LuaTeX-ja パッケージ

2011/4/4

本パッケージは，（最低でも pTeX と同等の水準の）日本語組版を LuaTeX 上で実現させることを目標としたマクロである．まだまだ足りないところはあるが，とりあえず動くようになった？ので公開する．

■特徴

- 欧文フォント / 和文フォントの内部独立管理．
- 違う和文フォントでも「メトリックが同じ」なら，字間に空白を挿入する際には同じものとして扱う．例：「ほげほげ）(ふがふが」は以下の出力より得られた：

  ```
  ほげほげ）{\gt （ふがふが}
  ```

- 欧文や和文のベースライン補正が可能．
- pTeX とある程度コマンド名が互換．

■制限

- 全体的にテスト不足です．
- 合字の前後の和欧文間空白の挿入処理は適当（合字でない場合の処理をそのまま流用）．
- **現時点で LaTeX での使用は全く考慮されていません．“plain LuaTeX” で使ってください．**
- \accent を和文文字に対して使うことはできません．これは「フォントを後で置換する」という実装上，仕方のないことだと思われます．
- 数式中の日本語は想定していません．\hbox か何かで囲ってください．
- pTeX にあった以下の機能はまだ実装していません．
  - \euc, \jis, \sjis, \kuten といった，コード変換プリミティブ．
  - \kansuji, \kansujichar.
  - \showmode, \jfam, \jcharwidowpenalty.
  - 縦組み関連一式．\tate, \tfont, \tbaselineshift, \dtou, . . .

■ファイル構成

- luatexja-core.sty: マクロ本体．拡張子は sty であるが，この単一のファイルで plain TeX と LaTeX 両方に対応するように設計する方針である．しかし，**現時点で LaTeX での使用は全く考慮されていない．**
- luatexja-core.lua: Lua コード部分．
- luatexja-kinsoku.tex: 禁則用ペナルティ等のパラメータを書いたファイル．下のファイルによって ukinsoku.tex (in upTeX-0.30) から自動生成されたもの．
- luatexja-kinsoku_make.tex: 上の luatexja-kinsoku.tex を作るためのファイル．
- luatj-ujis.lua: upTeX-0.30 の ujis.tfm ベースのメトリックサンプル．
- luatj-mono.lua: 「全文字が全角幅」のメトリックサンプル．

使用方法

大雑把に言うと，plain TEX の状況で，以下のようにすればよい．

```
\input luatexja-core.sty                % ←マクロ本体を読み込み
\loadjfontmetric{mt}{ujis}              % ←メトリックの読み込み
\font\tenipam={file:ipam.ttf}at13.5\jQ
\jfont\tenipam{mt}                      % 和文フォントとして認識
\tenipam\parindent=1\zw
\yabaselineshift=32768                  % （例）32768 sp =  0.5 pt
\rm abcほげほげ）（あいう本文本文……
```

実装解説

■ attributes, dimensions, . . .

以下は LuaTEX-ja パッケージ内で使用する attribute やその他の種類のレジスタである．上 4 つは内部処理用なので利用者が意識することはない．それ以外は，pTEX に類似の名前の primitive があることから，意味は容易にわかるだろう：

- attribute `\luatexja@curjfnt`: 現在の和文フォント番号
  pTEX では内部のグローバル変数で「現在の横組 / 縦組和文フォント」をそれぞれ保持していたが，当然ながら欧文用 TEX ではそのようなことはそのままではできない．node p が保持している attribute `\luatexja@curjfn` の値 k は，「もし p の中身が和文文字であれば，そのフォントは k 番の番号のフォントである」という意味をもつ．
- attribute `\luatexja@charclass`: （和文文字の）文字クラス
- attribute `\luatexja@icflag`: この属性をもつ kern はイタリック補正由来である
  pTEX では，`\kern` 由来の kern と，イタリック補正由来の kern を内部で区別していた．しかし，欧文用の TEX ではそのような区別はなく，LuaTEX においても区別がないようである．
- language `\luatexja@japanese`: 「日本語」に対応する `\language` 番号
- attribute `\yabaselineshift`: 欧文文字ベースラインの補正量．
  – **sp $= 2^{-16}$ pt 単位の整数値**で指定．正の値を指定すると，その分だけ欧文文字は下にずれる．
  – 数式中では，box や rule もこの量だけずれる
  （よって，行中数式は全体が `\yabaselineshift` だけずれたように見える）．
- attribute `\ykbaselineshift`: 和文文字ベースラインの補正量．
  pTEX では「和文が主」という考えからか，常に和文文字のベースラインが基準であり，欧文文字の方をずらすことになっていた．しかし，「欧文の中に和文をちょっと入れる」ような場合では，逆に和文文字をずらす方が理にかなっているので，和文文字のベースラインもずらせるようにした．
  また，これを用いることで異なる文字サイズの文字を「上下中央揃え」で組むことも可能．
- skip `\kanjiskip`: 和文文字同士の間に入る空白量．
- skip `\xkanjiskip`: 和文文字と欧文文字の間に入る空白量．
- dimen `\zw`, `\zh`: 現在の和文フォントの「幅」/「高さ」（メトリックから指定）
- dimen `\jQ`, `\jH` $= 0.25\,\mathrm{mm}$

