .2ex}%
{\edef\@svsec{\thesubsection.\hskip 1em}\large\bf}}
\makeatother
\begin{document}
\section{ピリオドつきの節番号}
(本文)
\end{document}
```

この例のようにプリアンプルに宣言すると、目的が達せられます。

¹©藤田 眞作「L^AT_EX 2_εまくろの八衢」(11)

このマクロのポイントは、`\@startsection` 命令の最後の引数の中に、次のような `\@svsec` 命令の再定義をもぐりこませたところにあります。

```
\edef\@svsec{\thesecion.\hskip 1em}
```

これで、目的が達せられることは、`\@startsection` 命令の定義 (`latex.tex` 中) をよく読んでみるとわかります。次節の説明を読んでから、考えると一層理解が深まると思います。

要するに、番号の出力様式を規定している `\@svsec` を再定義すればよいのですから、いろいろなことができます。次のようにプリアンプルに宣言してみましよう。

```
\makeatletter
%1992 May 27 S. Fujita%%Section Numbering 1 ==> 1. %%
\def\section{\@startsection {section}{1}{\z@}{-3.5ex plus -1ex minus
-.2ex}{2.3ex plus .2ex}{\edef\@svsec{\fbox{\thesecion}\hskip 1em}\Large\bf}}
\def\subsection{\@startsection{subsection}{2}{\z@}{-3.25ex plus -1ex minus
-.2ex}{1.5ex plus .2ex}%
{\edef\@svsec{\fbox{\thesubsection}\hskip 1em}\large\bf}}
\makeatother
```

すると、節番号「**6** タイトル」、項番号「**6.1** タイトル」のように枠で囲むこともできます。ちょっと悪ふざけですが、つぎのようにプリアンプルに宣言してみましよう。

```
\makeatletter
%1992 July 8 S. Fujita%%Section Numbering 1 ==> 1. %%
\def\section{\@startsection {section}{1}{\z@}{-3.5ex plus -1ex minus
-.2ex}{2.3ex plus .2ex}{%
\edef\@svsec{\$spadesuit$\,\thesecion\,$spadesuit$\hskip 1em}
\Large\bf}}
\def\subsection{\@startsection{subsection}{2}{\z@}{-3.25ex plus -1ex minus
-.2ex}{1.5ex plus .2ex}{%
\edef\@svsec{\$heartsuit$\,\thesubsection\,$heartsuit$\hskip 1em}\large\bf}}
\makeatother
```

すると、節番号「**♠6♠** タイトル」、項番号「**♡6.1♡** タイトル」のように出力することもできます。これは、さすがに悪趣味のような気がします。

すこし、まじめなものに戻して、この節を終わることにしましよう。

```
\makeatletter
%1995 May 27 S. Fujita%%Section Numbering 1 ==> 1. %%
\def\section{\@startsection {section}{1}{\z@}{-3.5ex plus -1ex minus
-.2ex}{2.3ex plus .2ex}{\edef\@svsec{\$S$\thesecion.\hskip 1em}\Large\bf}}
\def\subsection{\@startsection{subsection}{2}{\z@}{-3.25ex plus -1ex minus
-.2ex}{1.5ex plus .2ex}%
{\edef\@svsec{\$S$\thesubsection.\hskip 1em}\large\bf}}
\makeatother
```

すると、節番号「**§6. タイトル**」、項番号「**§6.1. タイトル**」のようになります。これは、和書にときどきみられる形式です。

17.2 内部マクロ細見

「6. SECTION NUMBER」, 「6.1. SUBSECTION NUMBER」のように節番号, 項番号にピリオドをつけ, タイトルが大文字になるという形式を要求する論文誌もあります. `\section{Section number}`と入力しても, このように出力するには, もう少し深いレベルの内部マクロを再定義する必要があります. この機能の実現のためには, `@\sect`(通常処理)と`@\sssect`(スターの付いたときの処理)を修正します. これらは, `\startsection` 命令の内部命令です. まず完成品を示しましょう.

```
\documentclass{jarticle}
\usepackage{a4j}
\makeatletter
\def\@sect#1#2#3#4#5#6[#7]#8{%番号出力あり
\ifnum #2>\c@secnumdepth\def\@svsec{}\else%番号出力するか否かの条件
\refstepcounter{#1}%カウンター値を1ふやす
\edef\@svsec{\csname the#1\endcsname.\hskip 1em }%番号出力形式
\fi
\@tempskipa #5\relax%空白をスキップに設定(正負の判断のため)
\ifdim \@tempskipa>\z@ %スキップが正のとき, 見出しで改行
\begingroup #6\relax%書体の指定がここに入る
\@hangfrom{\hskip #3\relax\@svsec}%番号出力
{\interlinepenalty \@M \uppercase{#8}\par}%タイトルを大文字に
\endgroup%
%(注意) オプション引数#7がないときは#8を#7に複写
\csname #1mark\endcsname{#7}%柱への出力内容
\addcontentsline{toc}{#1}{%toc ファイルへの出力
\ifnum #2>\c@secnumdepth
\else\protect\numberline{\csname the#1\endcsname}\fi
#7}}%
\else%スキップが負のとき, 見出しのあとの改行なし
\def\@svsechd{#6\hskip #3\@svsec \uppercase{#8}
\csname #1mark\endcsname{#7}
\addcontentsline{toc}{#1}{%toc ファイルへの出力
\ifnum #2>\c@secnumdepth
\else\protect\numberline{\csname the#1\endcsname}\fi
#7}}%
\fi
\@xsect{#5}%見出しのあとの段落処理など
%%%スターのついたとき
\def\@sssect#1#2#3#4#5{%番号出力なし
\@tempskipa #3\relax%空白をスキップに設定(正負の判断のため)
\ifdim \@tempskipa>\z@%スキップが正のとき, 見出しで改行
\begingroup #4
\@hangfrom{\hskip #1}{\interlinepenalty \@M \uppercase{#5}\par}
```

```

\endgroup
\else%スキップが負のとき, 見出しのあとの改行なし
\def\@svsechd{#4\hskip #1\relax \uppercase{#5}}\fi
\@xsect{#3}}%見出しのあとの段落処理など
\makeatother
%(例文)
\begin{document}
\section{ピリオドつきの節番号} (本文)
\subsection{Subsection number} (本文)
\section{Section number} (本文)
\section*{Section number} (本文)
\end{document}

```

もとの`\@sect`, `\@ssect` の定義 (`latex.tex`) と比べてみると, 修正するといってもわずかな部分です. 上記のソースリストでは, 改行を工夫して, もとの定義よりも処理がたどり易いようにしました. また, 要所には, 各処理の説明を注釈行として入れてあります. ペナルティーなどわからない部分もあるかも知れませんが, あまり気にする必要はありません. 引数などの対応を考えてゆくと, どこを直せばよいかは自ずからわかってきます. `\@tempskipa` は, 一時保存に使うスキップ (寸法に伸び縮みの量を付加したもの) を示しています.

まず, 引数の対応を頭に入れてください. `\@sect` の引数 #1 から #6 は, `\@startsection` 命令の 6 個の引数を引き継いでいます. また, #7 と #8 は, それぞれ `\section` などのオプション引数と通常の引数を引き継いでいます. 一方, `\@ssect` の引数 #1 から #4 は, `\@startsection` 命令の 6 個の引数のうち, #3 から #5 を引き継いでいます. また, `\@ssect` の引数 #5 は, `\section*` などの通常の引数を引き継いでいます.

さて, 変更した部分の説明を始めましょう. まず, 番号にピリオドを打つところは,

```
\edef\@svsec{\csname the#1\endcsname.\hskip 1em }%番号出力形式
```

で行っています. `\csname... \endcsname` は, 挟まれた文字列の先頭に `\` を付けた制御綴 (`control sequence`) を作るためのものです. このためは, 引数 #1 の文字列が `section` の場合は, 結局 `\thesection` という制御綴を作成していることになります.

大文字出力は, タイトルが文字列 #8 に格納されていますから, これを `\uppercase{#8}` のようにすればよいことがわかります.

`\addcontentsline` は, すでに説明しました. `toc` ファイル (第 1 引数) に, #1 (第 2 引数, これは `section` などの文字列) の項目として, 第 3 引数 (すなわち, 条件文の条件に合致したときのみ,

```
\protect\numberline{\csname the#1\endcsname}
```

および引数 #7 (#8 の複写)) を出力するというものです. #1 が `section` のときは, #7 (#8 の複写) が `Section number` のときは,

```
\contentline {section}{\numberline{9} Section number}{12}
```

のように, `toc` ファイルに出力されますが, 下線で示したところが, 対応する部分です. 仮に `\thesection` の値を 9 とし, 12 ページとしてあります.

スターの付いた命令`\section*`の場合 (`\@ssect` の修正) も, 同様ですので, 読者みずから処理の過程をたどってみてください.

第18章 罫線を利用した数学・化学記号

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

罫線を使う機能を応用して、既存の記号からちょっとした変更を行うことができます。この章では、簡単な例を紹介します。罫線については、拙著第12.1節に詳しい説明があります。参照されれば、本章で述べた以外にもいろいろな方法があることに気付くでしょう。

18.1 標準をあらわす添え字

化学熱力学の慣行として、ある化学過程の標準状態を表すのに「 \ominus 」という記号を用います。たとえば、イアン・ミルズら(朽津耕三訳)「物理化学で用いられる量・単位・記号」講談社サイエンティフィク,(1991) p. 49に、この慣行の説明があります。この記号を、罫線を使って作ってみましょう`\soms`という命令の定義は次の通りです。

