LaTeX 文章から HTML 文章へ変換するツールの製作

図師　博章

目　次

——LaTeX 文章から HTML 文章へ変換するツールの製作——

1　はじめに

(1) TeX(LaTeX) とは

TeX は組版ソフトの一種である。組版とは印刷関係で活字を組んで版を作成することを意味している。TeX はその組版をコンピュータ上で行うためのソフトウェアである。また、TeX の機能を強化したものが LaTeX である。

LaTeX には次のような特徴がある[1]。

- ソフトウェアがフリーで提供されており、誰でも無償 で使用することができる。また、オープンソースである為自由に中身を調べたり改良することができる。
- MS-DOS, Windows, Macintosh, UNIX(Linux 含む) 等、様々な種類の OS に対応している。その為、環境の構築が容易である。
- 文章はプレーンテキストで構成されており、普通のテキストエディタ等で編集できる。
- 組版を目的としたソフトである為、美しい文章を作成することができる。特に数式は定評があり、数式をテキストで表す事実上標準となっている。

また、LaTeX は主に学者や学生に利用されており、多くの学会や学術出版社が TeX による投稿を受け付けていることからも、一部では有名なソフトウェアであると言える。

(2) 変換ツール製作の経緯

LaTeX の文章作成は Microsoft 社の Microsoft Word 等のワープロソフトとは違い、プログラミングの要素が含まれる。ワープロソフトが直感的に文章を装飾していくのに対し、LaTeX ではコマンドを使用して文章を処理するのである。例えば文章内で太字の文字を表示したければ `\textbf{太字}`と記述する。

また LaTeX 文章の構成は、コマンドを使用する点やプレーンテキストで処理するという点から HTML の構成と相似している。例えば、前述の例を HTML タグで記述すると`<B>太字</B>`となる。

私はこの LaTeX と HTML の構成が相似しているという点から、何らかのプログラム言語を使用して LaTeX のコマンドを HTML のタグに変換することが可能ではないかと考え、また現在そのようなプログラムは存在するのかどうか調べた。

調べた結果、LaTeX2HTML(2)、TtH(3)、TeX4ht(4)、HEVEA(5)の 4 つの変換ツールを Web 上で発見した。また、これらは全て英語環境で製作及び提供されており、唯一 LaTeX2HTML のみ有志の人物が日本語化を行い Web 上で公開されていた。

この 4 つの変換ツールについて、Web 上にあるマニュアルやプログラムその物を実際に使用し調べてみたのだが、どれも私の望む変換ツールとはいい難い物であった。具体的にどのような部分に不満を持ったのかということであるが、まず基本的な部分として、4 つの変換ツールが全て英語環境で開発及び提供されている為、日本語に対応していないという点である。しかし、これについては LaTeX2HTML が日本語化されているのでそれを使用すれば解決できる問題であるともいえる。

次に環境の構築が難解であるという点である。これについては前述した 4 つの変換ツールは基本的に LaTeX のコマンド全般に対応している為、変換ツール本体の他に外部ツールも利用して処理を行っていることが理由として挙げられる。

また、これら言語的な問題や構築環境の問題に加えて変換後の HTML 文章が多少複雑な物になってしまうことも不満点の一つとして挙げておく。これは、変換した HTML 文章を使用者が好みのレイアウトに調整しようとする際の妨げとなってしまうからである。

では実際に私が求めている変換ツールはどのような物かということであるが、私の求めている変換ツールとは、まず「環境の構築が簡単」という点と、「規格に忠実な HTML に変換できる」という点の二点を満たしたツールである。しかし、前述の変換ツールではこれら二点を同時に満たしている物は存在しなかった。

また、私が求めている変換ツールは少なくとも私と同様に LaTeX を文系論文の作成をする為にのみ使用しているユーザーにとっては需要があるのではないかと考えた。これは、私自身が前述の変換ツールについて実際に使用して調べている時に実感したことなのだが、数式等を使わない人にとっては外部プログラムによる処理なども必要ではなく、その為に環境構築の部分で無駄な労力が発生するのはユーザーにとってデメリットでしかないと考えたからである。

