TEX の HTML 変換ソフトについての考察

林　恭子

目　次

——TEX の HTML 変換ソフトについての考察——

1　はじめに

(1) 制作の経緯と現状

福田ゼミでは卒業論文を TEX という組版ソフトで作成する。卒業論文提出後は、出力結果を PDF ファイルに変換し、インターネット上で公開するのが通例となっている。

通常、TEX のファイルは DVI というファイル形式で出力される。しかし、そのままの形式ではインターネット上に公開することが出来ない。そのため、公開可能なファイル形式に変換するという一手間を加える必要がある。そして現在、TEX で書いた文書をインターネット上で公開するためには、DVI ファイルを PDF ファイル、もしくは HTML 文書に変換する方法が一般的となっている。

特に PDF ファイルに変換する方法は、TEX の出力結果を忠実に表現してくれる。しかも、dvipdfm のソフトがインストールしてあれば簡単に変換することが出来る。しかし、PDF ファイルを閲覧するためには、Adobe 社の Acrobat Reader が閲覧者のコンピュータにインストールされていることが絶対条件となる。また、閲覧ごとに Web ブラウザから Acrobat Reader を起動する必要があり、PDF ファイルの読み込みに多少の時間がかかる。他にも、ブラウザがフリーズするなどの現象が起こる場合もあるため、一部のユーザーには敬遠されている(1)。

では、HTML ファイルに変換する方法はどうだろうか。

HTML は、インターネット上で最も一般的な表現方法である。昨今ではスタイルシートの充実により、デザインの凝った表現が出来るようになってきた。また、HTML ファイルは、PDF ファイルよりファイルサイズが小さく済む。しかし、未だに表や数式等に関しては不得手な部分も多い。元々 TEX は数学などの学術書用の組版ソフトとして開発さ

れたものである。大半が文章というシンプルな TEX ファイルであれば HTML でも十分に対応できるだろう。しかし、複雑な数式を使用するという場合、それらを HTML で忠実に再現するというのは、現時点ではほぼ不可能であり、何らかの対応策を考える必要がある。

つまり、PDF ファイルに変換する方法、HTML ファイルに変換する方法、どちらの方法にも一長一短が存在する。そのため、変換する TEX 文書によって適宜使い分けるとよい。

福田ゼミの卒業論文の場合、大半は文章であるので、本来ならば HTML 文書でも十分に対応できる。そもそも、TEX コマンドと HTML タグには共通点が存在する。例えば、TEX では、本文は `\begin{document}`と`\end{document}`の間に書く。対して、HTML では <body> と </body> の間に書く。他にも、`\documentclass` と <html> や、`\section` と h1 タグなどが挙げられる。このように、似たような機能及び記述方法を持つものが多いのである。このことから、TEX ファイルを HTML ファイルに変換することは、決して不可能ではない。しかし、一度 TEX 文書として仕上げたものを一から HTML 文書に書き換えていくのは大変な時間と労力が必要となる。さらに、必然的に HTML やスタイルシートの知識も必要になってくる。これでは、いくら TEX と HTML に類似点が多いとしても、HTML は全く知らない、というユーザーには高いハードルとなることは必至である。となると、シンプルな TEX 文書であっても、HTML ファイルにではなく、PDF ファイルに変換してしまったほうが良いと考えるかもしれない。ならば、PDF ファイルに変換するような手軽さで、自動で TEX のファイルを HTML ファイルに変換してくれるソフトがあれば、自分の TEX ファイルをインターネット上に公開することが、より容易に行えるよう

になるのではないだろうか。

そこで、まったく HTML 関連の知識を有していないユーザーでも、手軽に TEX ファイルを HTML ファイルに自動変換できるプログラムについて考察してみたい。

そもそも、そのようなプログラムは実在しないのだろうか。

調べてみたところ、LATEX2HTML[(2)]や TEX4th[(3)]など、TEX ファイルを HTML ファイルに自動変換するソフトウェアの存在がいくつか確認できた。そして、それら存在するプログラムの大半が英語圏のプログラムであった。TEX 自体が英語圏で生まれたものであるから、当然のことではある。もちろんのこと、これらは日本語を考慮して設計されていない。