■和文フォントの定義

LuaTEX-ja では，大雑把にいうと和文フォントは「実際の字形」と「和文用のメトリック情報」の組である．

- メトリック情報は，和文文字の幅や，和文文字間の空白の入り方などを規定する．pTEX における JFM ファイルのようなものと考えてよい．

このため，和文フォントを使うには，以下のような手順が必要である．

1. `\font`⟨*font*⟩`={file:`⟨*fontfile*⟩`} [at` ⟨*size*⟩`]`
   TEX の `\font` primitive を用いてひとまずフォントを読み込む．和文フォントの場合は，このように luaotfload パッケージの機能を用いて TrueType/OpenType フォントを読み込むことになる．
   – この段階では，制御綴 ⟨*font*⟩ はまだ「現在の（欧文）フォントを変更する」意味である．
   – 例えば，

   ```
   \font\tenipam={file:ipaexm.ttf:script=latn;+jp90}
   ```

   のように各種の feature を用いてもよい．**本文書では全てのフォントに `jp90` feature を適用させている**（「辻」など）．

2. `\loadjfontmetric{`⟨*key*⟩`}{`⟨*file*⟩`}`
   Lua ソース `luatj-`⟨*file*⟩`.lua` に書かれたメトリック情報を読み込む．内部では ⟨*key*⟩ というキーで参照されることとなる．同じ key で 2 回以上読み込むことはできないが，同じメトリック情報に異なるキーをつけることは差し支えない．

3. `\jfont`⟨*font*⟩`{`⟨*key*⟩`}`
   制御綴 ⟨*font*⟩ を「metric key が ⟨*key*⟩ の和文フォント」と認識させる．正確に言うと，制御綴 ⟨*font*⟩ の意味を，「`\luatexja@curjfn` ← *v*」というように変更する．これにより，以後 ⟨*font*⟩ を発行しても，欧文フォントは変わらない．

■その他命令類

- `\(set/get)inhibitxspcode{`⟨*code*⟩`}{`⟨*num*⟩`}`,
  `\(set/get)xspcode{`⟨*code*⟩`}{`⟨*num*⟩`}`,
  `\(set/get)prebreakpenalty{`⟨*code*⟩`}{`⟨*num*⟩`}`,
  `\(set/get)postbreakpenalty{`⟨*code*⟩`}{`⟨*num*⟩`}`
  これらは pTEX の（`set/get` を抜かした）命令と意味は概ね同じであるが，以下の点が異なる．
  – 同じ文字コードについて `\prebreakpenalty`, `\postbreakpenalty` を両方指定することが可能．
  – `\(set/get)xspcode`（欧文文字用）は `\(set/get)inhibitxspcode`（和文文字用）の別名であり，

  ```
  \setxspcode{12289}{1}  \setinhibitxspcode{12289}{1}
  ```

  は全く同じ意味である．`\setxspcode`, `\setinhibitxspcode` の第 2 引数の意味は異なるが，両者の意味は文字コードにより判断している．

- `\inhibitglue`: 指定箇所での和文フォントメトリック由来の glue/kern の挿入を禁止する．内部的には，`user_id` が 30111 の whatsit node を作成している（メトリック由来の glue/kern 挿入処理で役目を終え，削除される）．

■大まかな処理の流れ

LuaTEX-ja パッケージでは，次のような流れで処理を行う．

- 行末空白の削除 : `process_input_buffer` callback

  通常，TEX において改行は空白とほぼ同じ意味であり，改行した箇所には自動的に空白が入るようになっている．だが，日本語ではそのような振る舞いは不自然であり，pTEX でも「和文文字で行が終わった場合，改行文字は無視する」という使用になっている．

