獣を馴らす準備(*)

第 1 部：btxhak.pdf を読むために

私立文系🔰初級ユーザー

2024 年 12 月 4 日

TEX & LATEX Advent Calendar 2024

構成についてと文体について

⚠ 2 部構成になっています ——

本ペーパーは、BIBTEX のドキュメントを読み解く助けとなることを目指すものですが、

- 第 1 部：btxhak.pdf を読むために
- 第 2 部：btxbst.doc を読むために

という 2 部構成になっていて、物理的にも、2 つの文書に分けてあります。こちらはその第 1 部のほうになります。

⚠「薮からスティック」的な文体になっています ——

私は私立文系出身で、簡単な英語であれば（頑張れば）なんとか読めないこともないのですけれど、数学やプログラミングには無縁なため、それらに関連する用語の日本語の定訳が分かりません。そこで、的外れな訳語を捻り出すよりも、英語のママのほうが間違いを減らせるのではないかなと思った結果、全体的に、「薮からスティック」とか「身を powder にする」みたいな文体になってしまいました。

最初に、少々長めの導入部があって、続けて、まずは簡単な実例を眺めてみます。それを踏まえてからようやく、BIBTEX が用意している COMMANDs と built-in functions の働きについて概観することにします。

なお、BIBTEX の基本的な使い方については、既にご存じのものと前提します。

1 まえおき的な何か

ここでは、余計な言い訳をちょっとして、それから、このペーパーで取り上げる基本ドキュメントについて説明した後、本論に入る前に簡単に用語の整理をしておきます。

1.1 ナマケモノの言い訳

実は、私自身はこれまで、BIBTEX を使ったことがありません。そもそも、

- 私ごときが目を通せる論文の数なんて高が知れていて、しかも、その中で使えるもの（＝引用に値するもの）はほんの一握りに過ぎませんので、文献をデータベースで管理しないことにはどうにも始末に負えない、なんてことはありません(1)。
- 私が論文を寄稿する雑誌や論文集の書誌の体裁は大体どれも同じなので、発表媒体ごとに書誌の書き方が千差万別で、データベースから自動で生成しないととてもじゃないけどやってられない、ということもありません（もちろん、同じ研究分野であっても、異なる書式を採用している雑誌がないこともないのですが、その別流派の雑誌に私が自ら進んで投稿することは恐らくありません）。
- 私の専門分野の場合、出典の情報は脚註や後註に入れる形式が一般的で、論文の稿末に引用文献の一覧を載せるように求められることもほとんどありません(2)。

そういうわけで、今後も私には BIBTEX は必要なさそうです。そんなナマケモノが、興味本位でまとめたペーパーですので、実用性は期待しないでください。あと、日本語の扱いについても、全く考慮していません。

1.2 BIBTEX の基本的ドキュメント

BIBTEX について理解するために読むべき基本的なドキュメントは、以下の 3 点かと思われます：

[1a] Oren Patashnik, *BIBTEXing*, February 8, 1988. (btxdoc.tex|pdf)

(*) 表題は、Markey, *infra* note (8) のタイトル“*Tame the BeaST*”を意識しています。日本語での表記に関しては、シェイクスピアの「じゃじゃ馬馴らし」（William Shakespeare, *The Taming of the Shrew*）とか、ロバート・B・パーカーの小説『海馬を馴らす』（Robert B. Parker, *Taming a Sea-Horse*）なんかが念頭にありました。
あと、このペーパーを読みさえすれば立ちどころに「獣を馴らす」ことができるようになりますよ、と大見えを切る自信はありませんので、馴らす“準備”としました。

(1) 有名な理系の某先生は、あるインタビュー記事で「朝起きたら論文をチェックするのが日課で、毎朝 100 本は論文をチェックします。…地方の学会に行くと、夜、研究仲間と飲みながら『あの論文にデータが出ていたよね』などとディスカッションする。『3 段落目のあの文章は読んだ?』『あれは仮説がすぎるよね?』など、皆で話す」とおっしゃっていました。また、昨年、ある院生の方が、毎朝英語の論文を 1 本読むことを日課にされて、読まれた論文について詳細な論文メモを作成のうえ毎日研究室 Slack に投稿されていて、3 年間で 1000 本以上の論文を読破されたとのことで、その方法論を公開なさっていました。こういう、何千本、何万本と論文を読まれて、何百本も御論文をご執筆なさるような超一流の研究者の皆さんの場合には、データベースを利用しての文献管理は必須なのでしょう。

(2) 実は一度だけ、ドイツの出版社から、文献一覧を付けるように要請されたことがありました。適当に作成して提出したところ、後日届いたゲラでは、先方が書誌を整形し直してくれてました。そういえば、Scribe のマニュアルにも、こんな風にあります：“Experienced authors tend to handle these nuisance variations in bibliography format rules by ignoring them and using the same format for all of the papers that they submit, passing to the journal editors the job of cleaning up the Bibliography. Inexperienced authors tend to spend days getting every last comma in the right place.” Unilogic, Ltd., *Scribe: Document Production System: User Manual*, 4th ed., 1984, Chap. 12. 私は全然“experienced author”なんかじゃないんですけれど。

[2a] Oren Patashnik, *Designing BibTeX Styles*, February 8, 1988. (btxhak.tex|pdf)

[3a] Howard Trickey and Oren Patashnik, *BibTeX 'plain' family*, 1984, 1985, 1988, 2010. (btxbst.doc)

これらはいずれも BibTeX と一緒に配布されています[3]。

[1a] と [2a] は本来一体のものとのことですが、[1a] が一般ユーザー向けなのに対して、[2a] が BibTeX のスタイルファイル作成者向けということで、物理的に 2 つの文書に分けられています。そのため、[1a] には最初の第 1 節〜第 4 節が含まれていて、他方 [2a] は最後の第 5 節のみから成っています。そして、[3a] は、plain.bst、unsrt.bst、alpha.bst、abbrv.bst という 4 つの standard styles の“documented source”です[4]。

本ペーパーでは、第 1 部で [2a] を、第 2 部で [3a] を扱うことにします[5]。

1.3 用語の確認と整理

本論に入る前に、BibTeX に関連する用語について、少しだけ整理をさせてください。

例えば、

```
@BOOK{srcibe,
   author = "Reid",
   title = "Document Production System"}
@BOOK{latex,
   author = "Lamport",
   title = "A Document Preparation System"}
```

という中味の sample-db.bib というファイルを用意したとします。そして、test.tex という文書の中に、

```
\nocite{srcibe,latex}
\bibliography{sample-db}
\bibliographstyle{plain}
```

と書いたとします。それで、これを LaTeX で処理した後に、BibTeX を実行すると、test.bbl が出来て、その中味は、

```
\begin{thebibliography}{1}
   \bibitem{latex} Lamport.
      \newblock {\em A Document Preparation System}.
   \bibitem{srcibe} Reid.
      \newblock {\em Document Production System}.
\end{thebibliography}
```

みたいな感じになっています（若干整形を施しています）。最後にもう 2 回ほど LaTeX を実行すると、ようやく、

Heading

[1] Lamport. *A Document Preparation System*.