```
\def\soms{\ooalign{\hfil\hbox{$\circ$}\hfil\cr\cr%
\raise.55ex\hbox to.5em{\hrulefill}}}}
```

この中で使っている`\ooalign`は二つの文字を重ね打ちするために、 $\text{T}_\text{E}\text{X}$ で用意されている命令です。応用範囲が広いですから、おぼえておくとよいでしょう。`\cr\cr`命令の前後にあるのが、重ね打ちする文字です。`$$\circ$`は「 \circ 」を出力します。その前後に`\hfil`命令がありますが、これで中央揃えを実現しています。罫線は、`\hbox to.5em{\hrulefill}`という命令で作っています。実際にここに宣言すると「`_`」と出力されます。これは、幅`0.5em`の横箱(`\hbox`)を作って、その中に横罫を幅いっぱいに出力(`\hrulefill`)するということを実行しています。この罫線は下過ぎますから、`\raise`命令を用いて、`0.55ex`だけ上方に移動してから出力しています。

ついでに、標準モルギブス関数`\sGibbs`と標準モルエントロピー`\sEntropy`も定義しておきます。

```
\def\sGibbs#1{\mbox{$\Delta G_{\rm #1}^{\soms}$}}
\def\sEntropy#1{\mbox{$\Delta S_{\rm #1}^{\soms}$}}
```

例文を示しておきましょう。

(例文) 標準モル生成ギブス関数を、`\sGibbs{f}`と書くと約束する。このとき、

```
\begin{equation} \rm
CO(g) + 1/2O_2(g) \rightarrow CO_2(g)
```

¹©藤田 眞作「 $\text{L}^{\text{A}}\text{T}_\text{E}\text{X} 2_\epsilon$ まくろの八衢」(12)

`\end{equation}`

のような反応の標準モルギブス関数は、つぎのようにしてえられる。

`\begin{eqnarray}`

`\sGibbs{m} & = & \rm`

`\sGibbs{f}(CO_{2}, g) - \sGibbs{f}(CO, g) - 1/2\sGibbs{f}(O_{2}, g) \\`

`& = & \rm \{-394.38-(-137.27)-0\} \quad \text{kJ mol}^{-1} \\`

`& = & \rm -257.11 \quad \text{kJ mol}^{-1}`

`\end{eqnarray}`

出力は、次の通りです。

(例文) 標準モル生成ギブス関数を、 ΔG_f^\ominus と書くと約束する。このとき、



のような反応の標準モルギブス関数は、つぎのようにしてえられる。

$$\Delta G_m^\ominus = \Delta G_f^\ominus(\text{CO}_2, \text{g}) - \Delta G_f^\ominus(\text{CO}, \text{g}) - 1/2\Delta G_f^\ominus(\text{O}_2, \text{g}) \quad (18.2)$$

$$= \{-394.38 - (-137.27) - 0\} \text{ kJmol}^{-1} \quad (18.3)$$

$$= -257.11 \text{ kJmol}^{-1} \quad (18.4)$$

なお、`\oalign`の使い方は、拙著「化学者・生化学者のためのL^AT_EX」の98, 224ページに例がありますので、参考してください。また、`equation`環境で、化学式を書くための工夫は、同第9章「化学環境」で詳しく述べました。

18.2 不等号

以上 (\geq)、以下 (\leq) を表す記号は、L^AT_EX 2_εでは、`\geq`、`\leq`を使います。これが、数学・化学の分野で国際的に通用している記号ですが、日本では、下の等号が2本棒のものを使う慣行があります。L^AT_EX 2_ε標準では、用意されていないので、これを作ってみましょう。

もっとも簡単には、plain T_EXの中にある`\cong` (\cong) の定義を応用します。この定義は、T_EXbookのAppendix Bに記載されているように、`\sim` (\sim) と等号 (=) を合成して作っています。その部分を引用しますと、

`\def\cong{\mathrel{\mathpalette\@vereq\sim}} \quad \text{\sim over =}`

となっています。`\@vereq`が=の上に記号をのせるための制御綴です。この定義もT_EXbookの同じ箇所に載っていますので、興味のある方はご覧ください。したがって、

`\makeatletter`

`\def\Geq{\mathrel{\mathpalette\@vereq>}} \quad \text{> over =}`

`\def\Leq{\mathrel{\mathpalette\@vereq<}} \quad \text{< over =}`

`\makeatother`

とすれば、よいことになります。実際に`\$a\Geq b\$`と`\$c\Leq d\$`書きますと、 $a \geq b$ と $c \leq d$ のようになります。等号の二つの横棒の間隔が広すぎるような気がします。`\mathrel`と`\mathpalette`については、のちほど説明します。

この点を改良してみましょう。作成したマクロを示しておきましょう。

% \Geq () と \Leq () の定義

```
\makeatletter
\def\GLEq@#1#2{\mathrel{\lower.2ex\hbox{\vbox{\hbox{#1#2$\kern-.1em}$
\kern.2ex\hrule\kern.2ex\hrule}}\kern.1em}}
\def\Geq{\mathchoice{\GLEq@{>}}{\GLEq@{>}}
{\GLEq@{\scriptstyle>}}{\GLEq@{\scriptscriptstyle>}}}
\def\Leq{\mathchoice{\GLEq@{<}}{\GLEq@{<}}
{\GLEq@{\scriptstyle<}}{\GLEq@{\scriptscriptstyle<}}}
\makeatother
```

実際に $a \geq b$ と $c \leq d$ 書きますと, $a \geq b$ と $c \leq d$ のようになります. 数式環境の中で添え字などに使っても大きさが調整されるようになります.

$$\sum_{i \geq j} a_i a_j, \quad \sum_{i \leq j} a_i a_j$$

ここで, マクロの成り立ちについて説明しましょう. まず, \GLEq@命令の二つの引数のうち, 第一のものには, 記号の大きさを規定するための, 制御綴 (\scriptstyle など) を入れます. 第二の引数には, > または < を入れます. \GLEq@命令の内部では, \vbox 内で, これらの記号の下に罫線を 2 本入れます. \kern.2ex で示したのが, 罫線と罫線の間隔で, 0.2ex のように文字の大きさに依存する単位を用いています.

\mathchoice 命令は, plain TeX の命令で, 4 個の引数をとります. それぞれ,

\displaystyle, \textstyle, \scriptstyle および \scriptscriptstyle

で出力すべき数式を入れておきます. このあたりの事項は, TeXbook 第 17 章に詳しく載っています (とくに EXERCISE 17.15). 通常これらの大きさを表す記号は明示する必要はありません. この定義の \Geq および \Leq では, 内部に \vbox などを用いているため, これらを明示する必要がでてきています. 興味のある方は, 上の定義から, 大きさを表す制御綴を除去した場合に, 出力がどうなるかためされると理解が深まるでしょう.

\mathrel は関係記号を作るためのものです. この点は, 拙著の 18.2 「記号の新設」で詳しくふれましたので, 参照してください.

\mathchoice のかわりに, \mathpalette 命令を使って, 定義することもできます. これは, TeX 内部では, \mathchoice と同じく四つの場合に展開してから処理しています. こちらの方が, 大きさの指定をしなくてもよいので, 定義が短くなります.

% \Geq () と \Leq () の定義 (II)

% by S. Fujita 1995/03/16

```
\makeatletter
\def\GLEq@#1#2{\lower.2ex\hbox{\vbox{\hbox{#1#2$\kern-.1em}$
\kern.2ex\hrule\kern.2ex\hrule}}\kern.1em}}
\def\Geq{\mathrel{\mathpalette\GLEq@>}}
\def\Leq{\mathrel{\mathpalette\GLEq@<}}
\makeatother
```

この定義による実例です. $a \geq b$ と $c \leq d$ 書きますと, $a \geq b$ と $c \leq d$ のようになります.

$$\sum_{i \geq j} a_i a_j, \quad \sum_{i \leq j} a_i a_j$$

さらに、別の方法も考えられます。これは、plain T_EX の隠しコマンド `\vereq` (上記の `\cong` で使用) を今回の目的に合うように変えたものです。`\ialign` のような制御綴がでていますが、この説明は長くなるので省きます。

% `\Geq ()` と `\Leq ()` の定義 (III)

% by S. Fujita 1995/03/16

`\makeatletter`

`\def\@verEQ#1#2{\setbox0=\hbox{\kern-.046em%`

`\vbox{\kern.1ex\hrule width.56em\kern.2ex\hrule width.56em}\kern.046em}%`

`\lower.5pt\vbox{\baselineskip0pt\lineskip0pt%`

`\ialign{\$m@th#1\hfil#\hfil$\crcr#2\crcr\box0\crcr}}}`

`\def\Leq{\mathrel{\mathpalette\@verEQ<}}`

`\def\Geq{\mathrel{\mathpalette\@verEQ>}}`

`\makeatother`

この定義による実例です。`$a\Geq b$` と `$c\Leq d$` 書きますと、 $a \geq b$ と $c \leq d$ のようになります。

$$\sum_{i \geq j} a_i a_j, \quad \sum_{i \leq j} a_i a_j$$

第19章 X^YM_TE_X のロゴ

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

筆者は、化学構造式を描くためのマクロ集 X^YM_TE_X を作成し、@nifty の FPRINT LIB 7 にアップロードしました。T_EX のアーカイブである CTAN (the Comprehensive T_EX Archive Network) への転載、X^YM_TE_Xbook の刊行²、ホームページ³ への掲載をなっています。⁴ やや特殊なマクロ集ですが、使う人が増えているようで作者として喜んでいます。さて、今回はこのロゴを書くためのマクロを説明します。手始めに T_EX のロゴの定義を説明し、さらに X^YM_TE_X のロゴの定義を説明します。この章では、文字のカーニングなどに関する細かい命令として、`\kern` 命令、`\raise` 命令、`\lower` 命令、`\smash` 命令などが出てきます。

19.1 T_EX のロゴの定義

T_EX のロゴの定義は、

```
\def\TeX{T\kern-.2em\lower.5ex\hbox{E}\kern-.06em X}
```

のように与えられています。`\kern` 命令は、文字間のカーニングを指定するためのものです。負の値を入れると空白が小さくなります。極端には、字を重ねることもできます。`\lower` は後続の `\hbox` を基準線から下げるといふものです。この逆の命令が `\raise` でたとえば、