その中でも、LATEX2HTML は、日本でも支持されているフリーソフトウェアである。特徴のひとつとして、HTML で表現が難しい数式等の複雑な記述は、画像に変換するという手法をとっている。このことにより、HTML で数式を表現することを可能にするのである。動作環境は元々 UNIX OS で開発されていたことから、Perl がインストールされていれば、大体どの環境でも動作するとある。しかも、新潟工科大学の竹野茂治氏により LATEX 2_ε 対応の日本語パッチが開発されており、日本語の環境にも対応させることが出来る。また、現在もバージョンアップが行われており、これがあればどんな TEX の文書をインターネット上に公開するのも難しくはないだろう。

そこで、早速試してみた。

LATEX2HTML を使用するためには、通常の TEX を実行する環境に dvips、Ghostscript、netpbm、nkf などのソフトウェアが必要となる。もちろん日本語に対応させたいので、日本語パッチもインストールす

る。インストールは、基本的にコマンドライン入力方式で行う。もちろんのこと、マウスが使える環境ではない。そのため、多少の UNIX OS 等のコンピュータの知識及び経験が無いと難しいだろう。因みに、インストール方法は、『LATEXWeb コンパニオン』と Google にて検索した Web サイト(4)等を参考とした。

しかし、何故か上手くインストールが実行されない。指示された通りにインストールを試みるのだが、思うような結果が得られないのである。簡単に言うとエラーが出て停止する。因みに、今回試みた PC は、大谷大学内に設置されている Mac OS X である。そのため、ハードもしくはセキュリティ関連の問題である可能性も考えられる。が、自分は Mac の専門家ではないため、結局原因は分からなかった。

また、簡単な TEX 文書用の変換プログラム(5)も試してみた。そちらは文系論文作成者支援を目的に作られており、確かに数式等には対応していない。また、テーブルにも未対応(6)であった。が、もともと日本語対応として作られているため、Python がインストールされていればコマンド入力 1 行で実行できるという手軽さがある。他にも市販品で存在するようではあるが、残念ながらそちらは試せていない。

これらの変換ソフトにはいくつかの問題点が存在する。

ひとつは、インストールや実行にコマンド入力を求めるものが多いということである。

現在、一般的に高い知名度と高い普及率を誇るのが Windows OS である。アイコンやマウス入力などで簡単に操作できるため、`ls -la` や `cd ../`のようなコマンド入力に馴染みのない人も多い。TEX もコマンドを使用せずに簡単にインストール出来る(7)ようになってきており、専門的な知識がなくても TEX を扱うことが出来るようになってきている。

しかしこれら変換ソフトでは、コマンド入力によるインストールや Perl のパスを設定するなど、多少の専門的知識が必要となる。TEX もコマンド入力でコンパイル等を行う。しかし、コマンド入力初心者であれば、インストール等に関して誰か知識や経験のある協力者が必要となるかもしれない。

もうひとつは、複数のソフトを使用する点である。

例えば、LATEX2HTML では、Perl、LATEX はもちろんのこと、dvips、Ghostscript、netpbm、nkf、更に LATEX2HTML の日本語パッチと必要に応じてインストールしなければならない。が、数式を使用しない簡単な TEX 文書であればこれほどのソフトウェアは必要とならないだろう。また、LATEX2HTML や TEX4th では、より細かく高度な設定が行え、かなりの高機能である。確かに LATEX2HTML は、大は小を兼ねるというソフトウェアである。しかし、少なくとも、文系ユーザーにとっては持て余すほどの機能がついていると言える。

また、基本的に TEX ファイルを HTML ファイルに変換するプログラムは英語圏のものが多いことも問題の一つである。

もちろん、英語に堪能なユーザーならば、なんら問題はない。が、そうでないユーザーは、英語で書かれたマニュアルと格闘し、日本語で解説されたものを探すという努力が必要になる。どうしても HTML 形式でインターネット上に公開する必要があると言うことでなければ、PDF ファイルに変換する方法を選んだほうが時間と労力の短縮となる。

これらのことから、「手軽に TEX ファイルを HTML ファイルに自動変換できるプログラム」に合致するプログラムとして、一番期待できそうなものは、文系論文作成者支援用の簡単な TEX 文書に対応している変換プログラムだが、何分対応しているコマンドがもう一押し欲しい