[2] Reid. *Document Production System*.

という風に出力されるわけですよね。

それで、まずは、\bibliograpy というコマンドについてですが、このコマンドは、その引数“sample-db”を test.aux に書き出して、それから、BibTeX が生成した test.bbl を読み込んでいます。sample-db.bib は拡張子が“.bib”ですが、この書誌情報をまとめたファイルは「データベース」と呼ばれることが多いように思われます。データベースファイルの拡張子が“.bib”なのは少しだけ混乱するような気もしますが、それでも、\bibliography に指定するファイルの拡張子が“.bib”で、そして、\bibliography の場所に thebibliography 環境（test.bbl）が出力されることからすると、これはこれで合理的な命名なのでしょう。

\bibliographystyle のほうについては、その引数のファイルの拡張子が“.bst”となっていて、その中味も「スタイルファイル」なので、こちらは格別、直観には反しません。

次に、thebibliography 環境によって出力される「文献一覧」のことを一般に何と呼ぶのかについては、いろいろとあって困ります。上の例で“Heading”とした見出し文字列は、例えば、article.cls では“References”、book.cls だと“Bibliography”、jarticle.cls の場合は「参考文献」、そして jbook.cls では「関連図書」となっています[6]。

更には、この「文献一覧」に相当するものは、Lamport さんの LaTeX マニュアルでは“source list”、Patashnik さんのドキュメントだと“reference list”と呼ばれています。

最後に、.bib ファイル内の、個々の

```
@ARTICLE{. . . . .}
@BOOK{. . . . .}
```

等々や、thebibliography 環境内の

```
\bibitem[ ]{ } . . . . .
\bibitem[ ]{ } . . . . .
```

そして、最終的な Bibliography 内の

```
[1] . . . . .
[2] . . . . .
```

は、これらはいずれも、エントリ（entry）と称されています。

つまり、.bib ファイル内には、@ARTICLE、@BOOK 等々の書誌情報の「エントリ」が並べてあって、これらエントリが BibTeX によって .bst ファイル内の article や book といった function の定義に従って整形された上で .bbl ファイルに thebibliography 環境の「エントリ」\bibitem として出力されます。そして、LaTeX が thebibliography 環境をタイプセットすると、[1]、[2]、… 等のラベルが付いた Bibliography の「エントリ」となる、というわけです。

2 とりあえず、.bst ファイルの中を覗いてみる

手始めとして、BibTeX が用意している COMMANDs や functions の実際の使用例について、さらっと眺めてみましょう。

2.1 適当な例

まず、私がいい加減に作ったサンプルです。

次のようなファイル sample1.tex があるとします：

```
\documentclass{article}
\begin{document}
\nocite{cite-key:Foo,cite-key:Bar}
\bibliography{demo1}
\bibliographystyle{mysettings}
\end{document}
```

demo1.bib の中味は、こんな感じです：

---

[3] 日本語訳もあります：[1b] 松井正一訳「BibTeXing：BibTeX の使い方」（1991 年 1 月 1 日）(jbtxdoc.tex|pdf)；[2b] 松井正一訳「BibTeX スタイルの作り方」（1991 年 1 月 1 日）(jbtxhak.tex|pdf)；[3b] 松井 正一「JBibTeX 'jplain' family」（1994-10-25）(jbtxbst.doc)。私の手元の W32TEX [2020/07/19] にはいずれも含まれているのですが、TEX Live には入っていないのかも知れません。その場合には、CTAN に行って、[1b] と [2b] については pbibtex-manual パッケージ、[3b] については pbibtex-base パッケージを探してみてください。

[4] ちなみに、これら standard styles の“standard”の意味について、Patashnik さんご自身は次のようにおっしゃっています：“By the way, these are called “standard” styles not because they are supposed to be some sort of standard, but because they are in the standard release of BibTeX.” Oren Patashnik, *BibTeX yesterday, today, and tomorrow*, TUGboat 24-1 (2003), p. 27.

[5] [1a] の内容については、必要に応じて触れることにします。sort key として使う field の優先順位や、alpha.bst が label として使う field の優先順位、それに“special character”の説明などは、どれも [1a] でなされていますので、[1a] も必読です。

[6] LaTeX の外の世界に目を向けてみますと、MLA: “Works Cited”; APA: “References”; CSE: “Cited References”; The Chicago Manual of Style: “Bibliography” (Notes and Bibliography Style) or “References” (Author-Date Style) といった例が見られます。

```
@entry-type.degree{cite-key:Foo,
   field.1 = "Case Institute of Technology, 1960.",
   field.2 = "Case Institute of Technology, 1960.",
   field.3 = "California Institute of Technology, 1963."}

@entry-type.degree{cite-key:Bar,
   field.1 = "M. I. T., 1960.",
   field.2 = "Brandeis University, 1963.",
   field.3 = "Brandeis University, 1972."}
```

そして、適当に作ったものすごく単純な mysettings.bst は、次のような内容です：

```
ENTRY
   {field.1
    field.2
    field.3}
   {}
   {}

FUNCTION{entry-type.degree}
   {"\item[" type$ "]" * * write$ newline$
    "\item[" cite$ "]" * * write$ newline$
    "\item Bachelor -- " field.1 * write$ newline$
    "\item Master -- " field.2 * write$ newline$
    "\item Doctor -- " field.3 * write$ newline$}

FUNCTION{begin.output}
   {"\begin{itemize}" write$ newline$}

FUNCTION{end.output}
   {"\end{itemize}" write$ newline$}

READ

EXECUTE{begin.output}

ITERATE{call.type$}

EXECUTE{end.output}
```

このような３つのファイルを用意した上で、LaTeX で sample1.tex を処理するとまず、sample1.aux が出来ます：

```
\relax
\citation{cite-key:Foo}
\citation{cite-key:Bar}
\bibdata{demo1}
\bibstyle{mysettings}
```

この sample1.aux に対して BibTeX を実行すると、生成される sample1.bbl はこんな風になります：

```
\begin{itemize}
\item[entry-type.degree]
\item[cite-key:Foo]
\item Bachelor -- Case Institute of Technology, 1960.
\item Master -- Case Institute of Technology, 1960.
\item Doctor -- California Institute of Technology, 1963.
\item[entry-type.degree]
\item[cite-key:Bar]
\item Bachelor -- M. I. T., 1960.
\item Master -- Brandeis University, 1963.
\item Doctor -- Brandeis University, 1972.
\end{itemize}
```

それで最後にまた LaTeX で処理しますと、この sample1.bbl が読み込まれてタイプセットされて、以下のような結果が得られます：

entry-type.degree

cite-key:Foo

- Bachelor – Case Institute of Technology, 1960.
- Master – Case Institute of Technology, 1960.
- Doctor – California Institute of Technology, 1963.

entry-type.degree

cite-key:Bar

- Bachelor – M. I. T., 1960.
- Master – Brandeis University, 1963.
- Doctor – Brandeis University, 1972.