```
\def\TEX{T\kern-.2em\raise.5ex\hbox{E}\kern-.06em X}
```

のように定義すると、`\TEX` と書くことにより、T^EX のように出力されます。

19.2 X^YM_TE_X のロゴの定義

19.2.1 ロゴの由来

この項は、閑話休題。X^YM_TE_X のロゴについては多少苦労しましたので、その由来を述べておきましょう。すでに、ChemT_EX という名称のマクロ集が出まわっていますので、これと区別する必要があります。

¹©藤田 眞作「L^AT_EX 2_ε まくろの八衢」(12a)

²藤田眞作，“X^YM_TE_X—Typesetting Chemical Structural Formulas”，アジソンウェスレイ・ジャパン，東京（1997）

³<http://imt.chem.kit.ac.jp/fujita/fujitas/fujita.html>

⁴X^YM_TE_X の使い方については、@nifty (NIFTY-Serve) に入門編や上級編をアップしてあります。公式には、注にあげた X^YM_TE_Xbook のほか、S. Fujita, “Typesetting structural formulae with the text formatter T_EX/L^AT_EX”, *Computers and Chemistry*, **18**, 109–116 (1994); S. Fujita, “X^YM_TE_X for Drawing Chemical Structure”, *TUGboat*, **16**, 80–86 (1995) などをご覧ください。

T_EX というのが、ギリシア語を語源にしていますので、接頭語もやはりギリシア語源とするのが、造語法の決まりです。この決まりに従いたいということと、chemisty にちなんだ名前にしたいというのが念頭にあって、X_YM_TE_X の本体を作るのと同じくらい神経を使いました。ギリシア語の辞典をめくって chemisty の語源になっているものを探しまくり、やっと X_YM_TE_X という名前にたどり着きました。

X_YM_TE_X のロゴでは、Y 以外は通常のアルファベットと同じ字体になってしまっていますが、各文字はすべてギリシア文字の大文字です。先頭の X_YM を小文字に直すと $\chi\upsilon\mu$ となり、これは、chemisty の chem の語源です。T_EX もギリシャ文字で、小文字に直すと、 $\tau\epsilon\chi$ となるのはご存じの通りです。その結果、X_YM_TE_X ($\chi\upsilon\mu\tau\epsilon\chi$) は、X (χ) に始まり、X (χ) の終ることになります。また Y と上にあげ、E を下にさげて釣り合いをとるようになっていきます。これは、L^AT_EX 2_ε のやり方を換骨脱胎したものです。このようなロゴが使えない場合は、x_ym_te_x でもよいことにしてあります。ギリシア文字をアルファベットに置き換える場合は、v を u とすることもありますが、ここでは y を採用しました。

19.2.2 第一法

X_YM_TE_X のマニュアルには、X_YM_TE_X のロゴの定義として、二つあがっています。第一の定義を次に引用します。

```
%%%XyMTeX Logo: Definition 1%%%
\newcount\TestCount
\def\XyM{\ifnum\fam=-1\relax\fam=0\relax\fi\TestCount=\fam%
X\kern-.30em\smash{\raise.50ex\hbox{\fam\TestCount\Upsilon}}%
\kern-.30em{M}}
\def\XyMTeX{\XyM\kern-.1em\TeX}
```

このマクロは、Y の出力のところに工夫があります。次のような例を試してみましょう

```
\begin{center}
\begin{tabular}{ccc}
\hline
ローマン体 & ボールド体 & イタリック体 \\
\hline
{\rm $\Upsilon$} & {\bf $\Upsilon$} & {\it $\Upsilon$} \\
{\rm \TestCount=fam $\fam\TestCount\Upsilon$}&
{\bf \TestCount=fam $\fam\TestCount\Upsilon$}&
{\it \TestCount=fam $\fam\TestCount\Upsilon$}\\
{\rm \XyMTeX} & {\bf \XyMTeX} & {\it \XyMTeX} \\
\hline
\end{tabular}
\end{center}
```

出力は次のようになります。表の第一行目でわかるように、 Υ の出力は、字体を変えても変化しません。

ローマン体	ボールド体	イタリック体
Υ	Υ	Υ
Υ	Υ	Υ
$\mathcal{X}\mathcal{M}\mathcal{T}\mathcal{E}\mathcal{X}$	$\mathcal{X}\mathcal{M}\mathcal{T}\mathcal{E}\mathcal{X}$	$\mathcal{X}\mathcal{M}\mathcal{T}\mathcal{E}\mathcal{X}$

字体を変えるには地の文の `\fam` の値を数式環境の中に代入するという方法をとります。ここでは、この目的で、`\TestCount` という一時的なカウンターを使いました。二行目のように字体が変わります。この仕組みさえわかれば、あとは $\text{T}\mathcal{E}\mathcal{X}$ のロゴと同じですから、おおよそは、理解できると思います。ただ、`\smash` という命令には説明が必要です。この命令は、文字の高さを 0 にして出力するものです。このようにすると、多少上に突き出しても、行間隔の変更は起こらないということになります。

19.2.3 第二法

第一法よりももうすこしすっきりした方法です。定義を次に示しましょう。

```
%%\XyMTeX Logo: Definition 2%%
\def\UPSILON{\char'7}
\def\XyM{X\kern-.30em\smash{\raise.50ex\hbox{\UPSILON}}\kern-.30em{M}}
\def\XyMTeX{\XyM\kern-.1em\TeX}
```

この定義のポイントはやはり Υ の文字の指定の仕方です。`\UPSILON` の定義の中の `\char'7` は、 $\text{T}\mathcal{E}\mathcal{X}$ のフォント表の中の位置を示しています。たとえば、 $\text{T}\mathcal{E}\mathcal{X}$ book の Appendix F をご覧ください。`\char'6` は Σ 、`\char'13` は ff のように割り当てられています。`\char` 命令はフォント表の位置を直接指定して、文字を呼び出す機能をもっています。何も書体指定をしない場合は、ローマン体 (フォント名 `cmr10` の表。 $\text{T}\mathcal{E}\mathcal{X}$ book では、Appendix F の表 1) が呼び出されます。字体をボールド体を指定すると、`{\bf \char'6}` は Σ 、`{\bf \char'13}` は ff のようにボールド体 (フォント名 `cmbx10`) が呼び出されます。あとの命令は、すでにでてきたものですからおわかりいただけるとと思います。ここで、第 2 の定義によるものをまとめておきましょう。

```
\begin{center}
\begin{tabular}{ccc}
\hline
ローマン体 & ボールド体 & イタリック体 \\
\hline
{\rm $\Upsilon$} & {\bf $\Upsilon$} & {\it $\Upsilon$} \\
{\rm \UPSILON} & {\bf \UPSILON} & {\it \UPSILON} \\
{\rm \XyMTeX} & {\bf \XyMTeX} & {\it \XyMTeX} \\
\hline
\end{tabular}
\end{center}
```

と宣言しますと、次の出力がえられます。

ローマン体	ボールド体	イタリック体
Υ	Υ	Υ
Υ	Υ	Υ
$\mathcal{X}^{\mathcal{M}}\text{TE}^{\mathcal{X}}$	$\mathcal{X}^{\mathcal{M}}\text{TE}^{\mathcal{X}}$	$\mathcal{X}^{\mathcal{M}}\text{TE}^{\mathcal{X}}$

第20章 L^AT_EX のロゴ細見

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

L^AT_EX のユーザーとして、数年しか経っていないのに、「化学者・生化学者のための L^AT_EX—パソコンによる論文作成の手引」(東京化学同人, 1993, ISBN4-8079-0386-1) という L^AT_EX の使い方の本を出版しました。このときに、L^AT_EX のロゴには苦労したので、その話をひとつ。すでに、この本の囲み記事として、概要を書いたのですが、もう少し説明を加えたいと思います。

20.1 オリジナルの定義

L^AT_EX のロゴのオリジナルの定義は、L^AT_EX のマクロソースである latex.tex (L^AT_EX2.09) の中にあります。

定義 1

```
\def\LaTeX{\rm L\kern-.36em\raise.3ex%
  \hbox{\sc a}\kern-.15em
  T\kern-.1667em\lower.7ex%
  \hbox{E}\kern-.125emX}}
```

ちなみに、この定義の後半は、T_EX のロゴのマクロの定義に一致し、lplain.tex に含まれています。

```
\def\TeX{T\kern-.1667em\lower.5ex%
  \hbox{E}\kern-.125emX}
```

これらの定義であきらかなように、L^AT_EX の字体は常にローマン体になり、T_EX は指定した字体で印刷されます。したがって、章や節の見出しなどに使ったばあい、「T_EX と L^AT_EX」のように字体が不揃いになります。考えようによっては、オリジナルの定義だから、このような場合も仕方がないということも言えると思います。また、この場合は、審美的にも、まあ許せる範囲とおもいます。

ところが、L^AT_EX の A がスモールキャピタル (\sc) で定義してあるため、このフォントがない場合に小文字の a で代用されます。たとえば、標準 10pt で article スタイルを用いる場合、{\Huge \LaTeX} や {\scriptsize \LaTeX} とすると、

L^AT_EX L^AT_EX

のようになってしまいます。この出力が、章の見出しなどに現れた本や記事をみるがありますが、ちょっと審美的にもまずいような気がします。

¹©藤田 眞作「L^AT_EX 2_ε まくろの八衢」(13)

20.2 改良版

これを改良しようとしたのが、TUG (T_EX Users Group) で採用されているものです。定義を次に、引用します。

定義 2