ところである。また、公開からあまり更新された形跡がない。過去にグレードアップがなされた形跡もないので、今後もプログラムが拡張される可能性は少ないと見られる。以上のことから、「手軽に TEX ファイルを HTML ファイルに自動変換できるプログラム」の制作する価値は十分にあるという結論に至った。

2　プログラムの設計

このプログラムの目標は「手軽に TEX ファイルを HTML ファイルに自動変換できるプログラム」とする。

(1) 制作において考慮すべき点

ここであらかじめ想定される前提条件が存在する。

1つ目は、数式等複雑な表記には対応しない、ということである。

TEX は、上で述べたように、元々は数学などの学術書用の組版ソフトとして開発されたものである。そのため、優れた数式記述能力を有しており、現在では、科学技術者を中心に多くのユーザーに支持されている。そのため、TEX の出力結果を忠実に再現しようとするならば、LATEX2HTML のように、HTML では表現が難しい記述を画像ファイルとして書き出すことは、確かに有効な手段である。しかし、現在では、簡単に DVI ファイルを PDF ファイルに変換することが出来る。故に、あくまで、今回対象とする TEX ファイルは数式や複雑な表記のないものとする。そのため、対象ユーザーは数式等を利用しない文系論文作成者を想定することとなる。

2つ目は、TEX と HTML で全く同じレイアウトを再現することは不可能である、ということである。

昨今では、スタイルシートの発展、普及により、HTML でもかなりレイアウト設定が行えるようになってきた。しかし、記述方式等に類似点が多くとも、互換性を全く考慮されていない両者では、全く同じものにすることはできない。また、HTML は各ブラウザでも表示が違うという性質も存在する。故に、数式等のコマンドでなくとも未対応、もしくは、可能な限りの再現に留まる場合が考えられる。

また、対応する TEX のバージョンは現在日本での最新バージョンである pLATEX 2_ε とし、これ以前のバージョンに関しては全く考慮しない。と、同時に、変換する TEX ファイルは正常にコンパイルできることを前提とする。そのため、TEX コマンドの誤記は考慮しない。

次に、プログラムの制作環境についてである。

今回のプログラムでは、OS は Mac OS X、表示確認を行う Web ブラウザは Safari を使用することとする。ここで Mac OS X を使用とするのは、今年度より福田ゼミで使用するコンピュータが Mac OS X になったためである。また、同様の理由で、基本とする Web ブラウザは第 1 位に Safari、第 2 位に Firefox とする。因みに、Internet Explorer での確認は、現時点では想定していない。このため、Internet Explorer で正常に表示されない場合も考えられる。

また、プログラム開発に使用するプログラミング言語は Python とする。これは、大学で学習し、文字処理に問題ない機能を有していると考えたためである。使用する Python のバージョンは現在の最新版である 2.5 とする。

(2) プログラムの概要

では、以上の前提を考慮した上で、制作するプログラムの概要について決定していく。

まず、変換対象とするコマンドであるが、基本的に『文書処理システム LATEX 2ε』の TEX コマンドとする。この対象に、数式及び HTML で表現不可能な TEX コマンドは含まない。また、TEX ではパッケージを読み込むことで TEX コマンドの拡張が行える。しかし、パッケージはユーザー自らが拡張することも出来、全てのパッケージに対応することは実質不可能である。よって、パッケージによる拡張は考慮しない。ただし、TEX 文書で使用頻度が高く、HTML でもサポートしているコマンドに関しては対応を検討する。さらに、福田ゼミでの卒業論文では EasyLayout という TEX パッケージを使用している。そのため、福田ゼミの卒業論文を変換するためには、EasyLayout への対応も考慮する必要がある。

次に対象ユーザーであるが、数式等の複雑な表現を使用しない文系論文及び文系論文作成者とし、コマンド入力で TEX をコンパイルできる程度のコンピュータ技能は有しているものとする。

また、制作技術は、Python を使用して実現することとする。TEX ファイルを読み込み、TEX コマンドを対応させる HTML タグに置換する。方法としては、文字列の単純置換と、正規表現によるパターンの置換を使い分ける。この際使用する文字コードは UTF-8 に統一する。これは、プログラム制作に Mac OS X を使用するためである。そのため、変換する TEX 文書の文字コードも UTF-8 であることが望ましい。