勘のいい方でしたら、以上の demo1.bib、mysettings.bst、sample1.aux、sample1.bbl を見比べてみるだけで、.bst ファイルの設定が何をやってるのかが、お分かりになるかも知れません（私だったら無理っぽいですけれど）。

比較のために、普通の場合のサンプルも見てみます。

2.2 一般的な例

sample2.tex です：

```
\documentclass{article}
\begin{document}
\nocite{btxdoc,btxhak}
\bibliography{demo2}
\bibliographystyle{plain}
\end{document}
```

demo2.bib です：

```
% excerpt from: btxdoc.bib
@UNPUBLISHED{btxdoc,
   author = "Oren Patashnik",
   title = "{{\BibTeX ing}}",
   note = "Documentation for general {\BibTeX} users",
   month = "8~" # feb,
   year = 1988 }

@UNPUBLISHED{btxhak,
   author = "Oren Patashnik",
   title = "Designing {\BibTeX} Styles",
   note = "The part of \BibTeX's documentation
                    that's not meant for general users",
   month = "8~" # feb,
   year = 1988 }
```

そして、plain.bst から、大事そうな部分を抜粋してみますと、こんな感じです：

```
ENTRY
  { author
    month
    note
    title
    year
  }
  {}
  { label }

FUNCTION {unpublished}
{ output.bibitem
  format.authors "author" output.check
  new.block
  format.title "title" output.check
  new.block
  note "note" output.check
  format.date output
  fin.entry
}

READ

ITERATE {presort}

SORT

EXECUTE {begin.bib}

ITERATE {call.type$}

EXECUTE {end.bib}
```

（この抜粋は、plain.bst 全体の流れを示すためと、このサンプルに関連する限りでのものです。実際の plain.bst の中では：

- ENTRY ではもっと沢山の fields が宣言されています。
- 定義されている entry-type function も、unpublished だけではありません。article 等々も、もちろん定義されています。
- presort、begin.bib、end.bib の定義もここでは割愛していますが、実際にはちゃんとそれぞれ定義されています。）

BibTeX が生成する sample2.bbl は、こうなります：

```
\begin{thebibliography}{1}
  \bibitem{btxdoc} Oren Patashnik.
    \newblock {{\BibTeX ing}}.
    \newblock Documentation for general {\BibTeX} users,
    8~February 1988.
  \bibitem{btxhak} Oren Patashnik.
    \newblock Designing {\BibTeX} styles.
    \newblock The part of \BibTeX's documentation
    that's not meant for general users, 8~February 1988.
\end{thebibliography}
```

（plain.bst の中に、

```
MACRO {feb} {"February"}
```

という定義があるので、demo2.bib 内の month field の“feb”は、sample2.bbl では“February”に置換されています。また、.bib ファイルでは“#”は連結〔concatenation〕を意味します[7]。）

（どこかで“\BibTeX”の定義をしておかないと、このままタイプセットしたのではエラーになりますね…。）

以上の 2 つの .bst ファイルの例において、赤色の大文字で書かれているものが BibTeX の COMMANDs で、末尾に“$”が付いている茶色の文字列（と茶色の“*”）が built-in functions、そして、青色の文字列が user-defined functions です。
（なお、.bib ファイル内の文字列についても、青色や緑色にした部分がありますが、それらは飽くまで、.bst ファイルで定義されたり宣言されたりしている functions や fields との対応関係を分かりやすくするために、そうしてみた迄です。）

“EXECUTE”や“ITERATE”する function は、built-in function でない限りは、予め“FUNCTION”を使って定義しておく必要があります。

BibTeX には LaTeX の \usepackage とか \input みたいな仕組みは用意されてないみたいですので、必要な user-defined functions は、汎用的なものを含めて全て、ひとつの .bst ファイルの中でいちいち定義が行われています。

user-defined な function の定義内部で使われている“下請けの function”についても同様です。例えば、上の例の“unpublished”という function の定義の中では、“output.bibitem”や“format.authors”等々の functions が使われていますが、これらはみな、実際には plain.bst 内の“FUNCTION {unpublished}”よりも前の部分で、縷々定義がなされているわけです。

ということで、.bst ファイルの中味を理解するには BibTeX が提供する COMMANDs や built-in functions の働きが分からないといけないのですが、それらは btxhak.pdf で説明されています。

3 BibTeX の COMMANDs と functions

Designing BibTeX Styles (btxhak.pdf) [2a] の構成は、

5 Bibliography-style hacking
- 5.1 General description
- 5.2 Commands
- 5.3 The built-in functions
- 5.4 Name formatting

という風になっています。以下では、5.1～5.3 の内容について、私なりにまとめ直してみます（5.4 は割愛します）。
（なお、私はプログラミングのプの字も知らない人間ですので、専門的な観点からするとおかしな言い回しがありましたら、適宜正しい表現に脳内変換していただきたく、また、お詳しい皆さんにとっては当たり前すぎるような説明の部分は、何卒読み飛ばしてくださいますようお願いします。あと、正確な説明が必要な場合には、直接原文にあたるようにしてください。）

3.1 Postfix-Notation

btxhak.pdf の 5.1 は、

> You write bibliography styles in a postfix stack language.

という文章から始まります。どうやら、BibTeX の言語は、スタックを利用していて、その Notation としては postfix を採用しているということのようです[8]。代表的な（?）Notation には、Infix-Notation、Prefix-Notation、Postfix-Notation がありますよね。例えば、「1 に 2 を加える」という操作をそれぞれで表わしてみますと、

- Infix-Notation: 1 + 2　　(⟨*operand*⟩ ⟨*operator*⟩ ⟨*operand*⟩)
- Prefix-Notation: + 1 2　　(⟨*operator*⟩ ⟨*operand*⟩ ⟨*operand*⟩)
- Postfix-Notation: 1 2 +　　(⟨*operand*⟩ ⟨*operand*⟩ ⟨*operator*⟩)

のようになります。BibTeX の言語の Notation はこの 3 番目のやつということです。なお、この「BibTeX の言語」については、次の段落で、

> Basically the style file is a program, written in an unnamed language, that tells BibTeX how to format the entries that will go in the reference list.