以上のような経緯から、「容易な環境構築」及び「規格に忠実な HTML に変換する」という 2 つの要点に特化した変換ツールを製作するに至ったのである。

2　変換ツールの設計

(1) 変換ツール製作の概要

変換ツール製作の経緯は前述の通りであるが、具体的に製作に入るに当たって初めに大まかな方向性を定めることにした。

(i) 変換ツール製作の目的

変換ツールを製作する目的であるが、製作する以上私のみが使用するものを製作するのではなく、全般的な文系論文を変換対象とし、文系論文の作成者が論文を Web 公開するための手助けとなれるような変換ツールを製作するのを最終的な目的とすることにした。

(ii) 変換対象とする文章

変換対象とする文章であるが、文系論文が問題無く変換できる精度のツールを製作することにした。したがって、数式等の数学系コマンドには技術的な面も含め対応を見送った。また、文系論文には縦書きと横書きがあり LaTeX では両方の書式で文章を作成することができるが、今回はコンピュータで論文作成する際に標準とされる横書きの文章のみを変換の対象とすることにした。

(iii) 文系論文を作成する LaTeX について

文系論文を作成する LaTeX についてであるが、これは日本語で TeX を使う上で多くのユーザーが使用している pLaTeX 2_ε(日本語化された LaTeX 2_ε) で作成された文章を変換対象とした（以後文章内で LaTeX と表記する部分は pLaTeX 2_ε のこととする)。

ちなみに、今回変換する為のサンプル文章を作成した LaTeX のバージョンは以下の通りである。

```
This is pTeX, Version 3.141592-p3.1.3
(sjis) (Web2C 7.5.2)
pLaTeX2e <2001/09/04>+0
(based on LaTeX2e <2001/06/01> patch level 0)
```

(iv) 変換する HTML 文章の仕様

変換する HTML についてであるが、これは前述した通り変換後に容易に手直しができるよう、規格に沿った HTML 文章に変換できるように心掛けることとした。規格についてであるが、これは HTML の規格の一つである『HTML 4.01 Transitional』を使用することにする。

また、出来るだけ LaTeX 文章に忠実な変換をする為に、HTML の規格内でも推奨されている CSS(スタイルシート) を使用し、HTML タグと CSS の 2 つのファイルで LaTeX 文章を HTML 上で再現することにした。

(2) 製作に使用するプログラム言語の決定

変換ツールを製作する為のプログラム言語であるが、私は Python というプログラム言語を採用し変換ツールを製作することにした。

何故 Python を使用するのかということだが、まず LaTeX2HTML で使用されている Perl より文法がシンプルでより直感的であったという点と、次に前述の概要に含んだ「比較的環境の構築が容易」という部分に当てはまるという点という二つの利点が存在したからである。

また、今回製作するに当たって使用した Python のバージョンは Python-2.2.3(released May 30, 2003) 及び 2.3.4(released May 27th, 2004) であり、製作環境は Windows98 及び 2000 である。

(3) 製作するに当たっての準備

概要及び製作に使用するプログラム言語が決定した所で、次に製作する為の準備に取り掛かることにした。

(i) 相関表の作成

まず、LaTeX のコマンドをどの HTML タグに変換すれば良いのかを調べ、相関表(6)の作成に取り掛かった。また、その際表に記述していく LaTeX コマンドは文系論文で使用されると予測されるコマンドだけを選び記述していくこととした。

(ii) ブレインストーミング及び KJ 法による作成手順の決定

次に、実際の製作に入るに当たって製作の手順及びどのようにプログラムを用いるかなどをブレインストーミングを使用して変換すべき箇所及びその方法を抽出し、KJ 法によってそれを構造化及び順序化することにした。

KJ 法によって順序化された製作の大まかな手順は次の通りである。

1. プレアンブル部分の処理
   (a) 文頭、文末の処理
   (b) タイトル、著者、日付の処理
2. 本文部分の処理
   (a) `\begin{...}-\end{...}`形式のコマンドの処理
   (b) `\command{...}`形式のコマンドの処理
   (c) `{\command ...}`形式のコマンドの処理
   (d) 見出しの処理
   (e) アクセント、特殊記号の処理
   (f) 脚注の処理