このプログラム制作のための資料は、LATEX 制作者のマニュアル書である『文書処理システム LATEX 2ε』、および人文情報学科の実践科目で

ある DTP 演習 1 のレジュメ[8]を参考とする。特に、DTP 演習 1 のレジュメは文系ユーザーでも使用頻度の高いコマンドが多いので、優先的に実装する。

制作の手順としては、まず対象とする TEX コマンドを資料からリストアップし、それらに対応する HTML タグの選定をする。その後は実際の制作を行い、完成後はゼミ生によるベータテストを依頼し、そのテスト結果を元に調整を行うこととする。

このプログラムの実行方法は、Mac ターミナルからのコマンドライン入力とし、動作の前提条件として、ユーザーのコンピュータに Python がインストールされていることとする。プログラムの動作確認に関しては Mac OS X、Windows XP もしくは Windows Vista の両 OS で行うこととする。

では、以上を踏まえて制作に取り掛かる。

3　制作

(1) 実際の制作過程

最初に、対象とする TEX コマンドを選定する。

選別方法は、基本として『文書処理システム LATEX 2_ε』から、補助として DTP 演習 1 のレジュメより、数式以外のものとした。その結果、約 250 個の TEX コマンドが変換対象の候補として挙げられた。更に、その候補からページスタイルなどの HTML 変換後に不要と思われる TEX コマンドや類似コマンドを取り除き、候補を約 160 個まで絞り込んだ。

次に、変換対象となった TEX のコマンドを類似形式ごとに分類する。

これにより、`\begin{引数}`と`\end{引数}`が対になっている環境コマ

ンド、\section{**引数**}のように引数を取るコマンド、\newpage のように引数をとらないコマンド、{\small **引数**}や\item 引数のように特殊な引数の取り方をするコマンドと、大きく 4 つに分類された。因みに、TEX のコマンドは \ の後に 1 個以上の英字と定義されている(9)。このため、現時点では\usepackage(10)や\newcommand(11)などによる日本語のコマンドは一切考慮していない。

では、選別した TEX コマンドに対応する HTML タグを決定する。

基本的に、環境コマンドなどの段落や広範囲を指定するものは などのブロック要素を、一部または段落を形成しないものには などのインライン要素を対応させるようにした。また、出来るだけ、デザイン部分はスタイルシートで指定するようにした。スタイルシートは、基本部分は外部スタイルシートに記述し、タグごとの設定は style 属性で個別に設定することとする。

見出しは、TEX では \part、\chapter、\section、\subsection、\subsubsection、\paragraph、\subparagraph の 7 種類が存在する。しかし、HTML では h1 タグから h6 タグまでの 6 種類である。そのため、クラス article には存在しない \chapter を除いた 6 つを採用することにする。このため、日本語環境でのクラス article である、クラス jsarticle に対応させることになる。

選別等が完了すれば、実際のプログラムの制作に移行する。

今回は、変換する TEX ファイルを全て読み込み、全文を一気に処理するという方法をとる。変換方法は、文字列による単純置換と正規表現によるパターンで置換との 2 種類の方法により、選別した TEX コマンドを、対応させる HTML タグに書き換えていく。昨今ではコンピュータの性能も向上しているため、ファイルを丸ごと読み込み、検索を行っ

ても、十分に短時間での処理が行える。また、全文を処理するため、途中に改行が入るような長い文書であっても、改行を超えて処理が行うことが可能である。

では TEX ファイルのデータを読み込み、置換作業を行う。

まず、コメントの % コマンド以降を HTML のコメント表記<!---->内に内包して置換し、verbatim 関連コマンドを pre タグに処理する（資料編 866 行–879 行)。同時に、TEX 内で意味を持つ \ や % を HTML での特殊記号の表記方法に従い¥ の形に変換する。これにより、コメント内や pre タグ内のものがコマンドとして該当するのを阻止する。次に、`\documentclass` を html タグとヘッダーに、document 環境を body タグに処理し、HTML 文書の形式を整える（資料編 894 行–896 行)。このときに`\title`、`\author`、`\date` を処理し、title タグに挿入し、`\maketitle` も処理するようにした(12)。HTML は、現在最新となる HTML4.01 を使用し、使用予定の HTML タグを鑑みて、バージョンは HTML4.01 Transitional DTD を使用することとする。以上、ここまでの変換作業で、HTML の基本的な形は出来上がっており、残っている部分は、主に本文とプリアンブルにあたる。