となっていて、当初は“名無し”だったみたいですが、後には“the .bst language”と呼ばれたりもしています[9]。

それで、この the .bst language の objects は、

- 定数（constants）
- 変数（variables）
- functions
- スタック（stack）
- エントリ・リスト（entry list）

だそうです。

定数と変数の value type は、いずれについても、文字列(string)と整数(integer)とがあります。そして、string constant は " " で囲んで、integer constant には“#”を前置します[10]。BibTeX は原則大文字と小文字を区別しませんが、" " で囲まれた内部についてだけは、大文字と小文字の区別を保持します。また、スペースも、" " の内部のもののみが意味を持ちます。

---

[7] *BibTeXing* (btxdoc.pdf) [1a], 2.1 New BibTeX features: “2. You can now have as a field value … the concatenation of several strings. … You may concatenate as many strings as you like (except that there's a limit to the overall length of the resulting field); just be sure to put the concatenation character '#', surrounded by optional spaces or newlines, between each successive pair of strings.”

[8] ちなみに、BibTeX の有名なチュートリアルである Nicolas Markey, *Tame the BeaST: The B to X of BibTeX*, Version 1.4 – October 11, 2009 (ttb_en.pdf) の第 16 節は、“BibTeX uses the so-called Reverse Polish Notation.”という一文から始まります。「Reverse Polish Notation って何?」と一瞬身構えてしまいますが、Prefix-Notation の別名が Polish Notation で、Postfix-Notation はその逆だから、Reverse Polish Notation と呼ばれるのですね。Wikipedia には「〔Polish Notation の〕名称の由来は、ポーランド人の論理学者ヤン・ウカシェヴィチが考案したことによる。」とあったのですが、そんな雑な呼ばれ方って、ありなんでしょうか。

[9] Oren Patashnik, *BibTeX 1.0*, TUGboat 15-3 (1994), p. 271, 272; *Id.*, *BibTeX 101*, TUGboat 19-2 (1998), p. 206; *Id.*, *supra* note (4), p. 28, 29.

[10] “#”は、TeX だとマクロのパラメーターを表わして、BibTeX の .bib 内では concatenation を表わすので、ちょっと紛らわしいですよね。

BibTeX の COMMANDs は、後述のように 10 個ありますが、これらは（大文字か小文字かで違いはないものの）大文字で書かれることが多いです。

functions は、built-in functions と、“FUNCTION”という COMMAND を使って定義される user-defined functions とに分けられます。built-in functions の名前には（文字列から成る function 名の場合には）、末尾に“$”が付いています。

variable 名と function 名は、数字で始まっていてはいけなくて、あと、次の 10 個の文字は名前には使えません：

```
" # % ' ( ) , = { }
```

variables や functions は、その名前そのものを表わす場合には、variable 名や function 名の前に“'”を前置します（私のような素人にはちょっと分かりにくいのですけれど、variable や function の名前だけだとその「値」やその「実行」(?) を表わして、その名前そのものを表わすには“'”が必要とのことです）。

btxhak.pdf の 5.1 に挙げられている次のような sample function fragments を使って、BibTeX の Postfix-Notation について、確認してみましょう。

一つ目の例は、

```
▸ label "a" * 'label :=    % label := label * "a"
```

というものです[11]。

ここで、“label”は string variable で、その値は“Pat88”だとしましょう。"a" は string constant です。

そして、“*”と“:=”は built-in function です（これらには末尾に“$”は付いていませんが）。いずれも引数を 2 つ必要とする function で、“*”は連結 (concatenation)、“:=”は代入 (assignment) を行うものです。

なお、the .bst language においても、“%”はコメントを表わすシンボルで、このコメントでは Postfix-Notation が Infix-Notation に書き直されています。

そのコメントを読めば分かりますように、この操作の意味は、string variable“label”の値に“a”をアペンド（append）する、というものです。

これを BibTeX の側から見た場合には、まず最初に、変数“label”に遭遇すると、その値“Pat88”をスタックに push します。次に文字列 "a" に遭遇するので、これもスタックに push します。

次は“*”に遭遇しますが、“*”は引数を 2 つ必要とするので、スタックから順に、"a" と“Pat88”とを pop してきます。そして“*”の実行結果である“Pat88a”をスタックに push します。

続いて“'label”に遭遇するので、“label”という変数名をスタックに push します。次に“:=”に遭遇しますが、これもまた引数を 2 つ必要とするので、スタックから順に、変数名“label”と文字列“Pat88a”とを pop してきます。そして、“:=”が実行されて、“label”に“Pat88a”が代入されます。（丁寧に説明すると恐らくこうなるのかなと思うのですが、幸いなことに日本語は動詞が最後に来るので、左から順番に読んでいって、「変数 label の値に文字列 a を連結して、それを変数 label に代入する」と理解することもできそうです。）

もう一つ挙げられている例は、

```
▸ number.label #1 + 'number.label :=
                % number.label := number.label + 1
```

というものです[12]。

今度は、“number.label”は integer variable だとします。“#1”は integer costant です。そして、“+”と“:=”は、built-in function で、どちらも引数を 2 つ必要とします。

こちらもコメントを読めば、この操作の意味は、integer variable“number.label”の値を 1 だけインクリメント (increment) する、というものです。

integer constant には“#”を前置する、ということにだけ注意すれば、あとは一つ目の例と大体同じです。

btxhak.pdf の 5.1 に載っている例は以上の二つなのですけれど、ひとつ、オマケです。例えば“tempstrA”という string variable があったとして、

```
▸ 'tempstrA :=
```

というのは、どういう意味でしょうか。

原則通りに考えればいいだけなので、簡単ですかね。まず初めに、スタックのトップには仮に“somestring”という文字列があるものとします。それで、“'tempstrA”に遭遇すると、“tempstrA”という変数名がスタックに push されて、続く“:=”は引数を 2 つ必要とするので、“tempstrA”と“somestring”が pop されてきて、結局、変数“tempstrA”に文字列“somestring”が代入されるということになります。つまりこれは、平たく言うと、「スタックのトップを“tempstrA”に代入する」という操作になりますね。

3.2 COMMANDs

BibTeX には 10 個の COMMANDs があります。以下、私なりにざっくりと説明をしてみます[13]。

3.2.1 Declaration

ENTRY {⟨*field-list*⟩}{⟨*int-var-list*⟩}{⟨*str-var-list*⟩}

BibTeX が処理する各エントリは、エントリごとに、

- fields (string)
- integer entry variables (integer)
- string entry variables (string)