```
\newcount\TestCount
\def\La{\TestCount=\the\fam %
\leavevmode L\raise.42ex\hbox{%
$\fam\TestCount\scriptstyle\kern-.3em %
\if@bold a\else A\fi$}}
\def\LaTeX{\La\kern-.15em\TeX}
```

ここでは、混同をさけるため、TUGboat の定義の命令を TUGLaTeX とおきかえます。この定義は、さすがと思わせるものです。数式モードの中に入って、添え字の大きさ (\scriptstyle) で文字 A を出力するようになっています。数式モードのなかでは、 \fam の値でフォントを制御しています。そのため、 \TestCount で地の文のフォントの \fam の値を一時保存しています。数式モードのなかで、 \fam の値を \TestCount の保存値に設定し直して、地の文のフォントを使うようにしています。

この定義で大抵の場合は、うまくゆきますが、自作のマクロなどで \vcenter を使っている場合にはうまくゆかないことがあります。² たとえば、 \famtestbox を次のような定義で作成したとします。

```
\def\famtestbox#1#2{\noindent$\vcenter{\hsize=#1\mbox{#2}}$}
```

これを使って、

```
\famtestbox{2cm}{\TUGLaTeX}
```

と入力すると、出力は次のようになります。

$\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$

A の文字が斜体になっていることに注意してください。小さな現象なので、大抵は見落としてしまうものですが、気にしだすと気になるものです。この原因は、わかってしまうとなんのことではない、上の定義に直接に起因しています。つまり、 \fam の設定値にあります。 \vcenter を使うと、 \fam の値が -1 に設定されてしまいます (通常は 0 です)。念のために、実際に地の文と \famtestbox の中の \fam の値を求めてみます。 \the\fam と $\text{\famtestbox{30pt}{\the\fam}}$ としますと、それぞれ、 0 と -1 が得られます。

この現象を回避するには、 \TestCount の値が -1 になったときに、 0 に設定し直してやればよいことになります。すなわち、上記の定義に、1 行だけ付け加えます。

定義 3

```
\newcount\TestCount
\def\La{\TestCount=\the\fam %
\leavevmode L\raise.42ex\hbox{%
\ifnum\TestCount=-1 \TestCount=0\fi%追加
```

² $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2.09$ では、 minipage 環境や parbox の中がこのような仕様になっていました。 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ では、このような現象がおこらないように改良されています。

```

 $\fam\TestCount\scriptstyle\kern-.3em %$ 
 $\if@bold a\else A\fi$}}$ 
 $\def\LaTeX{\La\kern-.15em\TeX}$ 

```

こうすると、上と同じ入力から次の出力がえられます。みごとに、L^AT_EX の A が、ほかの字の書体と同じになります。

L^AT_EX

ところが、この改良した定義でもうまくゆかないことがあります。それは、`\scriptstyle` として、添え字の小さな字体が用意されていないことがあるためです。この時は、A が他の文字と同じ大きさで出力されることとなります。たとえば、`{\Huge\bf \LaTeX}` と書くと、

L^AT_EX

のように出力されてしまいます。我慢しようと思えば思える程度ですが、L^AT_EX のロゴとしての文字面が変わってしまっています。

20.3 さらに改良

この現象にも対処するため、原稿を書きながら不都合なところが出るたびに少しずつ直してゆきました。定義は長いので、興味のある方は上掲書をみていただくとして、ここでは、定義の本質的な部分だけ引用します。

定義 4 (一部)

```

 $\def\La{\ifnum \fam=\m@ne \fam=\z@ \fi%$ 
 $\TestCount=\fam%$ 
 $\edef\aaa{\fontname\scriptfont\fam}%$ 
 $\edef\bbb{\fontname\textfont\fam}%$ 
 $\leavevmode{L}%$ 
 $\ifx\aaa\bbb \resetfontsize%$ 
 $\raise.46ex\hbox{%$ 
 ${\tmpfontsize\bf$\fam\TestCount%$ 
 $\kern-.4em\tmpkern{A}$}}%$ 
 $\else%$ 
 $\raise.42ex\hbox{${\fam\TestCount%$ 
 $\scriptstyle\kern-.3em{A}$}}%$ 
 $\fi}$ 
 $\def\LaTeX{\La\kern-.15em\TeX}$ 

```

このマクロの考え方を説明します。数式モードに入ったときに `\textstyle` と `\scriptstyle` のフォント名を検出します。このマクロでは、それぞれ `\aaa` と `\bbb` に置いています。これが等しいかどうかを `\ifx` 文で調べ、等しいときにはときに、代替のフォントを捜します。これが `\resetfontsize` の命令です。等しくないときは、定義 3 と同じ処理を行うようになっています。

定義 4 のような定義が最初からでくると、食傷気味になってしまうのですが、少しずつ知識を広げてゆくうちになんとなく書けるようになってしまうのが、 $\text{\TeX}/\text{\LaTeX}$ の不思議なところです。そのときどきの実力に応じて使えるというのが、 $\text{\TeX}/\text{\LaTeX}$ のすばらしさだと思います。 \LaTeX のロゴなど小さい小さいといわれればそれまでですが、 \TeX のプログラム言語としての側面を示すための例題として、みていただければと思います。

20.4 $\text{\LaTeX} 2_{\epsilon}$ における定義

$\text{\LaTeX} 2_{\epsilon}$ になってから、 \LaTeX のロゴの定義は、次のように変更されています (latex.ltx)。

```
\def\TeX{T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX\@}
\DeclareRobustCommand{\LaTeX}{L\kern-.36em%
  {\sbox\z@ T%
    \vbox to\ht\z@{\hbox{\check@mathfonts
      \fontsize\sf@size\z@
      \math@fontsfalse\selectfont
      A}%
    \vss}%
  }%
\kern-.15em%
\TeX}
```

$\text{\LaTeX} 2_{\epsilon}$ では、NFSS という新しい方式でフォントを指定しているため、小さな A を選択するところが見通しのよいものになります。上記の定義を簡単に説明します。まず、`\check@mathfonts` 命令で現在のフォントを調べて、縮小が必要ならフォントの大きさを計算し、`\sf@size` に設定します。`\fontsize\sf@size\z@` という命令は、`\sf@size pt` の大きさの文字で、`\z@` (0pt) の行送りを設定するものです。このフォントの命令を有効にするため、`\selectfont` 命令を宣言しています。`\math@fontsfalse` のはたらきはしらべてみましたが、よくわかりません。多分、余計な数式フォントを設定しないようにするスイッチであろうと推測しています。

この定義により、この章でのべたことからは、実用的にはまったく必要がなくなりました。しかしながら、 $\text{\LaTeX} 2_{\epsilon}$ になっても、この本章で述べた解析手法は、諸兄の参考になるのではないかと考えています。

第21章 文字列の長さ

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

文字列の長さを測って、短いときは、適当な長さで均等割にするというマクロを作ってみましょう。表題などの出力に役に立つとおもいます。必要な要素技術を説明してから、このマクロの作成に取りかかりましょう。また、内部マクロを用いて、長いマクロを分割して処理を追跡しやすくする方法も述べます。

21.1 文字列の長さの測定

21.1.1 L^AT_EX 2_ε流測量術

日本語 T_EX では、漢字の幅の単位として、zw が使われています。これは、現在使っている書体に依存する単位です。これを実際に測ってみましょう。newlength 命令によって、新しい寸法として、TestDimen を作ります。L^AT_EX 2_εのnewlength 命令は、T_EX のnewskip 命令をもとにしていますので、ここで作成した寸法TestDimen は、スキップレジスター (寸法と伸縮値) として取り扱われます。したがって、厳密には T_EX のnewdimen 命令によって作られる寸法レジスター (寸法) とは扱いが違います。簡単のため、ここでは同じようなものとしておきます。区別する必要があるときは、そのときに断ることにしましょう。寸法を測るのは、settowidth 命令を使います。

```
\newlength{\TestDimen}
\begin{center}
\begin{tabular}{rr|rr|rr}
1 文字: & \settowidth{\TestDimen}{字} \the\TestDimen &
2 文字: & \settowidth{\TestDimen}{字字} \the\TestDimen &
3 文字: & \settowidth{\TestDimen}{字字字} \the\TestDimen & \
4 文字: & \settowidth{\TestDimen}{字字字字} \the\TestDimen &
5 文字: & \settowidth{\TestDimen}{字字字字字} \the\TestDimen &
6 文字: & \settowidth{\TestDimen}{字字字字字字} \the\TestDimen & \
\end{tabular}
\end{center}
```

この結果、次の表がえられます。

¹©藤田 眞作「L^AT_EX 2_εまくろの八衢」(14)

1 文字:	9.62216pt	2 文字:	19.24432pt	3 文字:	28.86649pt
4 文字:	38.48865pt	5 文字:	48.11081pt	6 文字:	57.73297pt

21.1.2 T_EX 流測量術

もう一つの方法は、T_EX 流の方法で、`\setbox` 命令を使います。この命令は、T_EX の基本的な命令で、いろいろな局面で使えます。本章では、単に文字列の長さを測る機能に絞って説明します。

```
\setbox0=\hbox{文字列の長さ}
\the\wd0 (幅), \the\ht0 (高さ), \the\dp0 (深さ),
\dimen0=\ht0
\the\dimen0 (高さ, レジスター保存)
```

と書いてください。`\the` 命令は、この場合箱の寸法を出力するために用いています。次の出力が得られます。
57.73297pt(幅), 7.77588pt(高さ), 1.38855pt(深さ), 7.77588pt(高さ, レジスター保存)

ここでは、「文字列の長さ」を入れた箱`\box0` が作られ、その寸法が`\wd0` (幅), `\ht0` (高さ), `\dp0` (深さ) に格納されます。また、T_EX でユーザー用に用意されている`\dimen0` という寸法格納用のレジスターを使用して、保存することもできます。

21.2 文字の間隔と均等割

表題などで、文字数が少ないときに、間隔をあけて、他の長い表題とバランスを取りたいということがよくあります。いわゆる均等割という手法です。

手動でおこなうことが最も簡単で、全角の空白を用います。たとえば、2文字だと「表 題」のように2文字分の空白を、3文字だと「章 表 題」と1文字分の空白を入れる方法です。T_EX では、全角の空白を、普通の文字のようにみなしています。

もう一つの方法は、`\hbox` の機能を用いる方法です。一定の長さの`\hbox` を作っておき、各文字の間に`\hfill` を入れる方法です。たとえば、