次は、引数を取らないコマンドや TEX で使用される特殊記号を HTML の特殊記号に変換する。ここでは、正規表現ではなく、多くを単純な文字列置換で対応できた（資料編 899 行目–929 行目)。引数を取らないために、パターンを使用する必要がないためである。次に、2 個以上の改行の処理や見出しの処理、書体・書式等の処理を行う（資料 931 行目-957 行目)。これは、比較的他のコマンドを内包しないと考えたためである。同様に、コマンドを内包しにくいものから処理を進めていく。環境に関しては、コマンドを内包しやすい。が、その形は \begin

が HTML の開始タグに、\end が HTML の終了タグに対応している。また、必ずペアで存在するので、単純に文字列置換でも対応できる。ただし、もっと厳密に行いたい場合は開始コマンドを内包しないように処理する必要がある。そのため、なるべく内包されるコマンドを先に処理し、環境コマンドの処理は処理後半に行った。

コマンドが一通り処理できた後は、余分な改行コマンド及びタグを処理する。通常、TEX の文書中での 2 個以上の改行、つまり一行以上の空行、は次の段落を意味する(13)。そのため、その部分を
に書き換える。本来ならば、段落なので p タグに書き換えるべきなのだが、明確に段落であると言えない場合もあるので、
で対処する。また、これを行うと<h1>や の前後に改行タグが入る場合がある。これは HTML で余り推奨されていないので、
を削除する処理を行う。また、このときに のような空白の段落も同様に処理する（資料編 182 行目–196 行目、821 行目–832 行目)。

変換作業の最後に、ヘッダー部分にコメント等を挿入することはあまり推奨されていない。そのため、プリアンブル部分をそのまま残しておくことはできないので、pre.txt というファイルに全て書き出すことにより対処した(14)。

(2) 問題点と解決策

特に問題となったものは、見出しと minipage 環境、そして tabular 環境であった。

見出しには、通常章や節の番号が振られる。そして、見出しの前に見出し番号が表示される。つまり、「1 章 4 節 例」ならば、「1.4 例」となるわけである。しかし、単純に処理を行った場合、各見出し毎に処理が

行われ、上位の見出しの番号を引き継ぐ形とならない。また、章ごと、節ごとに下位の見出し番号は 1 から始まるべきであるが、全てが通し番号となる。

このため、各見出しの終わりに目印となるコメントを挿入することで、各見出しの領域を指定し、上位の見出しの領域から下位の見出しの順に処理するようにした。つまり、とある\section の領域を選択し、次にその\section 内の\subsection を処理し、さらに、\subsection 内の\subsubsection の順に処理する。このようにして、下位の見出しがなくなれば次の上位の見出しである\subsection、\section の処理に移り、以降は同様となる訳である（資料編 448 行目–521 行目）。これにより、見出し番号が適切に引き継がれ、「1.4.3 見出し」のような表示が可能となった。この際、処理が完了した後、挿入したコメントを削除することを忘れてはならない。

また、minipage 環境であるが、この最大の問題は minipage 環境を使用した 2 段組であった（資料編 524 行目–570 行目）。

まず、正規表現のパターンを\\begin\{minipage\}\{0?\.([0-9]+?)\\textwidth\}(.+?)\\end\{minipage\}\n<!---->\n\\begin\{minipage\}\{0?\.([0-9]+?)\\textwidth\}(.+?)\\end\{minipage\}と書いた（資料編 153 行目）。しかし、これでは 2 段組を指定していない minipage 環境から 2 段組を行う minipage 環境までの広範囲が抽出される。これは、(.+?) の部分で次の\end{minipage}以降に<!---->が続くものまでを読み込むためである。対応のため、パターンに合致した部分に<!---->を含む場合はそのまま返し、次に前後に<!---->をとらない minipage 環境を変換するようにした。しかし、それではうまくいかなかったため、一致部分に、<!---->を含み、かつ

2 段組をしない minipage 環境がある場合、その部分を通常の minipage 関数の処理した後、2 段組の処理は行わずに return することにした (資料編 545 行目–554 行目)。他の部分にも共通するが、正規表現は一度正規表現のパターンに合致するとして読み込んだ部分は、重ねて読み込まず、次の行からパターンを検索する。このため、一度行った処理でも、漏れが発生する場合があり、同じ処理を複数回行う必要がある。そのため、今回は 2 段組の処理を 2 回、通常の minipage 関数の処理を 1 回行って対処した。