という 3 種類の変数を有しています。ENTRY は、それらを宣言する COMMAND です。変数の種類に対応する 3 つの引数を取ります。.bst ファイルには、ENTRY が必ず一つ必要です。

- ⟨*field-list*⟩ は、エントリの fields の宣言で、author とか title 等々必要なものを、スペースで区切って並べます。なお、ここで明示的に宣言されたものの他に、crossref という built-in field が内部的に追加で宣言されています。
- ⟨*int-var-list*⟩ は、エントリが有する integer variables の宣言です。一応宣言できるように用意はされていますが、btxbst.doc [3a] では“There are no integer entry variables.”として、空になっています。
- ⟨*str-var-list*⟩ は、エントリが有する string variables の宣言です。例えば plain.bst だと label が、alpha.bst では label、extra.label、sort.label が宣言されています[14]。ここで明示的に宣言されたものの他に、sort.key$

[11] alphabetic label の際に、重複するラベルに a、b、c、…を append する“reverse.pass”という function の内部には実際、“label extra.label * 'label :=”というコードがあります。

[12] 実は、btxhak.pdf の 5.1 に載っていた例は“lab.width #1 + 'lab.width :=”でしたが、ここでは（numeric label の場合の）“longest.label.pass”という function 内にあるコードに替えました。

[13] 記法については *The LaTeX Companion* を参考にしました。*TLC* はこれまで 3 版まで出ていますが、第 1 版（1993 年）はアスキー刊の邦訳（1998 年）しか私は手元になくて、第 2 版（2004 年）と照合してみますと、第 1 版の“13.7 Understanding BibTeX Styles”(+ 13.8) と、第 2 版の“13.6 The BibTeX style language”とが大体同じ内容みたいです。なお、残念ながら、昨年刊行された第 3 版（2023 年）には、the .bst language の解説はなくなってしまったようです。

[14] 細かい話になりますが、ここで宣言されている string entry variable“label”は、alpha.bst では実際 Bibliography のエントリのラベル文字列として使われていますが、plain.bst のほうは、ラベルは連番の数字ですので、“label”はラベルの文字列としてではなく、ラベルの数字列の最大幅を測るためだけに使われています（cf. output.bibitem, calc.label, longest.label.pass）。

という built-in string entry variable も、内部的に追加で宣言されています。

`INTEGERS {⟨int-var-list⟩}`

integer variables を宣言する COMMAND です。variable 名をスペースで区切って並べます。変数は、使う前に予め宣言しておく必要があります。

この COMMAND を使って明示的に宣言されるものの他に、`entry.max$` と `global.max$` という built-in integer variables も、内部的に宣言されています。

`STRINGS {⟨str-var-list⟩}`

string variables を宣言する COMMAND です。

ちなみに、*Tame the BeaST* の第 15 節には、

> Contrary to integer variables, string variables are very 'expensive', and BibTEX limits the number of such variables to twenty.

と書かれているのですが、手元でチラッと試してみた限りでは、21 個宣言してみても、特にエラーにはなりませんでした。

3.2.2 Definition

`FUNCTION {⟨function-name⟩}{⟨definition⟩}`

user-defined な function を定義する COMMAND です。一つ目の引数は function 名で、二つ目の引数はその定義となります。function も、使う前に予め定義しておく必要があります。

`MACRO {⟨macro-name⟩}{"⟨definition⟩"}`

文字列の置換マクロを定義する COMMAND です。二つ目の引数の置換文字列は " " で囲まないといけません。

.bib ファイル内の @STRING による置換の定義のほうが、この COMMAND で定義した置換よりも優先されます。

3.2.3 Execution and Manipulation

`EXECUTE {⟨function-name⟩}`

引数に指定した function を実行する COMMAND です。

`ITERATE {⟨function-name⟩}`

.bib ファイルから読み込まれたエントリ・リストの個々のエントリに対して、上から順番に、function を実行します。

READ してすぐの場合には、エントリは .aux 内の \citation の順番に並んでいますが、SORT の後の場合には、`sort.key$` に格納された文字列をキーにしてアルファベット順に並べ直された順番ということになります。

`REVERSE {⟨function-name⟩}`

読み込まれているエントリ・リストの個々のエントリに対して、逆順に（下から順番に）、function を実行します。

例えば、alpha.bst の場合には、

```
READ
ITERATE {presort}
SORT
ITERATE {forward.pass}
REVERSE {reverse.pass}
```

という順序で処理されていて、まず、READ で読み込んだエントリ・リストの個々のエントリに対して `presort` を実行して `sort.key$` にキーとなる文字列を代入し、そのキーを基準に SORT でソートしています。

それから、そのソート後のエントリ・リストに対して、上から順に `forward.pass` を実行した後、下から順番に `reverse.pass` を実行しています。

`READ`

.aux ファイルに並んでいる \citation の順番に、.bib ファイルからエントリを読み込んで、その各 field に値をセットする COMMAND です。引数はありません。

.bst ファイルには READ が必ず一つ必要です。また、ENTRY と MACRO は、READ よりも前に設定しておく必要があります。

`SORT`

読み込まれているエントリ・リストを、各エントリが有している string entry variable である“`sort.key$`”に格納されている文字列をキーにして、アルファベット順にソートします。引数はありません。

3.3 Built-in Functions

BibTEX には 37 個の built-in functions が用意されていますが、その説明の前にまず、“special character”に関してと、ここでの function の記法に関して触れさせてください。

3.3.1 special characters

BibTEX の“special character”については、*BibTEXing* (btxdoc.pdf) [1a] の“4 Helpful Hints”の 6. と 17. とで説明されています。

6. にはまず、次のようにあります：

> In general, if you want to keep BibTEX from changing something to lower case, you enclose it in braces. You might not get the effect you want, however, if the very first character after the left brace is a backslash. The “special characters” item later in this section explains.