（g）\verb の処理

（h）表の処理

今回は以上の作業手順に沿って製作し、手順に表記した全ての処理に対応させることを目標として変換ツールの製作に入ることにする。

3　変換ツールの製作

(1) プログラム全体の構成

まずプログラム全体の構成についてであるが、様々な試行錯誤をした結果、一行ずつ読みこんで処理していく形を取ることにした。この方法は変換ツール使用者に多少の制限を強いてしまうことになるが、一括読み込みでは見出しの処理が不完全なものになってしまう為、一行読み込みの方法を採用した。

また、プログラムの構成はそれぞれのコマンドの処理を関数として宣言し、メインの処理からその関数へ飛ぶように構成した。これによってメインの処理が複雑になる事を避けた。

以後、プログラムの説明の為に資料編 1 のプログラムソースを参照しながら説明を行っていくこととする。以後文章中に特に指定の無い行数表示は資料編 1 プログラムソースの行数を指しているものとする。

(2) プレアンブル部分の変換処理

\begin{document}より上に記述されているプレアンブル部分の変換についてであるが、LaTeX のプレアンブル部分とは、文章のレイアウトや使用するパッケージの宣言、後は文章の題名、著者、日付等を記述して置く部分である。

このプレアンブル部分は資料編 4 LaTeX 文章サンプルの ll.1-6 のように記述される。また、これを資料編 5 HTML 文章サンプルの ll.1-8 のような文章に変換できるように処理をすることとした。

このプレアンブル部分の変換に関して説明すると、まずプログラムのメインループ部分で

```
0454: if state == preamble:
0455:    if line.find('\\begin{document}') != -1:
0456:      print html_head(title_author_date['title']),
0457:      state = document
0458:    else:
0459:      re_title_author_date.sub(do_title_author
_date, line)
```

以上のような処理を行っている、これは`\begin{document}`が読み込んだ line の中にあったら HTML のヘッダ部分の文章 (ll.55-64) を書き出し、処理を document モードに切り替えるという処理である。

また、else 以下の部分は題名、著者、日付が line の中に存在したら

```
 0041: title_author_date =
 {'title':'', 'author':'', 'date':''}
```

という部分に収納するという命令 (ll.66-70) で、HTML のヘッダを書き出す際に題名があればそこから取り出し`<title></title>`の間に書き込む。

この収納された題名、著者、日付の部分は本文の処理中に`\maketitle`が来ることによって以下の部分で処理が実行される

```
0099: def maketitle():
0100:     data = ''
0101:     if title_author_date['title'] != '':
0102:         data = '<h1 class="title">%s</h1>\n' %
(title_author_date['title'])
0103:     if title_author_date['author'] != '':
0104:         data += '<h2 class="author">%s</h2>\n
<hr>\n' % (title_author_date['author'])
0105:     if title_author_date['date'] != '':
0106:         data += '<p class="date">%s</p>' %
(title_author_date['date'])
0107:     return data
```

これは題名、著者、日付が前述した l.41 の部分に収納されているかを l.101, l.103, l.105 の部分で調べ、それらが空でない場合に l.102, l.104, l.106 の命令によって出力するという命令である。また、l.107 の return data という命令は ll.100-106 で処理した内容を再びメインループ部分へ戻すという命令である。

(3) 主要コマンドの処理

LaTeX で使用されるコマンドには大まかに 3 種類ある。

- \begin{document}〜\end{document}のような\begin{...}と\end{...}で囲まれるタイプのコマンド
  (以下 begin-end 型と表記する)
- \textbf{**太字**}のような\command{...}とするタイプのコマンド

(以下 command1 型と表記する)

・{\Large **大きい字**}のような{\command ...}というタイプのコマンド **(以下 command2 型と表記する)**

以上の 3 種類である。この 3 種類のコマンドはそれぞれ違う処理を行っている。

(i) begin-end 型コマンドの処理

begin-end 型のコマンドは基本的に複雑な処理をせずにそのまま HTML タグに置き換えられる場合が多い。その為、Python の辞書という機能を用いて処理することにした。正規表現で抽出されたコマンドを辞書テーブルの中にあるコマンドと照合し HTML タグを出力させるのである。