TEX の tabular 環境は、HTML でいうテーブルタグに相当する。しかし、その表記はテーブルタグとは異なっている。

特に罫線の引き方が特殊であり、| が縦線、\hline と \cline が横線に相当する。このため、カスタマイズ性は高いが、HTML での設定はたやすくない。基本的に \\ は tr タグに、& は td タグに対応させることにして、罫線は table タグにではなく、それらの tr と td タグ全てにクラス名を振り、タグに対応するクラス名にスタイルシートを指定することとした。この場合、タグごとにスタイルシートを設定することになるため、スタイルシートの部分だけでもかなりの文字数となる。そのため、スタイルシート自体は別にテーブル用に外部スタイルシートを用意し、そこに書き込むことにする。これにより、簡単な罫線の問題はクリアした。また、全てのタグにクラス名を付与するので、セルの縦連結および横連結の際に単純に処理を行うと番号がずれる場合があり、その点も対応した。これにより相当複雑でなければ tabular 環境に対応が可能となった（資料編 572 行目–788 行目)。

全体を通して、基本は最小一致での処理となった。また、他のコマンドを内包すると誤作動を起こす場合が多いので、なるべく正規表現を

最小一致の (.+?) から最小一致かつ他のコマンドを内包しないように ([^\{]+?) に書き換えて対応した。ただし、この場合、内包されるコマンドが処理されないと、多くの検索漏れが発生する可能性があり、全体の処理を複数回行う必要がある。そのため、今回はコマンドの処理を 5 回繰り返すことにより対応している。因みに、文系ユーザーが TEX の文書を記述するに当たって、5 つものコマンドを内包するような文書を書くことは滅多にない。そのため、5 回で十分に対応できるものとする。

4 評価

(1) ベータテストにより発見された問題点と対応状況

基本的な TEX コマンドの実装が完了した後、ゼミ生に対しベータテストを依頼した。ベータテストの協力者は 3 名、テスト条件は、OS は Mac OS X を使用し、実行方法はターミナルからのコマンド入力方式とした。Python は公式バージョン 2.5 を使用した。また、変換する TEX のファイルは、こちらからは指定せず、テスト協力者が各自任意のものを使用、実行した。

結果、変換自体にエラーが発生し、プログラムが強制終了するということはなかった。しかし、テスト協力者全員の、変換後の HTML ファイルのレイアウトが崩壊した。原因が分からなかったので、他の TEX ファイルでも試してもらったが、同じ結果となった。そこで、試しに自身でテストに使用していた TEX ファイルを配布、実行してもらう。と、そちらは期待した結果が得られた。このことから、問題はプログラム自身にあると推測された。しかし、それだけでは、何故違う結果となったのかは判明しなかった。そのため、原因究明を目的にテスト協力者からサンプルの提供を受けた。比較のために、他のゼミ生からもサンプルの

提供を受け、原因究明を行った。

その結果、HTML のレイアウト崩壊した原因のひとつは、TEX ファイルの改行コードにあったことが判明した。これは、自身がサンプルとしていた TEX ファイルは Windows OS の改行コード CRLF が使用されていたのに対して、テスト協力者が使用した TEX ファイルは Mac OS X の改行コード CR が使用されていたため発生したものであった。本プログラムでは、改行コードは LF を前提として処理を行っていたため、Mac OS X で作成された TEX 文書の改行が認識されず、ファイル全体を 1 行として処理された結果であった。そこで、TEX ファイルを読み込む際に、open コマンドのモードを rU と指定することによって対応した（資料編 18 行目）。これにより、全てのファイルは、改行コードを LF に統一されて読み込まれることとなる。

もう一つの原因は、`\title` コマンド内で他のコマンドを内包することを許可していたためであった。これは改行コードを統一したことにより解決されたのだが、再発防止のために、入れ子式になっているコマンドは内側を優先的に処理するようにプログラムを変更した。