```
「\hbox to4zw{表\hfill 題}」および
「\hbox to5zw{章\hfill 表\hfill 題}」
```

と書くと、「表 題」および「章 表 題」のように出力されます。

これらの方法は、手動のときは有効です。しかし、章表題や節表題の場合には目次や索引に使いますから、処理の過程で文字列の照合が重要になってきます。このとき、余分な全角空白や制御綴があると、正確な照合を妨げる原因になります。したがって、文字列としては「表題」「章表題」のように取り扱って、出力のときのみ「表 題」「章 表 題」とするような自動的な方法が好ましいわけです。

このような自動的な均等割を行ううまい方法があります。アスキー版日本語 T_EX では、漢字と漢字の間を規定するグルー`\kanjiskip` と、漢字と英語の間を規定するグルー`\xkanjiskip` を使って行整形などを行っています。グルー (glue) とは T_EX の用語で、文字 (箱) の間に出力する伸縮可能な空白のことです。このグルーは明示的に与えることもできます。たとえば、

```
{\kanjiskip=10pt plus 15pt minus 15pt
```

```
「\hbox to4zw{表題}」および
「\hbox to5zw{章表題}」}
```

のように宣言すると、「表 題」および「章 表 題」のように出力されます。グルーは、基本の空白値と伸びる範囲 (plus) 縮む範囲 (minus) を規定しておきます。ここでは、安全のために、中括弧でくくって効果が他に及ばないようにしておきました。

TEX では、限りなく伸縮するという機能を実現するためのうまい方法が用意されています。これを用いて、グルーを規定してみましよう。`kintou` という命令を作ります。ここでは、`hbox` の中でグルーの設定を行っていることに注意してください。したがって、効果が外に及ぶことはありません。

```
\def\kintou#1#2{\hbox to#1{%
\kanjiskip=0pt plus 1fill minus 1fill
\xkanjiskip=\kanjiskip #2}}
```

と定義したあとで、「\kintou{5zw}{均等割}」と入力しますと、「均 等 割」と出力されます。

21.3 自動均等割

文字の長さを測定する方法と、上で作成した`kintou` 命令とを組み合わせ、自動均等割を実現してみましよう。文字列の長さで処理を分岐するため、`ifdim...fi` という条件文を使います。要所に注釈を入れておきました。安全のため、ちょうどの文字幅よりも少し大きいところで判断するようにしました。場合分けを丁寧にしてゆけばよいわけですから、次の命令の定義は、簡単に理解できるとおもいます。

```
\makeatletter
\def\jidoukintou#1{%自動均等割
\settowidth{\@tempdima}{#1}%文字列の長さを測る
\ifdim\@tempdima<1.2zw \hbox{#1}%1 文字
\else\ifdim\@tempdima<2.4zw \kintou{4zw}{#1}%2 文字
\else\ifdim\@tempdima<3.4zw \kintou{5zw}{#1}%3 文字
\else\ifdim\@tempdima<4.4zw \kintou{6zw}{#1}%4 文字
\else\ifdim\@tempdima<5.4zw \kintou{6zw}{#1}%5 文字
\else \hbox{#1}%6 文字以上
\fi\fi\fi\fi\fi}
\makeatother
```

一時的に長さを保存するために、寸法`@tempdima` を用いました。これは、L^AT_EX 2_εですでにこの目的で宣言済みのものですので、改めて宣言する必要はありません。

実際にこの命令をためしてみましよう。

```
\begin{center}
\begin{tabular}{r|l|r|l}
1 文字: & \jidoukintou{文} & 2 文字: & \jidoukintou{文字} &
3 文字: & \jidoukintou{文字列} & \
4 文字: & \jidoukintou{文字出力} & 5 文字: & \jidoukintou{自動均等割} &
6 文字: & \jidoukintou{文字列均等割} & \
```

```
\end{tabular}
\end{center}
```

と入力しますと、次の出力がえられます。

1文字: 文	2文字: 文字	3文字: 文字列
4文字: 文字出力	5文字: 自動均等割	6文字: 文字列均等割

上記の定義からわかるように、`\jidoukintou` の第 1 引数には、書体の指定を入れることができますが、この使い方は想定していません。仮に用いる場合は、十分に注意して用いてください。次の三つの出力を比較すると、その理由がわかります。

```
「\jidoukintou{文字列}」 「\jidoukintou{\Large\bf 文字列}」
「{\Large\bf \jidoukintou{文字列}}」
```

と入力しますと、それぞれ「文字列」「文字列」「文字列」となります。

21.4 内部マクロの作成と利用

本解説は、マクロの作成を主眼にしていますから、ここで、内部マクロを作る練習をしてみましょう。上にあげた `\jidoukintou` の定義の中で、条件文のところは、ほかにも使える可能性がありますので、ここを取り出して、`\@jidoukintou` という命令にしてみます。

```
\makeatletter
\def\xidoukintou#1{%自動均等割
\settowidth{\@tempdima}{#1}%文字列の長さを測る
\@jidoukintou{\@tempdima}{#1}}
\def\xidoukintou#1#2{%条件判断の内部マクロ
\ifdim#1<1.2zw \hbox{#2}%1文字
\else\ifdim#1<2.4zw \kintou{4zw}{#2}%2文字
\else\ifdim#1<3.4zw \kintou{5zw}{#2}%3文字
\else\ifdim#1<4.4zw \kintou{6zw}{#2}%4文字
\else\ifdim#1<5.4zw \kintou{6zw}{#2}%5文字
\else \hbox{#2}%6文字以上
\fi\fi\fi\fi\fi}
\makeatother
```

再定義した `\jidoukintou` 命令を用いて、上の表と同じ入力から、次の結果がえられます。当然のことながら、出力結果は同じです。

1文字: 文	2文字: 文字	3文字: 文字列
4文字: 文字出力	5文字: 自動均等割	6文字: 文字列均等割

このような、内部マクロは、同じ処理がほかに必要な場合に、流用することができますので、定義の簡略化に役に立ちます。うまく使えば、処理のアルゴリズムをたどりやすくするのも役に立つでしょう。このような練習として、`\jidoukintou` を、T_EX 流に書いてみます。`\Jidoukintou` という名前にしておきましょう。条件の判断は、`\@jidoukintou` と、まったく同じ手順を踏めばよいので、これを流用しましょう。文字列の長さを測るところが違うだけですので、比較してください。

```

\makeatletter
\def\Jidoukintou#1{%自動均等割
\setbox\@tempboxa=\hbox{#1}%文字列の長さを測る
\@jidoukintou{\wd\@tempboxa}{#1}}
\makeatother

```

注意することを少し述べます。 \@tempboxa というのは、 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X } 2_{\epsilon}$ で宣言済みの箱の番号です。一時的な箱として使えます。上記で、\setbox0 とした番号 0 に相当しています。したがって、\wd\@tempboxa は、上記の \wd0 と同様に、生じた箱の幅を表しています。

実際にこの命令をためてみましょう。テストの出力形式は同じものを使います。すなわち、

```

\begin{center}
\begin{tabular}{rl|rl|rl}
1 文字: & \Jidoukintou{文} & 2 文字: & \Jidoukintou{文字} & & \\
3 文字: & \Jidoukintou{文字列} & \& \& \& \\
4 文字: & \Jidoukintou{文字出力} & 5 文字: & \Jidoukintou{自動均等割} & & \\
6 文字: & \Jidoukintou{文字列均等割} & \& \& \& \\
\end{tabular}
\end{center}

```

と入力しますと、次の出力がえられます。

1 文字: 文	2 文字: 文 字	3 文字: 文 字 列
4 文字: 文 字 出 力	5 文字: 自 動 均 等 割	6 文字: 文 字 列 均 等 割

第22章 見出しの自動均等割

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

拙著第14章には、箇条書の環境として、(j)taxonomy や (j)describe を作成し、その使用法が述べられています。これらの環境は、付録のフロッピーディスクに含まれる mathchem.sty をオプションとして宣言すれば使えるようになっています。ここでは、これをさらに改良して、前章で述べた自動均等割を見出しに応用してみましょう。

22.1 箇条書環境への応用

Jdescribe 環境と Jdescription 環境をオプションスタイル jdescrib.sty に仕立ててあります。これを最初に引用しておきましょう。

```
% jdescrib.sty
% Copyright (C) 1995/05/27 by Shinsaku Fujita
%
% リスト環境のラベルを均等割に出力するマクロ
% \begin{Jdescribe}[LABEL] ... \end{Jdescribe}
% \begin{Jdescription} ... \end{Jdescription}
%
\typeout{Option Style 'jdescrib' 1995/05/27 by Shinsaku Fujita}
%
% \kintou
% by Shinsaku Fujita 1995/05/27
%
\def\kintou#1#2{\hbox to#1{%
  \kanjiskip=0pt plus 1fill minus 1fill
  \xkanjiskip=\kanjiskip #2}}
%
% \jidoukintou
% by Shinsaku Fujita 1995/05/27
%
```

¹藤田 眞作「 $\text{\LaTeX} 2_{\epsilon}$ まくろの八衢」(14 続き)