この begin-end 型で使用している辞書テーブルは ll.42-51 に記述されている物であり、ここに正規表現で抽出されたコマンドを照合することによってそれに対応する文字列が出力されるのである。例えば正規表現で enumerate という文字列が抽出された場合、environment_table に照合すると ol という文字列が出力されるのである。

```
0435: def do_begin(m):
0436:     command = m.group(1)
0437:     if environment_table.has_key(command):
0438:         return '<%s>' % (environment_table[command])
0439:     else:
0440:         return '<div class="%s">' % (command)
```

次に実際の処理の部分だが、ここでは前述の通り正規表現で抽出された文字列を l.437 の命令によって environment_table という辞書テーブ

ルに照合し、もしその文字列が辞書テーブル内に存在すれば l.438 の処理を行い、文字列が辞書テーブル内に存在しない場合は ll.439-440 の処理が行われるのである。この処理については、ll.442-447 の do_end 部分についても同様の処理が行われる。

また、begin-end 型のコマンドであるが\begin{verbatim}〜\end{verbatim}及び\begin{tabular}〜\end{tabular} については前述の処理では再現できない為、個別の処理を行った。

\begin{verbatim}〜\end{verbatim}については以下のような処理を行った。

```
0460:     elif state == verbatim:
0461:         if line.find('end{verbatim}') != -1:
0462:             print '</pre>'
0463:             state = document
0464:         else:
0465:             print line.encode('shift_jis')
0472:     elif state == document:
0473:         if line.find('begin{verbatim}') != -1:
0474:             print '<pre>'
0475:             state = verbatim
```

これは document を処理中に\begin{verbatim}があれば<pre>を出力し、モードを verbatim に切り替えるというものである。また、verbatim モードでは\end{verbatim}があるまで何の処理もせずに line を出力し、\end{verbatim}があれば</pre>を出力しモードを document に戻すという処理を行っている。

もう一つの\begin{tabular}〜\end{tabular}は後に表の処理を説明する部分で述べる。

（ii）command1 型のコマンドの処理

次に command1 型のコマンドについてであるが、この command1 型は前述の begin-end 型より難解であり begin-end 型の時には無かった問題点が存在する。それは\command{...} の {...} の中に別の LaTeX コマンドが入っている可能性があるという物である。このケースが何故問題であるのかと言うと、正規表現の段階で {} の中に {} があると外側の {} の { と中にある {} の } が一組として扱われてしまい、正確に文章が抽出できないという不具合が起きてしまうのである。例えば、\underline{\textbf{太字に下線}}というようなコマンドであるとエラーを起こしてしまうのである。

この問題を解決するための方法として私は中にあるコマンドから順に変換することにした。つまり正規表現の段階で{}の中に{}が無いものを抽出するようにしたのである。また、前述のような場合一行を一回読み込んで処理しただけでは処理できないので while ループを使い変換対象のコマンドがその一行から無くなるまで一行を繰り返し処理することにした。

実際のプログラムには以下のように記述した。

```
0523: while re_command1.search(line):
0524:     line = re_command1.sub(do_command1, line)
```

つまりこの部分の処理は正規表現 re_command1 が見つかるまでこの処理を続け、もし見つかれば変数 do_command1 にその文字列を送り処理するのである。

ちなみに関数 do_command1 の処理は

```
0378: def do_command1(m):
0379:     command = m.group(1)
0380:     parameter = m.group(2)
0381:     if command == 'item':
0382:         command += '_dt'
0383:     if re_font.match(command):
0384:         return html_font(command, parameter)
0385:     elif command_table.has_key(command):
0386:         return command_table[command](parameter)
0387:     else:
0388:         return '<span class="%s">%s</span>' %
 (command, parameter)
```

のように大まかには begin-end 型と同様の処理であるが、後述する font 関連の処理を別処理とする為に l.383 の命令で font 関連のコマンドだけ違う関数へ送り処理するようにしてある。

また、begin-end 型と同様に l.385 にて抽出した文字列を command_table という辞書テーブルに送り処理するのであるが、この辞書テーブルは begin-end 型の時に使用した environment_table とは違い、command_table は以下のようになっている。

```
0243: command_table = {
0244:     'section': html_section,
0245:     'subsection': html_subsection,
0246:     'subsubsection': html_subsubsection,
```

```
0247:     'part': html_part,
0248:     'paragraph': html_para,
0249:     'subparagraph':html_subpara,
0250:     'footnote' : html_footnote,
0251:     'item_dt': html_dt_item,
0252:     'underline': html_underline,
0253:     'item': '<li>',
0254: }
```