  そこで，入力が和文文字で終わった場合，コメント文字を挿入することによりこの問題を解決する方法をとっている．この部分のコードは前田氏の jafontspec パッケージの部分から拝借したが，挿入する文字を `%` から（通常使用されることはないと思われる）`U+FFFFF` へと変更している．

- 和文フォントへの置換 : `hyphenate`, `hpack_filter` callbacks

  この段階の前では，和文文字であっても，それを内部で表している `glyph_node` p は，「`\temrm あ`」のように，欧文フォントが指定されている状態になっている．しかし，p は「現在の和文フォント」の番号も attribute `\luatexja@curjfn` として保持している．そのため，この段階では，「和文文字が格納されている」`glyph_node` p に対して，次を行う．
  - p のフォントを attribute `\luatexja@curjfn` の値に置換．
  - p の `language` field を `\luatexja@japanese` の値に置換．誤って和文文字間でハイフネーションがされてしまうのを防止するため．
  - p の文字の文字クラスを計算し，その値を attribute `\luatexja@charclass` に格納．これにより，`jp90` 等の feature によりグリフが置換されても，文字クラスの値は保たれる．

- （luaotfload パッケージによるグリフ置換等の処理はこの位置で行われる）

- メトリック由来 glue/kern の挿入 : `pre_linebreak_filter`, `hpack_filter`

  ここで，メトリックに由来する和文文字間の glue/kern を挿入する．基本的には連続する和文文字間に挿入するが，
  - 水平ボックスの先頭 / 末尾，段落の先頭 / 末尾には「文字コード`'boxbdd'` の文字」があると見做して空白を挿入する．
  - 和文文字とそうでないもの（欧文文字，ボックス等）の間に関しては，和文文字でない方は「文字コード`'jcharbdd'` の文字」であると見做す．
  - フォントの異なる 2 つの和文文字においても，両者のフォントの metric key と size が一致した場合は，挿入処理においては「同じフォント」であるかのように扱う．
  - そうでない場合は，両者の間に「文字コード`'diffmet'` の文字」があると見做して，両和文文字からそれぞれ glue/kern `gb`, `ga` を計算し，そこから実際に入る glue/kern を，関数 `ltj.calc_between_two_jchar_aux` で計算している：
    - `gb`, `ga` の片方が glue，もう片方が kern の場合は，glue 側のみ挿入．
    - そうでないときは，両者の平均値の空白を挿入する．

- `\kanjiskip`, `\xkanjiskip` の挿入 : `pre_linebreak_filter`, `hpack_filter`

  pTEX の `adjust_hlist` procedure とほぼ同様の処理を用いて，和文間 glue `\kanjiskip` や和欧文間 glue `\xkanjiskip` を挿入する．
  - pTEX と同様に，これらの自動挿入は（box/ 段落ごとに）`\[no]auto[x]spacing` を用いて制御できる．
  - 数式境界 (`math_node`) との間に `\xkanjiskip` を自動挿入するかの決定は，pTEX

では数字 0 との間に挿入するかどうかで判定していたが，LuaTEX-ja では「文字コード'math' の文字」で判定している．

- ベースライン補正 : pre_linebreak_filter, hpack_filter
  この段階では，（主として）欧文文字のベースラインをずらす作業を行う．幸いにして，LuaTEX で文字を表す glyph_node には y_offset field があるので，作業は楽である．
  補正量は，attribute \luatexja@yablshift の値（先も書いた通り，sp 単位）である．和文文字の補正量は \luatexja@ykblshift の値で指定されるが，以前の「和文フォントへの置換」処理において，\luatexja@yablshift へと値を移し変えているので，この段階では \luatexja@yablshift の値のみを気にしている．
  さて，実際に補正されるのは次の場合である :
  – 文字 (glyph_node)
  – ボックス・rule（文中数式内部）．これによって，数式全体が下がったように見えるはず．

- 和文文字の幅の補正 : pre_linebreak_filter, hpack_filter
  例えば，括弧類は「フォント中では全角幅だが，組版では半角幅として扱う」ことが多いが，このように文字幅を補正する処理を最後に行う．jafontspec パッケージのように，補正対象となる glyph_node p を，しかるべき量の glue と共に \hbox にカプセル化して行っている．