BibTEX には後述する `change.case$` という built-in function があって、いくつかの field の文字列については、この function を使って（冒頭の単語の一文字目以外が）小文字に変換されるように設定されています。もしもそれを防ぎたい場合には、小文字にされたくない文字をブレースで囲んでおけばよいと書かれています。但し、開きブレースのすぐ次の文字がバックスラッシュであるような場合に限っては、`change.case$` の働きはそのブレースの内側の文字列にも及んでしまう、ということです。そして、この、

開きブレース＋バックスラッシュ＋文字列＋閉じブレース

という塊のことを、BibTEX では“special character”と称しているようです。

17. に例が挙げられていますが、例えば、

```
author = "\AA{ke} {Jos{\'{e}}} {\'{E}douard} G{\"o}del"
```

という field の場合、“{\'{E}douard}”と“{\"o}”が、special character です。

“{\'{e}}”は、それが更にブレースで囲まれているので、これは special character には当たりません（且つ、このようにブレースをネストさせると〔つまり、special character ではないと〕、ラベルの文字列にアクセント記号を付けるのにも失敗するということが、btxdoc.pdf の 2.1 の 4. で説明されています）。

次いで、例えば、

```
The {\TeX BOOK\NOOP} Experience
```

というタイトルに対して `change.case$` を実行すると、

```
The {\TeX book\NOOP} experience
```

になるということが示されています。

“The”の“T”は大文字のままで、他方、“Experience”は“experience”に変換されています。そして、special character の内部については、“BOOK”は“book”に変換されてしまっていますが、“\TeX”と“\NOOP”という TEX のコントロールシーケンスは手付かずのままとなっています。

この special character の使い途は三つ挙げられていて、一つ目は、今見ましたように、アクセント記号を付けるのに使われます。

二つ目は、ソート順に細工をするのに使われています（btxdoc.pdf の 2.1 の 6. で説明されています）。

例えば、ある著者の 2 巻本の文献について、第 1 巻の刊行年が 1968 年で、第 2 巻は 1971 年刊行だとします。そして第 1 巻だけ 1973 年に第 2 版が出ているとします。plain.bst の場合、sort.key$ は「著者名＋刊行年＋表題」なので、そのままですと、第 1 巻、第 2 巻、第 1 巻第 2 版の順に並びます。これを、第 1 巻、第 1 巻第 2 版、第 2 巻の順に並ぶようにしたいとします。

その場合、まず、\noopsort というマクロを、

```
\newcommand{\noopsort}[1]{}
```

のように、引数を呑み込むように定義しておきます。それで、当該エントリの year field を、

- 第 1 巻第 2 版：year = "{\noopsort{a}1973}"
- 第 2 巻：year = "{\noopsort{b}1971}"

のように入力します。すると、sort.key$ 内の年号部分は、purify$ という built-in function の働きで“a1973”と“b1971”になります。これで、a のほうが b より先ですので、1973 年刊の第 1 巻第 2 版が、1971 年刊の第 2 巻よりも先に並ぶことになります。Bibliography へ最終的な出力においては、\noopsort のお陰で、刊行年の部分に“a”や“b”は現われません。

なお、データベース内のこの著者の文献がこの 3 冊だけなら以上でうまくいきますが、他にもこの著者の文献が含まれていて、それが 1968 年よりも後に刊行されたものだとすると、第 1 巻と第 1 巻第 2 版との間に、その文献が割り込んでしまいます。その場合には、

- 第 1 巻第 2 版：year = "{\noopsort{1968a}1973}"
- 第 2 巻：year = "{\noopsort{1968b}1971}"

という風に入力すれば、1968 年刊の第 1 巻のすぐ次に、第 1 巻第 2 版と第 2 巻とが続くようにできます。

最後に三つ目としては、alpha.bst のラベルに文字列を押し込むのに使われています（xampl.bib ファイルに例があります）。

alpha.bst のラベルは、「著者名の最初の 3 文字＋刊行年の下 2 桁」で構成されていますが、BibTeX は special character 全体を 1 文字としてカウントする、という仕組みを利用しています。

author field の入力は、普通は例えば、

```
author = "Knuth, Donald"
```

みたいな感じで、仮に当該文献の刊行年が 1984 年だとしますと、alpha.bst によるのラベルは“[Knu84]”となりますよね。

それに対して、まず、引数をそのまま出力する

```
\newcommand{\singleletter}[1]{#1}
```

というマクロを定義した上で、author field を、

```
author = "Kn{\singleletter{uth}}, Donald"
```

のように入力すると、BibTeX にとってのこの名前の最初の 3 文字は“Kn{\singleletter{uth}}”なので、Bibliography のラベルが、“[Knuth84]”になります。

3.3.2 ここでの function の引数の表わし方

TLC, 1st (1993) の Table 13.5 や 2nd (2004) の Table 13.8、それから *Tame the BeaST* (2003–2009) の第 17 節では、built-in functions について解説する際の記法として、

$$\mathcal{S}_1\ \mathcal{I}\ \mathcal{S}_2 \quad \texttt{format.name\$} \quad (\mathcal{S})$$

みたいなカッコいい書き方が採用されています。

他方、第 2 部のほうで眺める予定の btxbst.doc の実際の使用例のドキュメンテーションや pseudo code の部分では、

```
format.name$(s, nameptr, "{ff~}{vv~}{ll}{, jj}")
```

というような書き方がされています。

TLC/Tame the BeaST 方式では、引数がそのまま並べられて function に前置されているのに対して、pseudo code のほうでは、引数はコンマで区切ってパーレンに入れて後置しています（pseudo code 自体は、Infix-Notation になっています）。

BibTeX のコードは Postfix-Notation なので、この pseudo code に対応する実際のコードでは、

```
s nameptr "{ff~}{vv~}{ll}{, jj}" format.name$
```

という風に、3 つの引数は function より前に置かれています。

慣れてしまえば、どういう記法で書かれていても問題ないのでしょうけれど、ここでは、もっと素人っぽく、

(⟨*name-list*⟩, ⟨*pointer*⟩, ⟨*pattern*⟩) format.name$

みたいに書くことにしようと思います。

これは私が今勝手にでっち上げた記法ですので、ちゃんとしたドキュメントを読まれる際にはご注意ください。

3.3.3 BibTeX の 37 個の built-in functions

“3.2 COMMANDs”の部分では、COMMANDs を 3 種類に分けて説明をしましたが、built-in functions についてはうまく分類することができませんでしたので、なんとなく似たような機能のものを順番に説明することにします（37 個ある built-in functions をアルファベット順ではなく適当な順番に並べ替えてしまったので、function が探しにくくなってしまいました…）。

前説が随分と長くなってしまいましたが、これでようやく、built-in functions の説明に入れます。

(⟨*integer*⟩, ⟨*integer*⟩) >
(⟨*integer*⟩, ⟨*integer*⟩) <

引数として、integer か integer variable を 2 つとって、両者の間に不等号の関係が成立するかどうかをテストします。成立する場合には 1 を push して、しない場合には 0 を push します。例えば、

```
nameptr #1 >
  {⟨true branch⟩}
  {⟨false branch⟩}
if$
```