これは、item を除く他のコマンドが照合された場合、直接対応した文字列を出力するのではなく、対応する部分に記述された関数へ飛び処理をするのである。例えば section の場合は html_section という関数へ飛び処理を実行するのである。

(iii) command2 型及びフォント関連の処理

command2 型の処理に関しては command1 型の処理とほぼ同様であるため特に記述すべき部分は無い。ただ command1 型のコマンドの時と同様フォント関係のみ独自の処理を行う。何故フォント関係のみ別処理を行うのかであるが、フォント関係は他の command2 型と同様の処理を行うと不具合が起きる場合があった為である。その場合とは`\textbf{\Large 大きい太字}`のように記述されていた場合である。この場合通常の処理で変換すると command1 型のコマンドが処理された段階で`<span class=''textbf''>\Large 大きい太字</span>`となってしまい`\textbf` の部分が command2 型のコマンドの正規表現にヒットしない為、正常な変換が出来ないという不具合が出てしまうのである。

この為フォント関係のみ別に処理することにしたのである。その処

理とは ll.83-96 の部分である。これはまず command1 型でヒットしたフォント関係のコマンドを、さらに l.84 の search で中身にフォントサイズ関係のコマンドが無いか検索する。そして、もし存在すれば書き出す文字列を{}で囲み出力するというものである。

また、そうして出力することによって再びメインループ部分の

```
0526: while re_command2.search(line):
0527:     line = re_command2.sub(do_command2, line)
```

にヒットし l.89 以降の処理を実行するのである。

(4) 見出しの処理

次に見出しの部分の処理についてである。見出しの部分は command1 型のコマンドであり、途中の処理までは command1 型の処理と同様なのであるが、この LaTeX の見出しには自動で番号を付ける機能がある為、ただ変換するだけでは LaTeX の見出しを再現をすることはできないのである。

その為、Python のプログラムを用いてこの部分を再現することにした。まず自動で番号を付ける機能を再現するために、

```
0038: section_num, subsection_num, subsubnum, part_num,
foot_num, verb_num = 0, 0, 0, 0, 0, -1
```

のように seciton_num や subsection_num 等を作り、またそれに 0 を代入しておく。

次にプログラム部分であるが、

```
0109: def html_section(param):
```

```
0110:     global section_num, subsection_num, subsubnum
0111:     section_num += 1
0112:     subsection_num = 0
0113:     subsubnum = 0
0114:     return '<h2 class="section">%d   %s</h2>'
% (section_num, param)
0116: def html_subsection(param):
0117:     global section_num, subsection_num, subsubnum
0118:     subsection_num += 1
0119:     subsubnum = 0
0120:     return '<h3 class="subsection">
%d.%d   %s</h3>' % (section_num, subsection_num, param)
```

というように section がヒットする度に section_num に 1 を足していき、出力する際に section_num を代入していくのである。これは、subsection や subsubsection の処理に関しても同様である。

また、section がヒットする度に subsection_ num と subsubnum は 0 にリセットされ subsection がヒットする度に subsubnum は 0 にリセットされるような仕組みとなっている。

(5) アクセント、特殊記号の処理

アクセント及び特殊記号は HTML で表現できる物に制限がある為、表現できる物は変換し、表現できないものはタグで囲むこととした。タグはそれ自身意味を持たないタグであり、タグで囲むことによって変換前に対応できなかったコマンドがあるという目印とする為である。

また、アクセント及び特殊記号は個々の辞書テーブルを作成し変換に対応した。特殊記号は special_table(ll.256-285)、アクセントは accent_table(ll.288-375) である。特にアクセントの辞書テーブルは対応する HTML の表記が省略できない物なので多少行数のかかる物となってしまった。

プログラムの処理に関しては、主要 3 コマンドの処理よりも先に処理をし、主要 3 コマンドの search にヒットしないようにした。

より具体的に処理について説明すると、l.502 の re_accent 及び l.505 の re_accent2 はアクセントを search しヒットしたら do_accent(ll.423-426) へ送り処理する。同様に re_special_char(l.508)、re_special_letter(l.511)、re_special_sign(l.514) は特殊記号を search し、do_special_char 及び do_special_sign に送り処理をするのである。