この他にも、テスト協力者及びサンプル提供者により TEX コマンドの記述方法が異なる場合がある。その結果、プログラムが誤作動する現象が見られた。

その中で、失念していたものが、`\TeX`、`\TeX{}`、`{\TeX}`など、引数を取らないコマンドの複数ある表記方法であった。ベータテストの時点では`\TeX` にのみ対応させていたので、`\TeX{}`、`{\TeX}`の表記では {} が残ってしまうこととなった。この場合、ユーザーがどの形式を利用するか定かではない。また、表記方法には限りがあるため、可能な限り対応するようにプログラムを書き換えることとした。

また、宣言に関して問題が発覚した。

TEX における宣言は、引数を取らない。正確に言うならば、必ず引数が {} に入っている他のコマンドとは違い、宣言された場所から、宣言前に記述された { もしくは\begin に対応する } や\end の直前までに適応される。引数部分が明確にならないため、正規表現が難しくなる。

誤作動を起こした原因のひとつは、宣言を他のコマンドと混同して処理してしまったことである。たとえば、{\small **文字**}のような表記があるとする。これは、正確には文字サイズの宣言であるので\small のみで以降に適応される。{} は、つけることに適応範囲を明確にする働きがある。しかし、今回のプログラムで想定していたのは{\small **文字**}のような表記のみで、\small 文字と記述した際に誤作動することになる。また、他にも\small{**文字**}にも適応されることもあり、正規表現での特定が難しい。また、{\small **文字**}の表記は、宣言とコマンドの引数と二通りの捉え方が考えられる。特に、文字サイズなど、他のコマンドに内包される可能性が高いものは、変換処理の順序により結果が異なることとなる。これの問題に関しては、残念ながら対応策が発見できていないため、未解決となっている。文字サイズの宣言に関しては{\small **文字**}の表記で統一しているので、差し当たり、ユーザーに対応する形式で記述してもらうしかない。

その他、改行の有無やコメントの挿入により、コマンドが正規表現に合致しない場合が確認された。

例えば、コマンド\newpage である。コマンドの前後に空行がなくても正常にコンパイルできる。しかし、ベータテストの時点では、正規表現に改行を含めて処理を行っていた。そのため、改行がないものは検索及び置換漏れが発生した。この場合は、誤差として、正規表現から改行

を除き、連続する改行の処理と段落タグ前後の
 処理の部分で対応させることにした。

今回のベータテストでは思わぬ結果となり、残念だった。しかし、全く予期しなかった問題を発見、解決につながり、他者によるテストの重要性を改めて感じることとなった。

(2) 自己評価

「手軽に TEX ファイルを HTML ファイルに自動変換できるプログラム」という目標に関して言うならば、かなり手軽なものが出来たと考える。必要なものは、TEX ファイルと Python とこのプログラム一式である。Python がインストールされていれば、他にインストールすべきものはない。

プログラムソースは、TEX ファイルの読み込みなど事前処理を最初に書き、その後正規表現、変数の用意、関数、実行処理、書き出し、という構成になっている。制作当初は、コマンドごとに使用する正規表現、変数、関数をまとめていた。しかし、これは単体の処理だけを見ると分かりやすいが、変数名や処理が重複しやすかった。そのため、それぞれをまとめ、順序も可能な限り処理順序で並べることにした。また、コメントで大きく見出しをつけるなどして、自分はもちろん、他者にも分かりやすくなるよう配慮した。

しかし、当初対応を予定していた TEX コマンド約 160 個に対し、実際対応できたのは約 90 個程度と 55% 程度に留まった。また、TEX のオプション引数やユーザーによる拡張性など、TEX の性能が十分に反映されたものとは言えない。

今回の達成率の原因は、大きく分けると 2 つである。

ひとつは、技術的問題である。

これは、TEX コマンドの選定段階では実現可能としていたが、実際には HTML での表現が不可能であったものと、制作者の力不足のためのものがある。全体を通して勉強不足であった。TEX に関しては、TEX コマンドを全て HTML タグに変換することは不可能であると認識していたにも関わらず、必要であるもの、必要でないものの選別が不十分であった。中には、使用経験のないものも多くあり、使用法の勉強やテスト環境の用意に思わぬロスが発生した。また、Python に関しては、処理順序の問題や、正規表現のコンピュータの正確さに、予期せぬ結果が続発した。環境 minipage の 2 段組の処理や見出し番号の処理、内包するコマンドの処理など、その打開のためにかなりの時間を消費することとなった。HTML やスタイルシートも例外ではない。現在、デザインは HTML 要素からスタイルシートでの指定に移行してきている。そして、スタイルシートは今まで以上のデザインを提供している。しかし、`\hfill` や`\hrulefill` など、当初 HTML で実現可能であると考えていたものも、Web ブラウザがスタイルシートのプロパティに未対応であったり、HTML タグがブロック要素であったり、と結局実装はかなわなかった。また、HTML タグやスタイルシートプロパティは全て把握していたわけではなかった。そのため、実装初期段階では <pre> を使用し、のちに一部を <code> に変更するなどの仕様変更を幾度となく行うこととなった。

