latex.ltx リーディング
第 6 回資料

東大 TEX 愛好会

2015 年 6 月 19 日（2015 年 8 月 19 日 版）

1　繰り返し処理（前半）

今回および次回は，LATEX の内部命令のうち繰り返し処理に関するものを扱っていきたい．ひとまず今回は，\@whilenum, \@whiledim, \@whilesw の 3 命令（以下，本資料ではこれらを総称して \@while 系命令と呼ぶことにする）を扱うことにする．latex.ltx を覗く前に，これらの命令の使用法を確認しておく．

1.1　\@while 系命令の使い方

\@whilenum はカウンタレジスタが特定の条件を満たす間だけ繰り返しを実行する命令で，例えば次のようにして利用する．

```
\makeatletter
\@tempcnta=\z@
\@whilenum\@tempcnta<3\do{\advance\@tempcnta\@ne \the\@tempcnta}
\makeatother
```

これにより，紙面には “123” と印字されることになる．また，\@whiledim はカウンタレジスタの代わりに寸法レジスタを用いて同様のことを実現する．

次に \@whilesw は，前々回の「LATEX の環境入門」で扱ったようなスイッチを終了判定に用いて繰り返し処理を行う命令である．

```
\makeatletter
\newif\ifhappy
\happytrue
\@tempcnta=\z@
\@whilesw\ifhappy\fi%
  {\advance\@tempcnta\@ne \ifnum\@tempcnta=3 I'm tired. \happyfalse\else I'm happy. \fi}
\makeatother
```

これにより “I’ m happy. I’ m happy. I’ m tired.” を得る．

1.2　\@whilenum と \@whiledim の実装

\@whilenum と \@whiledim は実装もほぼ同じなのでまとめて説明する．

```
\long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
    #2\relax}\fi}
\long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
        \else\expandafter\@gobble\fi{#1}}
\long\def\@whiledim#1\do #2{\ifdim #1\relax#2\@iwhiledim{#1\relax#2}\fi}
\long\def\@iwhiledim#1{\ifdim #1\expandafter\@iwhiledim
       \else\expandafter\@gobble\fi{#1}}
```

846, 847 行目が \@whilenum の定義となっている．\@whilenum の書式は 1.1 節で確認したように，

\@whilenum〈条件式〉\do{〈繰り返す処理〉}

であるが，846 行目のパラメータテキストを見ると，#1 が〈条件式〉，#2 が〈繰り返す処理〉であることがわかる．したがって，\@whilenum の定義内部を見やすいように書き下すと，次のようになる．

```
\ifnum〈条件式〉\relax
  〈繰り返す処理〉\relax
  \@iwhilenum{〈条件式〉\relax〈繰り返す処理〉\relax}
\fi
```

はじめに与えられた〈条件式〉が，最初から「満たされないもの」(偽) であった場合，上の定義から \@whilenum は何もせずに終了する．逆に，はじめに与えられた〈条件式〉が「満たされるもの」(真) であった場合，〈繰り返す処理〉が一度実行されたあと，残りの処理は \@iwhilenum に引き継がれている．

次に，\@iwhilenum の定義内部を考える．\@iwhilenum の引数は 1 つだけであり，さきほど \@whilenum の定義内部で {〈条件式〉\relax〈繰り返す処理〉\relax} が渡されていることがわかった．したがって，\@iwhilenum の定義内部を見やすいように書き下すと，次のようになる．

```
\ifnum〈条件式〉\relax
  〈繰り返す処理〉\relax
  \expandafter\@iwhilenum
\else
  \expandafter\@gobble
\fi
{〈条件式〉\relax〈繰り返す処理〉\relax}
```

最初の 2 行は \@whilenum とまったく同一である．後半も〈条件式〉が真である場合には

```
\@iwhilenum{〈条件式〉\relax〈繰り返す処理〉\relax}
```

に展開されることになるため，全体として \@whilenum と同様になることがわかる．これにより，〈条件式〉が真である間〈繰り返す処理〉が繰り返されることが確認できる．

逆に〈条件式〉が偽である場合，後半部分は次のように展開される．

```
\@gobble{〈条件式〉\relax〈繰り返す処理〉\relax}
```

\@gobble は引数を飲み込む LaTeX のマクロで，latex.ltx の 726 行目で定義されている．関連する定義とともに，ここで記しておこう．

```
726 \long\def \@gobble #1{}
727 \long\def \@gobbletwo #1#2{}
728 \long\def \@gobblefour #1#2#3#4{}
```