```

\def\jidoukintou#1{%自動均等割
\settowidth{\@tempdima}{#1}%文字列の長さを測る
\@jidoukintou{\@tempdima}{#1}}
\def\@jidoukintou#1#2{%条件判断の内部マクロ
\ifdim#1<1.2zw \hbox{#2}%1 文字
\else\ifdim#1<2.4zw \kintou{4zw}{#2}%2 文字
\else\ifdim#1<3.4zw \kintou{5zw}{#2}%3 文字
\else\ifdim#1<4.4zw \kintou{6zw}{#2}%4 文字
\else\ifdim#1<5.4zw \kintou{6zw}{#2}%5 文字
\else \hbox{#2}%6 文字以上
\fi\fi\fi\fi\fi}
%
% \Jidoukintou
% by Shinsaku Fujita 1995/05/27
%
\def\Jidoukintou#1{%自動均等割
\setbox\@tempboxa=\hbox{#1}%文字列の長さを測る
\@jidoukintou{\wd\@tempboxa}{#1}}
%
% Jdescribe & Jdescription common
% by Shinsaku Fujita 1995/05/27
%
\def\Jdescribelabel#1{%
\labelsep=1zw
\hspace\labelsep
\hspace\Jdescribeindent \mbox{\bf \jidoukintou{#1}}\hfil}
\newdimen\Jdescribeindent
\Jdescribeindent=1zw
%
% Jdescribe
% by Shinsaku Fujita 1995/05/27
%
\def\Jdescribe{\@ifnextchar[{\@Jdescribe}{\@Jdescribe[文字列の長さ]}}
\def\@Jdescribe[#1]{\list{}}{%
\setbox\@tempboxa=\hbox{\bf \jidoukintou{#1}}\hfil}%文字列の長さを測る
\labelwidth=\wd\@tempboxa
\labelsep=1zw
\parsep=\z@ \itemsep=.5zh \listparindent=1zw
\advance\labelwidth\labelsep
\advance\labelwidth\Jdescribeindent
\itemindent=\labelwidth
\advance\itemindent-\leftmargin

```

```

\let\makelabel\Jdescribelabel}}
\let\endJdescribe=\endlist
%
% \Jdescription
% by Shinsaku Fujita 1995/05/27
%
\def\Jdescription{\list{}{%
\parsep=\z@ \itemsep=.5zh \listparindent=1zw
\labelwidth=\z@ \labelsep=1zw \advance\labelwidth\labelsep
\advance\labelwidth\Jdescribeindent
\itemindent=\labelwidth \advance\itemindent-\leftmargin
\let\makelabel\Jdescribelabel}}
\let\endJdescription=\endlist

\endinput

```

(例文)

```

\documentclass[a4j]{jbook}
\usepackage{jdescrib}
\begin{document}
(省略)
\end{document}

```

22.2 Jdescription 環境の使い方

使い方は、 \LaTeX 2_ϵ の `description` 環境と同じです。項目のラベルは `\item` 命令のオプション引数として与えます。

```

\begin{Jdescription}
\item[文]
1 文字では、そのまま左寄せになります。
その後に、1 文字分の空白をおき、本文を出力します。
先頭は 1 文字分下げされます。
\item[文字]
2 文字では、ラベルは全角 2 文字分の空白をあけて出力されます。
その後に、1 文字分の空白をおき、本文を出力します。
(途中省略)
\end{Jdescription}

```

次に出力を示します。ラベルの出力に、均等割の効果がでていることを確かめてください。

文 1 文字では、そのまま左寄せになります。その後に、1 文字分の空白をおき、本文を出力します。先頭は 1 文字分下げされます。

文字 2文字では、ラベルは全角 2 文字分の空白をあけて出力されます。その後、1 文字分の空白をおき、本文を出力します。

文字列 3文字では、5 文字分の領域に均等割 (文字間に全角の空白挿入) が行われます。その後、1 文字分の空白をおき、本文を出力します。

文字出力 4文字では、6 文字分の領域に均等割が行われます。その後、1 文字分の空白をおき、本文を出力します。

自動均等割 5文字では、6 文字分の領域に均等割が行われます。その後、1 文字分の空白をおき、本文を出力します。

文字列均等割 6文字では、そのまま左寄せになります。その後、1 文字分の空白をおき、本文を出力します。

最も長いタイトル 6文字以上では、そのまま左寄せになります。ラベル出力に続いて、1 文字分の空白をおき、本文を出力します。

22.3 Jdescribe 環境の使い方

Jdescribe 環境は、オプション引数に文字列を指定できるようになっています。引数指定した文字列の長さのラベルの領域がとられます。オプション指定しない場合は、6 文字分のラベルの領域がとられます。項目のラベルは `\item` 命令のオプション引数として与えます。

```
\begin{Jdescribe}[文字列均等割]
```

```
\item[文]
```

1 文字では、そのまま左寄せになります。

Jdescribe 環境の引数指定した「文字列均等割」の長さのラベルの領域がとられます。

その後、1 文字分の空白をおき、本文を出力します。

先頭は 1 文字分下げされます。

```
\item[文字]
```

2 文字では、ラベルは全角 2 文字分の空白をあけて出力されます。

「文字列均等割」の長さのラベルの領域がとられます。

その後、1 文字分の空白をおき、本文を出力します。

(途中省略)

```
\end{Jdescribe}
```

次に出力を示します。ラベルの出力に、均等割の効果がでていることを確かめてください。また、指定した文字列の長さのラベル領域が取られていることに注意してください。

文 1 文字では、そのまま左寄せになります。Jdescribe 環境の引数指定した「文字列均等割」の長さのラベルの領域がとられます。その後、1 文字分の空白をおき、本文を出力します。先頭は 1 文字分下げされます。

文字 2 文字では、ラベルは全角 2 文字分の空白をあけて出力されます。「文字列均等割」の長さのラベルの領域がとられます。その後、1 文字分の空白をおき、本文を出力します。

文字列 3文字では, 5文字分の領域に均等割(文字間に全角の空白挿入)が行われます。「文字列均等割」の長さのラベルの領域がとられます。その後に, 1文字分の空白をおき, 本文を出力します。

文字出力 4文字では, 6文字分の領域に均等割が行われます。「文字列均等割」の長さのラベルの領域がとられます。その後に, 1文字分の空白をおき, 本文を出力します。

自動均等割 5文字では, 6文字分の領域に均等割が行われます。「文字列均等割」の長さのラベルの領域がとられます。その後に, 1文字分の空白をおき, 本文を出力します。

文字列均等割 6文字では, そのまま左寄せになります。「文字列均等割」の長さのラベルの領域がとられます。その後に, 1文字分の空白をおき, 本文を出力します。

最も長いタイトル Jdescribe 環境の引数よりも長いラベルでは, ラベル出力に続いて, 1文字分の空白をおき, 本文を出力します。

22.4 マクロの説明

22.4.1 ラベル出力のマクロの解説

Jdescription 環境も Jdescribe 環境も, 内部では list 環境を使って定義してあります。list 環境のパラメータについては, 拙著 14.1 節に詳しく説明してあります。とくに, 図 14.1 を参考にしながら読むと, 理解が容易になります。まず, 両方の環境で共通に使っている命令を説明しましょう。ラベルを 1文字分下げして出力するために, `\Jdescribeindent` という寸法を新しく作成し, これを `1zw` に設定しています。ラベルの出力は, `\Jdescribelabel` が担っています。

```
\def\Jdescribelabel#1{%
\labelsep=1zw
\hspace\labelsep
\hspace\Jdescribeindent \mbox{\bf \jidoukintou{#1}}\hfil}
```

list 環境の原形では, 版面から `\labelsep` だけ突きび出した形になっています。そこで, `\hspace` 命令を用いて, まず `\labelsep (= 1zw)` だけ右へ移動して, この突き出した分を補正します。さらに, 字下げ分 `\Jdescribeindent (=1zw)` だけ右へ移動したところで, `\mbox` 内に入れたラベルを出力します。この `\mbox` の中に均等割の命令 `\jidoukintou` を使っています。

`\Jdescribelabel` 命令や $\LaTeX 2_{\epsilon}$ の `\descriptionlabel` がこのような補正を行っていることは, ずっと不思議におもっていました。通常気付かないこのなので, ここで list 環境の原形を実際に出力して, 確かめてみましょう。

```
\begin{list}{}{}
\item[突き出し] list 環境の原形は, \verb/\labelsep/だけ突き出し。
この行がその例。
\end{list}
```

と入力しますと, 次のように出力されます。きわめてわずかですが, 突き出していることを確かめてください。

突き出し list 環境の原形は, `\labelsep` だけ突き出し。この行がその例。

22.4.2 Jdescription 環境のマクロの解説

T_EX 流で環境を作るには、\ENV と \endENV の組みを作成することによります。拙著第 14.2 節で詳しく説明してありますので、ご覧ください。L^AT_EX 2_ε 流の環境作成についても説明を加えました。また、いろいろな箇条書環境の作成についてもふれてあります。

\Jdescription の定義の中は、list 環境のパラメーターを設定することによって行っています。 \parsep と \itemsep は、段落の間の垂直方向の空白を決めているものです。L^AT_EX 2_ε の description 環境よりは、小さくっております。この設定を小さくすることは、レイアウトの英文臭さをなくすコツのようです。 \listparindent は 1 文字分を字下げするように設定しました。

\labelwidth はこのように始め 0 に設定した後、\advance 命令をもちいて、\labelsep (= lzw) や \Jdescribeindent の値を加えています。 \itemindent についても同様です。マクロの定義の最後のところで、前項で説明した Jdescribelabel 命令を \makelabel 命令に代入したのちに、ラベル出力を実行しています。このとき、\makelabel は、\item がオプション引数つきで宣言されたときに実行されます。最後に、\endJdescription に \endlist を代入して、環境の終わりを示す制御綴を作成しています。

22.4.3 Jdescribe 環境のマクロの解説

\Jdescribe 環境の定義は、新しいテクニックをいくつか含んでいます。一つは、オプション引数をとる命令の定義の仕方です。 \@ifnextchar [{A}]{B} を使っています。次の文字が [ならば、A の処理を行い、そうでなければ、B の処理を行うという意味です。したがって、