もうひとつの原因は、自身の立てた見通しの甘さ問題である。

大まかに計画を立てたが、余りにも大まか過ぎであった。時間配分と進行状況の確認を密に行わなかったため、修正がおろそかになり、計画の大幅な遅れが発生した。仕様書も厳密なものを作成しておらず、プ

ログラムの頻繁な書き換えや仕様変更に対応する際に歪みが発生した。その修正にも時間を食い、結果として他のコマンドが疎かとなってしまった。

当初、TEX と HMTL に類似点が多いために、変換自体に問題は起きないだろうと考えていた。しかし、TEX のコマンドを HTML で再現できない部分も多く、改めて TEX の機能の高さを感じさせられる結果となった。

(3) 今後の課題

結果として、福田ゼミで使用する EasyLayout への対応は見送り、約 70 個の TEX コマンドに対応できなかった。ただし、基本的な文書構造には対応できているかと思う。つまり、簡単な TEX 文書が変換できるレベルとなる。

今後の課題としては、まず、EasyLayout への対応が最優先となる。DTP 演習 1 で習得した TEX コマンドはほぼ対応済みである。このことから、EasyLayout への対応により、福田ゼミでの卒業論文の変換が可能となるのである。その後は、残りの TEX コマンドの再検討とオプションへの対応へと移行する。最終的には、インターネット上で CGI として公開し、ユーザーが Python をインストールする必要なく変換できるようになればと考えている。

元々は、卒業論文の研究用に制作を開始したプログラムである。しかし、現状と HTML 変換の可能性を見て、今後も制作を進めて行きたい。

注

(1) 因みに、大学に設置されている Mac OS X で Web ブラウザの Safari から PDF ファイルを閲覧する場合、管理者の ID とパスワードが必要となることがある。

(2) http://www.latex2html.org/

(3) http://www.cse.ohio-state.edu/ gurari/TeX4ht/mn.html

(4) http://radar.sci.hokudai.ac.jp/ kato/computer/TeX/l2h/

(5) http://www2.otani.ac.jp/ yfukuda/sotsuron2004/ nishioshogo/my_site/index2.html

(6) 同様のプログラムで簡易なテーブルに対応しているものも存在する。
http://www2.otani.ac.jp/ yfukuda/sotsuron2004/ ZushiHiroaki/tex2html/index.html

(7) http://www.ms.u-tokyo.ac.jp/ abenori/index.html

(8) http://tibet.que.ne.jp/otani/lecture.html

(9) 『文書処理システム LATEX 2_ε』P.36 より引用

(10) スタイルファイルを読み込む TEX のコマンド。

(11) 新しくコマンドを定義する TEX のコマンド。クラス jsarticle では日本語コマンドの作成が可能である。

(12) この部分に関しては、ベータテスト以降に処理の最終部分での処理に変更した（資料編 1037 行目–1051 行目）。

(13) 必ずしもそうではない。
（例）改ページコマンド \newpage の前後などの空行。

(14) 他のコマンドを処理した後にこの処理を行うと、コメントや \newcommand の内容が HTML に変換されたものが保存され

る。そのため、ベータテスト以降は、処理の最初にプリアンブル部分をファイルに書き出し、最後に文書中から削除するという方法に変更した。

文献表

奥村晴彦

2004 『美文書作成入門』技術評論社

Michel Goossens, Sebastian Rahtz 著　鷺谷好輝訳

2001 『LATEXWeb コンパニオン-TEX と HTML/XML の統合-』アスキー

レスリーランポート著　阿瀬はる美訳

1999 『文書処理システム LATEX 2ε』ピアソン・エデュケーション

(株) アンク

2002 『ホームページ辞典 第 3 版』翔泳社