以上で，〈条件式〉が偽となると，それ以上繰り返しが起こらないことも確認できた．

\@whiledim も \@whilenum のカウンタレジスタ用の命令（\ifnum）が寸法レジスタ用の命令（\ifdim）に置き換わる程度で，ほとんど違いがない．

■疑問 1　847 行目，#2 のあとの \relax はなぜ必要なのか．

1.3　\@whilesw の実装

定義は以下に示す通りである．

```
853 \long\def\@whilesw#1\fi#2{#1#2\@iwhilesw{#1#2}\fi\fi}
854 \long\def\@iwhilesw#1\fi{#1\expandafter\@iwhilesw
855         \else\@gobbletwo\fi{#1}\fi}
```

これも \@whilenum と \@whiledim ほどではないがかなり類似した定義となっている．スイッチはそれ自体が条件分岐機能と真偽情報を保持しているため，\@whilenum などより定義が簡単である．なお，パターンマッチに \do ではなく \fi が用いられているのは \loop の事情と同様と考えられる（第 1 回資料 2.2 節参照）．

2 \newcommand の動作（まとめ）

これまで数回に渡って考えてきた \newcommand の動作についてまとめようと思う．そもそも \newcommand の主要な機能として，

- 重複定義のチェック
- 第一引数のオプション化
- 引数に \par を含めることの可否の操作（\def でも \long をつけるだけで大丈夫だけど…）

が挙げられるが，それぞれ

- 重複定義のチェック → \@argdef 中の \@ifdefinable の働き
- 第一引数のオプション化 → \@newcommand の展開で分岐する，\@xargdef の働き
- 引数に \par を含めることの可否の操作 → \@star@or@long の働き

とであることがわかった．以降は，\renewcommand，\DeclareRobustCommand 等の関連のある制御綴について考えていきたいと思う．

3 \renewcommand の動作

制御綴を再定義する \renewcommand の実装について考察していく．まずは，latex.ltx 中の定義を載せる．

```
\long\def\@reargdef#1[#2]{%
  \@yargdef#1\@ne{#2}}
\def\renewcommand{\@star@or@long\renew@command}
\def\renew@command#1{%
  \begingroup \escapechar\m@ne\xdef\@gtempa{{\string#1}}\endgroup
  \expandafter\@ifundefined\@gtempa
     {\@latex@error{\noexpand#1undefined}\@ehc}%
     \relax
  \let\@ifdefinable\@rc@ifdefinable
  \new@command#1}
```

具体例を通じて考えていこう．以下のようなソースコードを処理するとする．

```
\renewcommand{\ほげ}[1]{\TeX は#1です．}
```

上のコードを定義に従い，適宜展開・実行をしてゆくと次のようになる．

```
   \renewcommand{\ほげ}[1]{\TeX は#1です．}

-> \@star@or@long\renew@command{\ほげ}[1]{\TeX は#1です．}

   (\@star@or@long の動作により, \l@ngrel@x に\long または\relax が代入される)

-> \renew@command{\ほげ}[1]{\TeX は#1です．}

-> \begingroup \escapechar\m@ne\xdef\@gtempa{{\string\ほげ}}\endgroup
   \expandafter\@ifundefined\@gtempa
      {\@latex@error{\noexpand\ほげundefined}\@ehc}%
      \relax
   \let\@ifdefinable\@rc@ifdefinable
   \new@command\ほげ [1]{\TeX は#1です．}
```

このコードを分析してみよう．まず，\begingroup から始まるグループ内を考えてみる．\escapechar とは，\string で制御綴を文字トークンに展開する際のエスケープ文字（カテゴリーコード 0 の文字）を定める TEX

プリミティブのパラメータであり，例えば

```
\escapechar=63\relax
\string\TeX
```

などすれば，以下はエスケープ文字として “?”（文字コード 63）が使用されるため，実行結果は，

?TeX

となる．もちろん，\escapechar のデフォルト値は 92（文字 “\” の文字コード）である．

ただし，今回のように，\escapechar に 0 未満または 256 以上の値を設定すると，\string による文字トークンへの展開の際，エスケープ文字として何も挿入されなくなる．従って，

```
\begingroup \escapechar\m@ne\xdef\@gtempa{{\string\ほげ}}\endgroup
```