```
\def\Jdescribe{%
\@ifnextchar[{\@Jdescribe}{\@Jdescribe[文字列の長さ]}}
```

と定義することによって、[があるときは、オプション引数を取得して、\@Jdescribe を実行しすること、[ないときは、標準の文字列 [文字列の長さ] を取得して、\@Jdescribe を実行しすることがわかります。この方法が、オプション引数をもつ場合のマクロの典型的な作成法です。充分練習して、習得しておく必要があります。 \setbox 命令で文字列の長さを測るところは、前の章の応用です。この長さを \labelwidth に初期設定するところを除くと、あとはほとんど \Jdescription 環境の定義と同じことに気付くでしょう。このあたりのことは、拙著の図 14.1 を参考にしながら読むと、容易に理解できると思います。

22.5 節見出しの自動均等割

節見出しや項見出しのレイアウトは、内部命令 \@sect などによって決められていることはすでに述べました。これらの内部命令のなかで、自動均等割の命令を用いると、見出しを均等割にすることができます。これまでの知識から、きわめて簡単に定義することができます。以前に、章立ての見出しを大文字に直すテクニックを紹介し、その中で \uppercase を使用しました。この部分を \jidoukintou{#8} などと変更するだけです。念のため、この部分を変更した \@sect などの定義を示しましょう。引用は大きいですが、変更したところは、星印 (*) を付けたところだけです。

```
\documentclass[a4j]{jarticle}
\makeatletter
%自動均等割
```

```

%1995/05/27 by S. Fujita
\def\kintou#1#2{\hbox to#1{\kanjiskip=Opt plus 1fill minus 1fill
\kanjiskip=\kanjiskip #2}}
\def\jidoukintou#1{%自動均等割
\settowidth{\@tempdima}{#1}%文字列の長さを測る
\@jidoukintou{\@tempdima}{#1}}
\def\@jidoukintou#1#2{%条件判断の内部マクロ
\ifdim#1<1.2zw \hbox{#2}%1 文字
\else\ifdim#1<2.4zw \kintou{4zw}{#2}%2 文字
\else\ifdim#1<3.4zw \kintou{5zw}{#2}%3 文字
\else\ifdim#1<4.4zw \kintou{6zw}{#2}%4 文字
\else\ifdim#1<5.4zw \kintou{6zw}{#2}%5 文字
\else \hbox{#2}%6 文字以上
\fi\fi\fi\fi\fi}
%章立ての見出しを均等割に
%1995/05/27 by S. Fujita
\def\@sect#1#2#3#4#5#6[#7]#8{%番号出力あり, 見出し均等割
\ifnum #2>\c@secnumdepth\def\@svsec{}\else%番号出力するか否かの条件
\refstepcounter{#1}%カウンター値を 1 ふやす
\edef\@svsec{\csname the#1\endcsname.\hskip 1em }%番号出力形式
\fi
\@tempskipa #5\relax%空白をスキップに設定 (正負の判断のため)
\ifdim \@tempskipa>\z@ %スキップが正のとき, 見出しで改行
\begingroup #6\relax%書体の指定がここに入る
\@hangfrom{\hskip #3\relax\@svsec}%番号出力
{\interlinepenalty \@M \jidoukintou{#8}\par}%タイトルを均等割 *
\endgroup%
%(注意) オプション引数#7 がないときは#8 を#7 に複写
\csname #1mark\endcsname{#7}%柱への出力内容
\addcontentsline{toc}{#1}{%toc ファイルへの出力
\ifnum #2>\c@secnumdepth
\else\protect\numberline{\csname the#1\endcsname}\fi
#7}}
\else%スキップが負のとき, 見出し (均等割) のあとの改行なし
\def\@svsechd{#6\hskip #3\@svsec \jidoukintou{#8}%タイトルを均等割 *
\csname #1mark\endcsname{#7}
\addcontentsline{toc}{#1}{%toc ファイルへの出力
\ifnum #2>\c@secnumdepth
\else\protect\numberline{\csname the#1\endcsname}\fi
#7}}%
\fi
\@xsect{#5}}%見出しのあとの段落処理など

```

```

%%%スターのついたとき
\def\@ssect#1#2#3#4#5{%番号出力なし，見出し均等割
  \@tempskipa #3\relax%空白をスキップにに設定（正負の判断のため）
  \ifdim \@tempskipa>\z0%スキップが正のとき，見出しで改行
    \begingroup #4
      \@hangfrom{\hskip #1}{\interlinepenalty \@M \jidoukintou{#5}\par}
    \endgroup% %タイトルを均等割*
  \else%スキップが負のとき，見出しのあとの改行なし
    \def\@svsechd{#4\hskip #1\relax \jidoukintou{#5}}\fi%タイトルを均等割*
  \@xsect{#3}}%見出しのあとの段落処理など
\makeatother
%(例文)
\begin{document}
\section{均等割} (本文)
  \subsection{項表題} (本文)
\section{節の表題} (本文)
\section*{番号出力なし} (本文)
\end{document}

```

このようにすると，たとえば，次のように出力されます．

1. 均 等 割
 - 1.1. 項 表 題
2. 節 の 表 題

番号出力なし

書籍などのスタイルでは，節や項の見出しが柱（ヘッダー）や目次などにも受け継がれて表示されますので，対応する出力マクロも均等割にする必要があります．ここでは，これ以上は省略しますが，辛抱強く直してゆくだけです，ためしてみてください．なお，拙著のフロッピーディスクの中に含まれている `my2book.sty` は，このような見出しの均等割を徹底的に行っておりあります．このスタイルは，実際に拙著の版下を作ったときのものですので，参考にしてください．

第23章 文字列の折り返し

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

難読の漢字に振りがなを付けるときに、ルビにすることはよく行われます。L^AT_EX 2_εでルビを実現する方法は、すでにいろいろな参考書で説明されていますので、ここでは触れません。一方、読みを示すのに、漢字の後の括弧の中に振りがなを入れる方法もあります。横書きの文書の場合に、たとえば、竿秤 (さおばかり) のように入れることもあります。竿秤 (読み) のように、2行に折り返して入れることもよく行われます。このような文字列の折り返しを自動的に行うマクロを作成してみましょう。

23.1 文字列の抽出

まず文字列から、任意の部分を取り出すマクロを定義しましょう。読みがなのマクロを作る目的からは過剰品質ですが、応用範囲の広いものを作っておきます。

```
\makeatletter
% extract #2-th to #3-th characters from a character string #1,
% then store the characters to \moji
% 1995/05/27 by S. Fujita
\newcount\TestCount
\def\ShortenChar#1#2#3{\@tempcnta\z@ \TestCount=#3%
\advance\TestCount\@ne\let\moji=\empty%
\@tfor\member:=#1\do{\advance\@tempcnta\@ne%
\ifnum\@tempcnta<#2\else \ifnum\@tempcnta<\TestCount%
\edef\moji{\moji\member}\fi\fi}}
\makeatother
```

これは、以前に作った`\shortenchar`を修正したものです。第1引数に文字列、第2引数に部分文字列の開始点、第3引数に部分文字列の終了点をそれぞれ与えます。最初`\moji`を空(`\empty`)にしておき、順次取り出した文字を付け加えてゆきます。このとき、`\xdef`を使っていますが、これは、`\global\edef`の短縮形です。`\global`は、定義を括弧の外(大域的)に及ぼすためのものというのはいすでに説明しました。一方`\edef`は`\expandafter\def`の短縮形です。`\@tfor`を使った構文はおなじみのものです。

使い方を示しておきましょう

¹©藤田 眞作「L^AT_EX 2_εまくろの八衢」(15)

```
\ShortenChar{document}{2}{5} 「部分文字列 = \moji」
\ShortenChar{document}{1}{6} 「部分文字列 = \moji」
\ShortenChar{スタイルファイルの作成}{5}{8} 「部分文字列 = \moji」
```

と入力すると、「部分文字列 = ocum」「部分文字列 = docume」「部分文字列 = ファイル」のように出力されます。

23.2 文字列の折り返し

次に、文字列を折り返すマクロを作成します。この場合は、折り返す点を、引数として与えることにします。内部で`\ShortenChar`を2度用いて、最初の文字列を`\Fmoji`、後の文字列を`\Rmoji`に格納したのち、`\setbox0`の中に縦に詰め込みます。文字の大きさを`\tiny`で組むようにしましょう。この際、縦方向の空白を除去するために、`\nointerlineskip`を入れておきます。

```
\makeatletter
\newcount\TESTCount
\def\Orikaeshi#1#2#3{%
\ShortenChar{#1}{1}{#2}\let\Fmoji=\moji
\TESTCount=#2\advance\TESTCount\@ne
\ShortenChar{#1}{\TESTCount}{#3}\let\Rmoji=\moji
\global\setbox0=\vbox{\tiny\hbox{\Fmoji}\nointerlineskip\hbox{\Rmoji}}
\makeatother
```

実際に使ってみましょう。

```
\Orikaeshi{ふりがな}{2}{4} 「\box0」
\Orikaeshi{さおばかり}{3}{5} 「\box0」
\Orikaeshi{スタイルファイルの作成}{6}{11} 「\box0」
```

と入力しますと、「ふりがな」「さおばかり」「スタイルファァィルノ作成」のように出力されます。

23.3 文字数の半分

さて、文字列に含まれる文字の数の総数を求めます。これは、すでに作成した`\characterNo`を修正したものです。求めた総数は、`\MaxCount`に保存しておきます。

```
\makeatletter
\newcount\MaxCount
\def\CharacterNo#1{\@tempcnta\z@
\@tfor\member:=#1\do{\advance\@tempcnta\@ne}%
\MaxCount\@tempcnta}
\makeatother
```

文字列の文字数が求められることは、

`\CharacterNo{スタイルファイルの作成}` 「文字総数 = `\the\MaxCount`」

と入力すると、「文字総数 = 11」と出力されることからわかります。今度は、この総数を $1/2$ にして、折り返し点を求めます。奇数のときは、上段の文字数を大きくしたいので、1 を加えてから、 $1/2$ にします。奇数かどうかの判断は、`\ifodd··\fi` の条件文を用います。割り算は、`\divide` 命令を用います。求めた半分を `\HalfCount` カウンターに保存します。