■和文フォントメトリックについて

LuaTEX-ja で用いる和文用のメトリック情報は，次のような構文で書かれた Lua ファイルである．見本として，luatj-ujis.lua を入れてある．

- jfm.dir: 組方向を指定する．将来的にはいずれ縦組（'tate'）を実装したいが，現時点では横組（'yoko'）のみの対応．

- jfm.zw, jfm.zh: それぞれ \zw, \zh のフォントサイズに対する割合を記述する．通常は両方とも 1.0 となるだろう．

- jfm.define_char_type(⟨*class*⟩, ⟨*chars*⟩)
  pTEX 用 JFM で言うところの「文字クラス」を定義する．
  – ⟨*class*⟩ は文字クラスを表す 1 以上 0x800 = 2048 未満の整数．
  – ⟨*chars*⟩ には，⟨*class*⟩ に属する「文字」達の Unicode におけるコード番号をリストの形 {...} で記述する．
    また，このリストには，以下の「仮想的な文字」も指定可能である．
    - '\boxbdd': 水平ボックスの先頭 / 末尾，段落の先頭 / 末尾．
    - 'jcharbdd': 和文文字達の連続の境界．
    - 'diffmet': 異なるメトリックの和文文字間に入る glue の計算に使われる．

- jfm.define_type_dim(⟨*class*⟩,⟨*left*⟩,⟨*down*⟩,⟨*width*⟩,⟨*height*⟩,⟨*depth*⟩,⟨*italic*⟩)
  文字クラス ⟨*class*⟩ ごとに，文字の寸法のフォントサイズに対する割合を記述する．
  – ⟨*left*⟩: 例えば開き括弧類は組版をする際には半角幅だが，TrueType フォント内では左に半角空白が付け加わって全角幅となっていることが多い．このような場合，逆に TrueType フォントを基準にすると，「左に半角幅ずらす」ことをしないといけない．⟨*left*⟩ はその「左へのずらし量」を指定する．
  – ⟨*down*⟩: 同様に，「下へのずらし量」を指定する．
  – ⟨*width*⟩, ⟨*height*⟩, ⟨*depth*⟩: それぞれ幅，高さ，深さ．
  – ⟨*italic*⟩: イタリック補正値．

- `jfm.define_glue(`⟨*bclass*⟩`,` ⟨*aclass*⟩`,` ⟨*width*⟩`,` ⟨*stretch*⟩`,` ⟨*shrink*⟩`)`
  文字クラス ⟨*bclass*⟩ の文字と ⟨*aclass*⟩ の文字の間に，自然長 ⟨*width*⟩，伸び ⟨*stretch*⟩，縮み ⟨*shrink*⟩ （いずれもフォントサイズ基準）の glue を挿入する．
- `jfm.define_kern(`⟨*bclass*⟩`,` ⟨*aclass*⟩`,` ⟨*width*⟩`)`
  文字クラス ⟨*bclass*⟩ の文字と ⟨*aclass*⟩ の文字の間に，幅 ⟨*width*⟩ の kern を挿入．

組版サンプル

出典：日本語 Wikipedia の「TEX」の項，2011/3/10

TEX（読み方については、「読み方」の小節を参照）は数学者・計算機科学者であるドナルド・クヌース (Donald E. Knuth) により作られた組版処理ソフトウェアである。

名称について

製作者であるクヌースによって以下のように要請されている。

■表記法

正しくは “TEX” と表記するが、それができない場合には “TeX” と表記する（“TEX” と表記するのは誤り）。

■読み方

TEX はギリシャ文字の T-E-X（タウ・イプシロン・カイ）であるから、「テックス」ではなく、ギリシャ語読みの [tex]（「テフ」）のように発音するのが正しい。しかしそのような発音は難しいので、クヌースは「テック」と読んでも構わないとしている。日本では「テフ」または「テック」という読み方が広まっている。

機能

TEX はマークアップ言語処理系であり、チューリング完全性を備えた関数型言語でもある。文章そのものと、文章の構造を指定する命令とが混在して記述されたテキストファイルを読み込み、そこに書かれた命令に従って文章を組版して、組版結果を DVI 形式のファイルに書き出す。DVI 形式というのは、装置に依存しない (device-independent) 中間形式である。

DVI ファイルには紙面のどの位置にどの文字を配置するかといった情報が書き込まれている。実際に紙に印刷したりディスプレイ上に表示したするためには、DVI ファイルを解釈する別のソフトウェアが用いられる。DVI ファイルを扱うソフトウェアとして、各種のヴューワや PostScript など他のページ記述言語へのトランスレータ、プリンタドライバなどが利用されている。