みたいな感じに使われます。

(⟨*integer*⟩, ⟨*integer*⟩) =
(⟨*string*⟩, ⟨*string*⟩) =

2 つの引数が一致するかどうかをテストして、一致する場合には 1 を push して、しない場合には 0 を push します。例えば、

```
tempstrA "others" =
  {⟨true branch⟩}
  {⟨false branch⟩}
if$
```

みたいな感じです。

(⟨*integer*⟩, ⟨*int-variable*⟩) :=
(⟨*string*⟩, ⟨*str-variable*⟩) :=

1 つ目の引数を 2 つ目の引数に代入します。

(⟨*integer*⟩, ⟨*integer*⟩) +
(⟨*integer*⟩, ⟨*integer*⟩) -

2 つの引数の加算と減算です。その結果を push します。

(⟨*string*⟩, ⟨*string*⟩) *

2 つの引数を連結（concatenation）して、その結果を push します。

(⟨*variable*⟩) empty$
(⟨*field*⟩) missing$

引数の ⟨*variable*⟩ が空かどうかや、⟨*field*⟩ が存在するかどうかをテストして、空や存在しないときには 1 を push して、そうでない場合には 0 を push します。例えば、

```
author empty$
  {⟨true branch⟩}
  {⟨false branch⟩}
if$
```

ですとか、

```
crossref missing$
  {⟨true branch⟩}
  {⟨false branch⟩}
if$
```

みたいな感じです。

(⟨*letter*⟩) chr.to.int$
(⟨*integer*⟩) int.to.chr$
(⟨*integer*⟩) int.to.str$

chr.to.int$ は、アルファベット一文字を ASCII コードに変換して、逆に、int.to.chr$ は、⟨*integer*⟩ を ASCII コードとして解釈して、対応するアルファベットに変換します。int.to.str$ は、⟨*integer*⟩ を string に変換します。例えば、

```
"a" chr.to.int$ top$
#98 int.to.chr$ top$
#99 int.to.str$ 'tempstrA := tempstrA top$
```

としてみますと、ターミナルには、“97”、“b”、“99” と表示されます。

(⟨*string*⟩) num.names$
(⟨*string*⟩) text.length$
(⟨*string*⟩) width$

num.names$ は、引数の ⟨*string*⟩ に含まれる “and” の数を数えて、それに 1 を加えたものを push します。author などの field に含まれている名前の数を数えるのに使われます。例えば、“"A and B" num.names$ top$” とすると、ターミナルには “2” と表示されます。

text.length$ は、引数の ⟨*string*⟩ の文字数を数えて、その数を push します。アクセント付きの文字 (= special character) は 1 文字として数えますが、ブレースは数えません。

width$ は、100 分の 1 ポイントを単位にして引数の ⟨*string*⟩ の長さを測った値を push します。btxhak.pdf の 5.3 の説明には、“ “special characters” are handled specially.” としか書かれてないのですが、手元で試してみたところ、

- "TeX"、"TeX{}"、"\TeX" を width$ で測ってみますと、“1916”、“2916”、“2416” となり、
- "Edouard" と "{\'Edouard}" は、どちらも “3741” で、
- "Knuth" と "Kn{\singleletter{uth}}" も、どちらも “2835” となりました。

つまり、通常はブレースやバックスラッシュも計測されますが、special character についてはそのブレースやコントロールシーケンスは計測に含まれないということのようです。この function は、ラベルの長さを測って比較して、最長のラベルを選び出すために使われています。

(⟨*string*⟩, ⟨*start*⟩, ⟨*length*⟩) substring$
(⟨*string*⟩, ⟨*length*⟩) text.prefix$

ここで、⟨*start*⟩ と ⟨*length*⟩ は integer です。

substring$ は、⟨*string*⟩ の部分文字列を取り出すものです。語頭から ⟨*start*⟩ 番目の文字から ⟨*length*⟩ 文字分を取り出して push します。⟨*start*⟩ が負の integer の場合は、語末から数えます。

例えば、“cite$ #1 #3 substring$” であれば、cite$ の文字列の最初の 3 文字を取り出して、“year #-1 #2 substring$” だと、year field の末尾 2 文字を取り出します。

text.prefix$ は、⟨*string*⟩ の語頭の ⟨*length*⟩ 文字分を取り出して push します。text.prefix$ は、文字数のカウントの際に、special character は 1 文字として数えて、また、ブレースは数えません。この function は、alphabetic label のアルファベット 3 文字の部分を生成するのに使われています。

(⟨*string*⟩) purify$

⟨*string*⟩ に含まれているスペース、ハイフン、タイ以外の nonalphanumeric characters を取り除きます (ハイフンとタイはスペースに変換されます)。special character のブレースやコントロールシーケンスも取り除かれます。したがって、例えば、"ab-c␣\TeX␣de~f" と "{\noopsort{1968a}1973}" をそれぞれ purify$ すると、“ab␣c␣TeX␣de␣f” と “1968a1973” になります。

(⟨*string*⟩, ⟨*switch*⟩) change.case$

ここで、⟨*switch*⟩ は string です。

⟨*switch*⟩ が、

- "t" or "T" のときは、⟨*string*⟩ 内の文字列の冒頭の一文字を除いて、⟨*string*⟩ を小文字に変換し、
- "l" or "L" のときは、⟨*string*⟩ 全体を小文字に変換し、
- "u" or "U" のときは、⟨*string*⟩ 全体を大文字に変換します。

"t" / "T" 指定の場合には、⟨*string*⟩ に含まれている文字列の中の、「コロン + (1 個以上の) スペース」の次の一文字も、小文字変換から除外します。

⟨*string*⟩ 内の文字列の一部をブレースで囲むと、当該文字列に対してはこの function の働きが及ぶのを防ぐことができますが、special character 内部の文字列に限っては、その働きが及びます。

(⟨*name-list*⟩, ⟨*pointer*⟩, ⟨*pattern*⟩) format.name$

ここで、⟨*name-list*⟩ は “and” で区切られた名前のリストで、⟨*pointer*⟩ は integer、そして ⟨*pattern*⟩ は名前を構成する 4 つの要素 (First, von, Last, Jr.) の整形パターンの指示です。

この function は、“and” で区切られた ⟨*name-list*⟩ の中の、⟨*pointer*⟩ 番目の名前を、⟨*pattern*⟩ の指示通りに整形して、push するというものです。

⟨*pattern*⟩ の部分は、上の 3.3.2 のところに出てきたような、"{ff~}{vv~}{ll}{, jj}" みたいな感じになりますが、その詳細については、btxhak.pdf の 5.4 や *Tame the BeaST* の第 18 章に譲ります。