(6) 脚注の処理

今回この変換ツールを製作する上で特に工夫したのが脚注の処理と後述する`\verb` の処理である。

LaTeX における脚注の処理とは、文中で`\footnote{...}`というコマンドを使用することにより。ページ下の欄外に表示されるというものである。

この LaTeX 独自の処理をどのようにして HTML に移植するかといろいろ試行錯誤した結果、Python のリスト機能を応用し処理を行うことにした。

実際に記述した処理は以下の通りである。

```
0138: def html_footnote(param):
0139:     global foot_num
```

```
0140:     foot_num += 1
0141:     footnote.append('<span class="footnotesize">
*%d.%s</span><br><br>' % (foot_num, param))
0142:     return '(*%d)' % (foot_num)
```

これは、文章中に\footnote{...}が来る度に line38 で設定した foot_num に 1 を足し、次に l.141 の命令で l.39 に作った footnote = [] というリストの中に footnote の中身の文字列を格納していき、最後に\footnote{...}があった部分に l.142 の (*%d)(%d の部分は foot_num の数字を代入) という文字列を出力するのである。

また、格納された footnote の中身の文字列であるが、これは文章の末尾の\end{document} を変換する際に foot_num の数字が 0 以外であるならば格納された文字列を出力するという処理を行っている (ll.479-488)。

(7) verb の処理

\verb|...|は記述した内容をそのまま表示するというコマンドである。これを Python で処理しようとすると、LaTeX のように|...|の中身は無視せずに処理を行ってしまうので、\verb 独自の処理が必要となってくるのである。また\verb は文字列を||で囲むのであるが、囲む||は他の記号でも良い。しかし、今回の変換ツールでは\verb を使用する場合は||を使用するという制限を付けることとした。

この verb の処理も Python のリスト機能を応用した。

```
0409: def do_verb(m):
0410:     global verb_num
```

```
0411:     parameter = m.group(1)
0412:     verb_num += 1
0413:     verb_table.append('%s' % (parameter))
0414:     return '[verb_%d]' % (verb_num)
0416: def do_verb2(m):
0417:     v_num = int(m.group(1))
0418:     return '%s' % (verb_table[v_num])
```

この処理はまず l.493 で他のコマンドよりも先に検索し抽出して一時的に処理するのである。具体的に説明すると、まず l.412 の処理で l.38 で作成した verb_num に 1 を足し、次に l.413 でリストに verb の中身を格納し、l.414 で verb のあった場所に [verb_%d](%d は verb_num を代入) を出力するのである。ちなみに、l.38 の verb_num の最初の数値は-1 であるが、これは文字列がリストに格納される際、付加される数字が 0 から始まっているということに対応した処置である。

この一時的に処理した verb は他の処理を全て終わらせた後に再び l.532 で search し、抽出した所で l.416 以降の処理を行っていくのであるが、ここで注目すべき点は l.417 の処理である。l.417 の m.group(1) とは l.12 の正規表現

```
0012: re_verb2 = re.compile(r'\[verb_([0-9]+)\]')
```

の ([0-9]+) のことでありここでは整数のみを抽出するのであるが、そのままそれを v_num として l.418 で使用しようとすると v_num が整数と判断されない為にエラーが起こってしまうのである。したがって、この部分で数字を整数化する命令である int を使用して v_num を整数化し、リストを機能させることとしたのである。

(8) 表の処理

表の処理であるが、この部分は私の技術不足な点から完全な対応はできす、シンプルな表にのみ対応するという制限を付け対応することとした。シンプルな表とは、

```
\begin{tabular}{lrrr} \hline
 \textgt{品名} & 単価（円） & 個数 & 購入 \\ \hline
  りんご & 100 &  5 &  3 \\
  みかん &  50 & 10 &  10 \\ \hline
\end{tabular}
```