```
\makeatletter
\newcount\HalfCount
\def\HalfLength#1{\CharacterNo{#1}\HalfCount=\MaxCount
\ifodd\HalfCount\advance\HalfCount by1\fi
\divide\HalfCount by2\relax}
\makeatother
```

文字列の文字数の $1/2$ が求められることは、

`\HalfLength{スタイルファイルの作成}` 「文字総数/2 = `\the\HalfCount`」

と入力すると、「文字総数/2 = 6」と出力されることからわかります。

23.4 読みのマクロ

これで、準備が終わりました。いよいよ、読みのマクロ `\Yomi` を作りましょう。上に述べてきた説明から、内部に `\HalfLength` 命令、`\Orikaeshi` 命令を使えばよいことがわかります。折り返し点の位置は、`\HalfCount` で与えられます。`\Orikaeshi` で得た `\box0` の内容を括弧の中に入れますが、このとき、`\lower` 命令を用いて、少し下方に移動させ、見かけを整えます。

```
\makeatletter
\def\Yomi#1#2{\leavevmode\HalfLength{#2}%
\Orikaeshi{#2}{\HalfCount}{\MaxCount}%
#1\kern.2zw(\lower.2zh\box0)}
\makeatother
```

たとえば、

```
\Yomi{竿秤}{さおばかり}
\Yomi{螺旋}{らせん}
```

と入力すると、竿秤 (さおばかり) 螺旋 (らせん) という出力が得られます。

もちろん、読みを括弧内に入れる方法には例外があります。たとえば拗音や撥音が 2 行目の冒頭にくることをさげたいことなど、考慮すべきことがあります。これに対処したマクロ `\yomi` を、拙著の付録のフロッピーディスクに含まれる `nippon.sty` というオプションスタイルに入れておきました。これに関する説明が、拙著第 17 章末尾の囲み記事にでています。参考に、ご覧ください。

第24章 練習その2

文書履歴 [2004/07/30 Version 2.00]-[1995/05/27 Version 1.00]¹

これまで出てきたテクニックのおさらいです。文字列の長さを測る方法と文字列の文字数を勘定する方法は、応用範囲のひろいものですから、よく練習をしてください。例によって解答を直後に書いてありますが、解答を見ずに一度自分で解いてみてください。

24.1 文字列の長さの測定

問題 24.1 引数として与えた文字列の長さを測定して、3字分の長さを足した枠 (`\fbox` で作成) の中に中央揃えをなさい。L^AT_EX 2_ε流と T_EX 流の両方を試してみてください。

ヒント L^AT_EX 2_εで宣言済みの寸法`\@tempdima`を使いなさい。

解答 ここでは、L^AT_EX 2_ε流のみを示します。

```
\makeatletter
\def\Waku#1{\settowidth{\@tempdima}{#1}
\advance\@tempdima by3zw
\fbox{\hbox to\@tempdima{\hfill #1\hfill}}}
\makeatother
\Waku{文字列}\quad\Waku{文字列の長さ}
```

とすれば、つぎの出力が得られます。

文字列

文字列の長さ

問題 24.2 引数として与えた文字列の長さを測定して、3字分の長さを足した枠 (`\fbox` で作成) の中に左寄せをなさい。L^AT_EX 2_ε流と T_EX 流の両方を試してみてください。

解答 ここでは、L^AT_EX 2_ε流のみを示します。

```
\makeatletter
\def\LWaku#1{\settowidth{\@tempdima}{#1}
\advance\@tempdima by3zw
\fbox{\hbox to\@tempdima{#1\hfill}}}
\makeatother
\Waku{文字列}\quad\Waku{文字列の長さ}
```

¹©藤田 眞作「L^AT_EX 2_εまくろの八衢」(16)

とすれば、つぎの出力が得られます。

文字列

文字列の長さ

問題 24.3 *LaTeX 2_ε* の `\parbox` は、第一引数として寸法を与えるようになっています。この第一引数を文字列にかえて、その長さを幅にもつ `\Parbox` 命令を作りなさい。

解答

```
\makeatletter
\def\Parbox#1#2{\settowidth{\@tempdima}{#1}%
\parbox{\@tempdima}{#2}}
\fbbox{\Parbox{文字列の長さを指定}{文字列の長さを指定することによって
箱の幅を決める。}}
\makeatother
```

とすれば、つぎの出力が得られます。

文字列の長さを指定
することによって箱
の幅を決める。

24.2 文字数の勘定

問題 24.4 `\@tfor` 命令を応用して、引数として与えた文字列の前後および文字間に ♣ を入れなさい。ただし、英単語などは含まないとします。

解答

```
\makeatletter
\def\insertclub#1{\def\mojiretu{}
\@tfor\member:=#1\do{
\edef\mojiretu{\mojiretu $\clubsuit$ \member}}
\edef\mojiretu{\mojiretu $\clubsuit$}
\mojiretu}
\insertclub{文字列} \quad \quad \insertclub{記号を挿入}
\makeatother
```

とすれば、つぎの出力が得られます。

♣ 文 ♣ 字 ♣ 列 ♣

♣ 記 ♣ 号 ♣ を ♣ 挿 ♣ 入 ♣

問題 24.5 `\@tfor` 命令を応用して、引数として与えた文字列の文字間に ♣ を入れなさい。ただし、英単語などは含まないとします。

解答

```

\makeatletter
\def\Insertclub#1{\def\mojiretu{}
\@tfor\member:=#1\do{
\ifx\mojiretu\empty \edef\mojiretu{\member}\else
\edef\mojiretu{\mojiretu $\clubsuit$ \member}\fi}
\mojiretu}
\Insertclub{文字列} \quad \quad \Insertclub{記号を挿入}
\makeatother

```

とすれば, つぎの出力が得られます.

文 ♣ 字 ♣ 列 記 ♣ 号 ♣ を ♣ 挿 ♣ 入

問題 24.6 \@tfor 命令を応用して, 引数として与えた文字列の前後および文字間に\hfill による空白を入れ, 幅 5cm の枠内に入れなさい. ただし, 英単語などは含まないとします.

解 答

```

\makeatletter
\def\INSERTspace#1{\def\mojiretu{}%
{\let\hfill=\relax%
\@tfor\member:=#1\do{%
\ifx\mojiretu\empty \xdef\mojiretu{\hfill \member}\else
\xdef\mojiretu{\mojiretu\hfill \member}\fi}%
\xdef\mojiretu{\mojiretu \hfill}}%
\hbox to5cm{\mojiretu}}
\fbbox{\INSERTspace{文字列}} \quad \fbbox{\INSERTspace{記号を挿入}}
\makeatother

```

とすれば, つぎの出力が得られます.

文 字 列

記 号 を 挿 入

問題 24.7 \@tfor 命令を応用した前問までの結果を踏まえて, 引数として与えた文字列の自動均等割を実現しなさい. ただし, 英単語などは含まないとします.

ヒント

文字列の文字数を勘定して, \hbox の長さをきめ, 文字と文字の間に\hfill を入れるようにしなさい.

解 答

```

\makeatletter
\def\JIDOUkintou#1{\@tempcnta\z@\def\mojiretu{}%
{\let\hfill=\relax\@tfor\member:=#1\do{%
\ifx\mojiretu\empty \xdef\mojiretu{\member}\else%
\xdef\mojiretu{\mojiretu\hfill \member}\fi%
\global\advance\@tempcnta by1}}%

```

```

\ifnum\@tempcnta<2%
  \else \ifnum\@tempcnta<6 \advance\@tempcnta by2 \fi\fi%
\hbox to\@tempcnta zw{\mojiretu}}
例: \fbox{\JIDOUkintou{文字}} \quad \fbox{\JIDOUkintou{文字列}} \quad
\fbox{\JIDOUkintou{自動均等}} \quad \fbox{\JIDOUkintou{自動均等割}} \quad
\fbox{\JIDOUkintou{文字列均等割}} \quad
\makeatother

```

とすれば、つぎの出力が得られます。

例: 文 字 文 字 列 自 動 均 等 自 動 均 等 割 文字列均等割

問題 24.8 前問で作成した\JIDOUkintouを用いて、\Jdescribe や\Jdescriptionの別の定義を作成しなさい。

解 答 (省略) 読者自ら試してください。 \jidoukintou 命令などのかわりに用いるだけですから、容易にできると思います。

問題 24.9 前問で作成した\JIDOUkintouを用いて、\caption命令の引数を均等割にしなさい。名前を\Captionとしなさい。ただし、\Caption命令はオプション引数をとらないとします。

解 答

```

\makeatletter
\def\Caption#1{\caption[]{\JIDOUkintou{#1}}}
\def\@capttype{table}
\Caption{文字列}
\Caption{自動均等割}
\makeatother

```

とすれば、つぎの出力が得られます。

表 24.1: 文 字 列

表 24.2: 自 動 均 等 割

問題 24.10 前問と同様\JIDOUkintouを用いて、\caption命令の引数を均等割にしなさい。ただし、今回は、\caption命令はオプション引数をとるとしなさい。

ヒント \caption命令の内部命令\@captionを再定義する方法をとりなさい。

解 答 内部命令\@captionは latex.tex の中を調べてみてください。この命令は、引数を三つ取ります。三番目の引数が\caption命令の引数を受け継いでいますので、これを均等割にします。

```

\makeatletter
\let\old@caption=\@caption

```

```
\def\@caption#1[#2]#3{\old@caption{#1}[#2]{\JIDOUkintou{#3}}}  
\def\@capttype{table}  
\caption{文字列}  
\caption[自動均等割]{自動均等割}  
\makeatother
```

とすれば, つぎの出力が得られます.

表 24.3: 文 字 列

表 24.4: 自 動 均 等 割