(⟨*string*⟩) add.period$

⟨*string*⟩ の末尾が “.”、“?”、“!” でない場合に、⟨*string*⟩ に “.” を付加して、push します。

quote$

btxhak.pdf の 5.3 には “Push the string consisting of the double-quote character.” と書かれていて、試してみると、実際 “"” がスタックに push されるのですけれど、standard styles では一度も使われてませんので、使い途がよく分かりません。

cite$
type$
preamble$

cite$ は、.bib ファイルから読み込まれたエントリ・リストのエントリの、“cite-key” を push します。つまり、例えば、当該エントリが “@BOOK{DK:texbook, ...}” だとしますと、“DK:texbook” という string のことです。

type$ は、.bib ファイルから読み込まれたエントリ・リストのエントリの、“entry-type” を push します。つまり、例えば、当該エントリが “@BOOK{DK:texbook, ...}” だとしますと、“BOOK” という string のことです。

preamble$ は、\bibdata (i.e., \bibliography) で指定された .bib ファイル内のすべての @PREAMBLE を連結して、push します。

`call.type$`

.bib ファイルから読み込まれたエントリ・リストの個々のエントリに対して、その“entry-type”と同名の function を実行します。

つまり、例えば、`@ARTICLE{...}` というエントリに対してはその entry-type と同名の `article` という function が、同様に、`@BOOK{...}` というエントリに対しては `book` という function が呼ばれて、実行されます。

(⟨*test*⟩,{⟨*function1*⟩},{⟨*function2*⟩}) `if$`

ここで、⟨*test*⟩ は integer です。

`if$` は、引数を 3 つとって、⟨*test*⟩ が 0 よりも大きければ ⟨*function1*⟩ を実行し、そうでなければ ⟨*function2*⟩ を実行します。

普通は、⟨*test*⟩ は 1 か 0 で、

```
⟨test⟩
  {⟨function1⟩}
  {⟨function2⟩}
if$
```

という形に書かれることが多いと思います。

({⟨*function1*⟩},{⟨*function2*⟩}) `while$`

ここで、⟨*function1*⟩ は integer を push するような処理です。

⟨*function1*⟩ が push する integer が正である限り、⟨*function1*⟩ の実行と ⟨*function2*⟩ の実行を繰り返します。

例えば、`tempcntA` という integer variable があるとして、その値が 3 であるとき、

```
{tempcntA #0 >}
{"Toi!" top$
 tempcntA #1 - 'tempcntA :=}
while$
```

としてみますと、ターミナルに“Toi!”が 3 回出力されます。pseudo code っぽく書き直してみますと、

```
while tempcntA > 0
  do
    top$("Toi!")
    tempcntA := tempcntA - 1
  od
```

みたいな感じでしょうか。

`skip$`

何もせず、次の処理へとスキップします。例えば、

```
output.state before.all =
  'skip$
  { after.block 'output.state := }
if$
```

という風に使われます（これは、“`new.block`”の定義です）。

(⟨*something*⟩) `pop$`

ここで、⟨*something*⟩ は、スタックのトップにあるものだと考えてください[15]。

この function は、スタックのトップにある ⟨*something*⟩ を pop してきて、それを削除します。

（この書き方だと、⟨*something*⟩ を push して、pop して、そして削除、みたいに見えてしまいますが…。）

(⟨*something*⟩) `duplicate$`

ここで、⟨*something*⟩ は、スタックのトップにあるものだと考えてください。

この function は、引数の ⟨*something*⟩ をコピーして、⟨*something*⟩ を 2 つ push します。例えば、

```
duplicate$ empty$
  { pop$ "" }
  { "{\em " swap$ * "}" * }
if$
```

というような使われ方をします（これは、“`emphasize`”の定義です。スタックのトップが空かどうかをチェックするために、一旦そのコピーを作っています。また、`dupulicate$` の引数と `pop$` の引数、`swap$` の 1 つ目の引数は、そのときどきのスタックのトップです）。

(⟨*something*⟩,⟨*something*⟩) `swap$`

スタックのトップにあるものとトップから 2 番目にあるものとを pop してきて、それらを入れ替えて push します。

(⟨*something*⟩) `top$`
`stack$`

`top$` は、スタックのトップを pop してきて、それをターミナルに出力します。

`stack$` は、スタックの内容をすべてターミナルに出力するものだそうです（試してません…）。

(⟨*string*⟩) `write$`
`newline$`

`write$` は、引数の ⟨*string*⟩ を .bbl ファイルに書き出します。

`newline$` は、.bbl ファイルに改行の指示を書き出します。

(⟨*string*⟩) `warning$`

“`Warning--`”という prefix を付けて、引数をターミナルに出力します。

* * *

以上の COMMANDs と built-in functions の説明で、2.1 で例に挙げたような私が適当に作った .bst ファイル程度であれば、理解できるものと思います。

また、プログラミングに慣れている方であれば、*Designing BibTeX Styles* (btxhak.pdf) [2a] の 5.2 と 5.3 を読んで、あとは実際のコードさえ見れば、どんな .bst ファイルでも読んだり書いたりできるようになられるのかも知れません。

でも、私だったらもうちょっとサンプルが見たいですし、できれば自然言語で説明してほしいです。

幸いにも、*BibTeX 'plain' family* (btxbst.doc) [3a] では、4 つの standard styles（plain.bst、unsrt.bst、alpha.bst、abbrv.bst）のコードについて、詳細に説明がなされています。

というわけで、第 I 部はここまでにして、続く第 2 部では btxbst.doc を眺めてみることにします（先に白状しておきますが、第 2 部は、ただの切り貼りなのですけれど）。

それでは、ひとまず、Merry TeXmas & Happy TeXing!

[15] ここで苦し紛れに ⟨*something*⟩ としたものは、btxhak.pdf や *TLC* では“literal”と称されています。*TLC*, 2nd, Table I 3.8 の説明を転記してしまいますと：“A “literal” $\mathcal{L}$ is an element on the stack. It can be an integer $\mathcal{I}$, a string $\mathcal{S}$, a variable $\mathcal{V}$, a function $\mathcal{F}$, or a special value denoting a missing field. ”とのことです。

それで、pop$、duplicate$、swap$、top$ についてはそれぞれ、$\mathcal{L}$ pop$、$\mathcal{L}$ duplicate$ ($\mathcal{LL}$)、$\mathcal{L}_1\mathcal{L}_2$ swap$ ($\mathcal{L}_2\mathcal{L}_1$)、$\mathcal{L}$ top$ という風に書かれています。

実は、:=、empty$、missing$ についても、*TLC* だと、$\mathcal{LV}$:=、$\mathcal{L}$ empty$ ($\mathcal{I}$)、$\mathcal{L}$ missing$ ($\mathcal{I}$) という書き方がされています。