の実行により，（グローバルな制御綴として）\@gtempa が “{ ほげ }” と定義されることになる．\begingroup と \endgroup は，\escapechar の値の変更による影響が，外部に漏れ出ないように存在していると考えられる．

■疑問 2　\@gtempa を，“ほげ” でなく “{ ほげ }” と定義している理由が分からない．

次の行の理解は容易い．\@ifundefined により，制御綴 \ほげ（\@gtempa を展開して得られる）が定義済みなのかどうか判定し，未定義である場合エラーを返す．定義済みであった場合，以降で使用される \@ifdefinable の定義を \renewcommand 用の \@rc@ifdefinable で書き換え，\new@command に進む．以下では，\renewcommand の動作の追跡を一旦やめ，\@ifdefinable 系の実装をまず理解することにする．

4　\@ifdefinable の動作

さて前節で話題になった \@rc@ifdefinable の中身であるが，latex.ltx の関連部分を抜き出すと，以下のようになる．

```
649 \long\def\@ifdefinable #1#2{%
650       \edef\reserved@a{\expandafter\@gobble\string #1}%
651      \@ifundefined\reserved@a
652          {\edef\reserved@b{\expandafter\@carcube \reserved@a xxx\@nil}%
653           \ifx \reserved@b\@qend \@notdefinable\else
654             \ifx \reserved@a\@qrelax \@notdefinable\else
655               #2%
656             \fi
657           \fi}%
658          \@notdefinable}
659 \let\@@ifdefinable\@ifdefinable
660 \long\def\@rc@ifdefinable#1#2{%
661   \let\@ifdefinable\@@ifdefinable
662   #2}
```

\@@ifdefinable は，他の部分で別途定義されているわけでもなく，659 行目が示す様に単なる \@ifdefinable のコピーである．

従って，\@rc@ifdefinable の機能は，単に \@gobble と等しいように思われる（今後より正確にその意味を理解できるのかもしれないが）．以下では，\@ifdefinable 自体の実装を考えていこう．

まず，\newcommand の実装などを参照することにより，\@ifdefinable の用法は以下のようであると推定される．

```
\@ifdefinable{〈チェックしたい制御綴〉}{〈定義可能である場合に実行されるコード〉}
```

このコードは比較的簡単に読んでいくことができる．まず冒頭の

```
\edef\reserved@a{\expandafter\@gobble\string #1}
```

であるが，これは \string で制御綴を文字トークンとして展開した後に，\@gobble で冒頭のエスケープ文字を取り除くことにより，制御綴名そのものを \reserved@a として得ている．

次に，\@ifundefined による分岐が入る．\reserved@a として得た制御綴名が既に定義済みであった場合 \@notdefinable が，そうでない場合中括弧内のコードが展開される．ここで \@notdefinable は 1058 行目で

```
1058 \gdef\@notdefinable{%
1059  \@latex@error{%
1060    Command \@backslashchar\reserved@a\space
1061    already defined.\MessageBreak
1062    Or name \@backslashchar\@qend... illegal,
1063    see p.192 of the manual}\@eha}
```

と定義されており，エラーを返す制御綴であることが分かる．

さらにその後，\reserved@b の定義に入る．\@carcube の定義

```
582 \def\@carcube#1#2#3#4\@nil{#1#2#3}
```

を見る限り，考えている制御綴名が一文字（\@など）である場合，

```
\reserved@b <- @xx
```

の様に代入され，二文字以上（\hoge など）である場合，

```
\reserved@b <- hog
```

となるように考えられる．要するに，制御綴名の内，前 3 文字を抜き出した文字列（3 文字以下の名前なら，末尾に適当に x を付け加えたもの）を \reserved@b として得ている．

また，その後に登場する \ifx の条件分岐では，それぞれ \@qend と \@qrelax との一致を調べているが，793 および 794 行目において，

```
793 \edef\@qend{\expandafter\@cdr\string\end\@nil}
794 \edef\@qrelax{\expandafter\@cdr\string\relax\@nil}
```

と定義されていることから分かるように，これは単純に制御綴名が end から始まらないか，relax という名では無いかをチェックしている．

これら全ての条件分岐を切り抜けて初めて，この制御綴は定義可能と判断され，第二引数に相当する目的のコードが実行されるようになっている．