組版処理については、行分割およびページ分割位置の判別、ハイフネーション、リガチャ、およびカーニングなどを自動で処理でき、その自動処理の内容も種々のパラーメータを変更することによりカスタマイズできる。数式組版についても、多くの機能が盛り込まれている。TEX が文字などを配置する精度は $25.4/(72.27 \times 2^{16})$ mm（約 5.363 nm、4,736,286.72 dpi）である。

TEX の扱う命令文の中には、組版に直接係わる命令文の他に、新しい命令文を定義するための命令文もある。TEX のこの機能を使って使用者が独自に作った命令文はマクロと呼ばれ、こうした独自の改良をマクロパッケージと呼ばれる形で配布できる。

比較的よく知られている TEX 上のマクロパッケージには、クヌース自身による plain TEX、一般的な文書記述に優れた LATEX (LaTeX)、数学的文書用の $\mathcal{AMS}$-TEX などがある。一般の使用者は、TEX を直接使うよりも、TEX に何らかのマクロパッケージを読み込ませたものを使うことの方が多い。そのため、これらのマクロパッケージのことも “TEX” と呼ぶ場合があるが、本来は誤用である。

TEX のマクロパッケージには、他にも次のようなものなどがある。

- BIBTEX (BibTeX) ……参考文献リストの作成に用いる。
- SLITEX (SLiTeX) ……プレゼンテーション用スライドの作成に用いる。
- $\mathcal{AMS}$-LATEX (AMS-LaTeX) ……数学的な文書の記述に強い $\mathcal{AMS}$-TEX の機能と LATEX の機能を併せ持つ。

- XyMTEX (XyMTeX) ……化学構造式の描画に用いる。
- MusiXTEX (MusiXTeX) ……楽譜の記述に用いる。

TEX とそれに関連するプログラム、および TEX のマクロパッケージなどは CTAN（Comprehensive TeX Archive Network、包括 TeX アーカイブネットワーク）からダウンロードできる。

数式の表示例

たとえば

```
-b \pm \sqrt{b^{2} - 4ac} \over 2a
```

は以下のように表示される。

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

また、

```
f(a,b) = \int_{a}^{b}\frac{1 + x}{a + x^{2} + x^{3}}dx
```

は以下のように表示される。

$$f(a,b) = \int_a^b \frac{1+x}{a+x^2+x^3}dx$$

生い立ち

TEX は、クヌースが自身の著書 The Art of Computer Programming を書いたときに、組版の汚さに憤慨し、自分自身で心行くまで組版を制御するために作成したとされている。開発にあたって、伝統的な組版およびその関連技術に対する広範囲にわたる調査を行った。その調査結果を取り入れることで、TEX は商業品質の組版ができる柔軟で強力な組版システムになった。

TEX はフリーソフトウェアであり、ソースコードも公開されていて、誰でも改良を加えることができる。その改良版の配布も、TEX と区別できるような別名を付けさえすれば許される。また、TEX は非常にバグが少ないソフトウェアとしても有名で、ジョーク好きのクヌースが、バグ発見者に対しては前回のバグ発見者の 2 倍の懸賞金をかけるほどである。この賞金は小切手で払われるのだが、貰った人は記念に取っておく人ばかりなので、結局クヌースの出費はほとんど無いという。

クヌースは TEX のバージョン 3 を開発した際に、これ以上の機能拡張はしないことを宣言した。その後は不具合の修正のみがなされ、バージョン番号は 3.14、3.141、3.1415、... というように付けられている。これは更新のたびに数字が円周率に近づいていくようになっていて、クヌースの死の時点をもってバージョン π として、バージョンアップを打ち切るとのことである。

クヌースは TEX の開発と同時に、TEX で利用するフォントを作成するためのシステムである METAFONT も開発した。こちらのバージョン番号は 2.71、2.718、2.7182、... というように、更新のたびに数字がネイピア数に近づいていくようになっている。さらにクヌースは METAFONT を使って、TEX の初期設定欧文フォントである Computer Modern のデザインも行った。

TEX および METAFONT は、これもクヌース自身によって提唱されている文芸的プログラミング (Literate Programming) を実現する WEB というシステムで Pascal へトランスレートされることを前提に記述されている。しかし実際には WEB2C で C 言語に変換してコンパイルされ実行形式を得ることが多い。

TEX の日本語化

日本語組版処理のできる日本語版の TEX および LATEX には、アスキー・メディアワークスによる pTEX (pTeX) および pLATEX (pLaTeX) と、NTT の斉藤康己による NTT JTEX (NTT JTeX) および NTT JLATEX (NTT JLaTeX) などがある。