このような書式で書かれた物であり、またこれに加えセルを統合する \multicolumn という命令に関しても制限(7)はあるが一部対応した。

表に関する具体的な処理方法であるが、これはまずプログラムのメインループ部分 (ll.449-535) で\begin{tabular}を探し (l.476)、\begin{tabular}があれば<table border="1">を出力し処理を tabular モードに切り替える。tabular モードでは\end{tabular}が来るまで関数 html_table(ll.144-156) の処理を行い、\end{tabular}来たら</table>を出力し処理を document モードに切り替えるという処理である。

また、処理の主な部分である関数 html_table 部分の処理であるが、この部分ではまず\multicolumn が行中に無いか検索し、もしあれば line の行頭に<tr>を付け、無ければ<tr><td>を付けるという処理を行っている。(ll.145-148)

次に\\を検索し、\\以下を</td></tr>に変換処理を行い、続けて \multicolumn を検索し、あれば対応する HTML タグへ変換処理を行っている。(ll.149-152)

最後に行中の&を検索し</td><td>に変換処理を行い、以上の変換処理を終えた line を関数 command_loop(ll.164-191) に送り処理を行ってから出力するという仕組となっているのである。

ちなみに関数 command_loop 部分 (ll.164-191) はメインループ部分から表の中で使用される可能性のあるコマンドのみを処理できるように変数として別に宣言したものである。

(9) スタイルシートの設定

今回の変換ツールでは変換後の HTML 文章をより LaTeX に近付ける為にスタイルシートを使用している。このスタイルシートは変換ツール使用者が自由に変更することができるのであるが、標準設定をいくつか設けた(8)。具体的に設定したのはフォント関連や文字のレイアウト関連の項目である。

(10) 変換ツールのテスト及び公開

変換ツールがある程度形となってきた段階で様々な文系論文でテストを行った。テストを行う際には特に正確に変換できているかを重視し、その結果によってスタイルシートやプログラム自体をより LaTeX に忠実な表現となるように調整を行った。

また、この調整が終了した段階で変換ツールの一応の完成とし、次に変換ツールを Web で公開する作業を行った。まず変換ツールに同梱するマニュアル(9)の作成を行い、それを変換ツール、CSS ファイルと共に

zip ファイルで圧縮し Web に掲載しておくこととした。

次に公開する Web の内容であるが、変換ツールを公開するのみであるので基本的にシンプルなレイアウトにし、コンテンツとしてマニュアルを Web 上にも掲載しオンラインマニュアル(10)として公開することとした。また、この段階で変換ツールの名前を便宜上『TeX to HTML』とすることとした。

次に公開する為のサーバーであるが、これは無料レンタルサーバーである infoseek isweb(11)のサーバーを借りて公開することとした。そうして Web 公開作業の行ったサイトが資料編 6 の図 5 であり、サイト URL は http://tex2html.hp.infoseek.co.jp/(2004 年 12 月 10 日更新)である。

4　総括

(1) 変換ツールの評価

一応の完成とした変換ツールであるが、この変換ツールを評価する為に当初の目標として提示した順序と照らし合わせて評価することとする。

まず、プレアンブル部分の処理であるが、この部分の変換に関しては基本的な部分には対応したのであるが、個人で設定するパッケージ等のコマンドに関しては補える部分には限りがあると判断し対応を断念した。また、プレアンブル部分に記述するタイトル等の項目であるが、これも日付部分で通常は使用できる`\today` という作成した日付を自動で記述してくれるコマンドの対応についても技術不足な部分から未対応とした。

次に本文の部分に関する処理であるが、これについてもプログラムの処理上、多少の制限(12)を付加する形となった。この制限についてはプログラムの構成上不可避な部分もあり、加えてそれを解消する技術を私自身が伴っていなかった為に行った処置である。この処置は使用者に対して LaTeX 文章の校正をさせるという労力を発生させてしまう為、今回製作した変換ツールの最大のデメリットといえる。

また、変換対応をした個々のコマンドを見ていくと、まず今回変換対応する上で設計段階から複雑な処理になるであろうと予測した`\section` 及び`\verb` の処理についてであるが、この二つのコマンドについては特に処理方法に関して工夫したので HTML に変換した時に遜色の無い変換ができる物となったといえる。同様に font 関係のコマンドに関してもスタイルシートを利用して LaTeX 文章に忠実な font の装飾を再現することに成功した。次に、特殊文字に関しては HTML で表現できる物に限りがあった為、完全な対応とはいかなかったが表現できる物に関しては対応した。