TEX の日本語対応において技術的に最も大きな課題は、複数バイト文字コードへの対応である。pTEX（および前身の日本語 TEX）は、JIS X 0208 を文字集合とした文字コード（ISO-2022-JP、EUC-JP、および Shift_JIS）を直接扱う。DVI フォーマットは元々 16 ビット以上の文字コードを格納できる仕様が含まれていた。しかしオリジナルの英語版では使われていなかったため、既存プログラムの多くは pTEX が出力する DVI ファイルを処理できない。またフォントに関係するファイルフォーマットが拡張されている。これに対して NTT JTEX は、複数の 1 バイト文字セットに分割することで対応している。例えば、ひらがなとカタカナは内部的には別々の 1 バイト文字セットとして扱われる。このためにオリジナルの英語版からの変更が小さく、移植も比較的容易である。ファイルフォーマットが同じなので英語版のプログラムで DVI ファイル等を処理することもできる。しかし後述するフォントのマッピングの問題があるため、実際には多くの使用者が NTT JTEX 用に拡張されたプログラムを使っている。

使用する日本語用フォントについては pTEX が写研フォントの使用を、NTT JTEX が大日本印刷フォントの使用を前提としており、それぞれフォントメトリック情報（フォントの文字寸法の情報）をバンドルして配布している。しかし有償であるこれらのフォントのグリフ情報を持っていなくても、画面表示や印刷の際に使用者が利用できる他の日本語用フォントで代用することができる。つまり写研フォントや大日本印刷フォントのフォントメトリック情報を用いて文字の位置を固定し、画面表示や印刷には他の日本語用フォントを用いていることが可能である。このため日本語化された TEX 関係プログラムのほとんどは、画面表示や印刷で実際に使うフォントを選択できるように、フォントのマッピング（対応付け）を定義する機能を持っている。

歴史的には、アスキーが日本語 TEX の PC-9800 シリーズ対応版を販売したために個人の使用者を中心に普及した。一方、NTT JTEX は元の英語版プログラムからの変更が比較的小さいという利点を受けて、UNIX および UNIX 互換 OS を使う大学や研究機関の関係者を中心に普及した。

しかし現在では次に挙げる理由から、日本語対応 TEX として pTEX が使われていることが多い。

- UNIX 用、および UNIX 互換 OS 用の主な日本語対応 TEX 配布形態である ptexlive や ptetex3 が pTEX のみを採用している。
- Microsoft Windows 用の主な日本語対応 TEX 配布形態である W32TEX が pTEX を扱える（NTT JTEX も扱える）。
- pTEX の扱い方を解説する文献の方が、NTT JTEX のものに比べて、出版物と Web 上文書の両方で多い。
- pTEX は縦組みにも対応しているが、NTT JTEX は対応していない。

TEX による組版の作業工程

TEX を利用して組版を行うには、通常次のような作業工程を取る。

1. テキストエディタなどを用いて、文章に組版用命令文を織り込んだソースファイルを作成する。
2. OS のコマンドラインから “`tex FileName.tex`” などと入力して TEX を起動し、`DVI` ファイルを生成させる。
   – ソースファイルにエラーがあれば、修正して再度 TEX を起動する。
3. `DVI` ウェアとよばれる `DVI` 命令文を解するソフトウェアを用いて組版結果を表示し、確認する。
   – `DVI` ウェアには xdvi/xdvik や dviout for Windows などの `DVI` ヴューア、Dvips(k) や dvipdfm/DVIPDFM*x* などのファイル形式変換ソフトウェアなどがある。
   – `DVI` ファイルを `DVI` ヴューアで画面表示または印刷する、あるいは PDF や PostScript に変換して画面表示または印刷することで、組版結果を確認する。
   – 修正の必要があれば、ソースファイルを修正して再度 `DVI` ファイルを作成、確認する。

この間、作業工程が変わるたびにそれぞれのプログラムを切り替えたり、扱う文書が大きいと章ごとにソースファイルを分割して管理したりと、比較的煩雑な作業を伴う。そのため、この工程に係わる各種のプログラムやソースファイルの管理を一元的に行う TEX 用の統合環境がいくつか作成されている。

■ GUI 環境と TEX

GUI は PC の普及に一役買ったが、同時に GUI しか触ったことのない PC 利用者が増加した。そのような利用者が、コマンドラインでの操作を余儀なくされる TEX を非常に扱いづらく感じてしまうのは否めないことである。このため、GUI に特化した TEX 用統合環境もいくつか作成されている。