表についてであるが、これに関してはコマンドが複雑であった為、部分的な対応に留まる結果となった。表の処理に関してだけは使用できるレベルではなく実用的であるとは言いがたい処理となった。

最後に変換ツール全体についての評価であるが、この変換ツールは当初の目的通り LaTeX で作成された文系論文を規格に準拠した HTML への変換するという部分において目的を達成しているといえる。しかし、その変換をする為に LaTeX 文章を校正しなければならない点や、まだまだ未対応である部分も多いことから、使用対象者が日常的に使用できるレベルではなく、実用する為には更に改良を加える必要があるといえる。

(2) 今後の課題

今後、製作した変換ツールの方向性としてはより実用的な物へ改良を行うことがまず考えられる。

実際に変換ツールを今後実用レベルの物にする為にどのような改良を加えればよいのかであるが、その為にはまず今回技術不足の為に苦肉の策として設けた制限を取り払い使用者が LaTeX 文章の校正を行うことなく変換処理ができるようにするということである。現段階で考えられる方法としてはプログラムの構成を変更するという方法や、もしくは自動で LaTeX 文章の校正を行う別プログラムを製作し、それを使用することによって使用者の労力を減らすという方法が考えられる。

次に今回の変換ツールでは対応しきれなかった部分に対しての対応である。具体的に述べると表の部分やプレアンブル部分のパッケージ等に関してであり、特にパッケージに関しては WEB 上にも多数存在する為、物理的にも完全対応というのは無理であるが、有名なパッケージについては対応するという形でより変換の幅ができ実用度が増すことに繋がると考えられる。

また、これは別プロジェクトとなってしまうのであるが、Python 以外の言語で製作するということも一つ方法として考えられる。例えば JAVA を使用し、主に Windows を使用しているユーザーを対象としたツールを製作するということなどが考えられる。

最後に、どの言語で製作する際にも言えることなのだがプログラムを書く際にソースコードに詳細なコメントを付けることが重要であると感じた。今回の変換ツールの製作の際にはあまり詳細なコメントを付けていなかったのだが、これは後にこのツールを改良して使用するユーザー

にとっては不親切なことであると後々になって気付いたからである。

(3) まとめ

最後にプロジェクト全体の総括であるが、今回の変換ツールにおける製作の過程及び製作したプログラムは当初の目的であった「環境の構築が容易で、かつ外部プログラム等を多用しない変換ツールの製作」の部分に関して、製作した変換ツール自体は実用レベルとまでは至らなかったが、「ユーザーの利便性を重視する」という方向性の提示の役割は果たせたといえる。

また、変換ツール自体に対してもそのような方向性の実例として実証化したという点で十分な成果を上げられたと考えている。

今後、この変換ツールを実用レベルまで改良し、それを対象とするユーザーに使用してもらうことこそが、今回提示した方向性をより鮮明化し更なる利便性の追求に繋がると考えている。

文献表

奥村晴彦
2004 『[改訂第 3 版] LaTeX 2_ε 美文書作成入門』技術評論社
アラン・ゴールド 著、松葉 素子 訳
2001 『Python で学ぶプログラム作法』株式会社ビアソン・エデュケーション

注

（1）『[改訂第 3 版] LaTeX 2_ε 美文書作成入門』P1-2
（2）http://www.latex2html.org/(2001 年 9 月 25 日更新)
資料編 6 図 1 参照
（3）http://hutchinson.belmont.ma.us/tth/(2003 年 3 月 17 日更新)
資料編 6 図 2 参照
（4）http://www.cse.ohio-state.edu/ gurari/TeX4ht/mn.html
(2004 年 2 月 11 日更新) 資料編 6 図 3 参照
（5）http://pauillac.inria.fr/ maranget/hevea/
(2003 年 11 月 7 日更新) 資料編 6 図 4 参照
（6）資料編 7 相関表参照
（7）資料編 3 マニュアル参照
（8）資料編 2 スタイルシートの項参照
（9）資料編 3 マニュアル参照
（10）資料編 6 図 6 参照
（11）http://isweb.www.infoseek.co.jp/(2005 年 1 月 9 日更新)
（12）資料編 3 マニュアル参照