LuaTeX-ja パッケージ

2011/5/14

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/和文フォントの外部独立管理．
- 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．
- 現時点で **LaTeX** での使用は殆ど考慮されていません．今は **“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．\/を試験的に日本語にも対応させました．アクセントも，make_accent の処理を Lua コードで書けば可能だと思われます．
- 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.
  - 縦組み関連一式．\tate, \tfont, \tbaselineshift, \dtou, ...

■ファイル構成

- luatexja-core.sty: コア部分．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: コア部分に使われる Lua コード．
- luatexja-jfont.lua: 和文フォント定義部の Lua コード．
- luatexja-xkanji.lua: \[x]kanjiskip 自動挿入処理の Lua コード．
- luatexja-rmlgbm-data.lua: 非埋込和文フォント用のデータ（小塚明朝 Pr6N R 由来）．
- luatexja-rmlgbm.lua: 非埋込和文フォント (Ryumin-Light etc.) 定義部．
- mk-rmlgbm-data.tex: luatexja-rmlgbm-data.lua 作成用スクリプト（小塚明朝を luaotfload で読み込んだ時のキャッシュが必要）．

- `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．
- `jfm-ujis.lua`: upTEX-0.30 の `ujis.tfm` ベースのメトリックサンプル．
- `jfm-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
\jfont\tenipam={file:ipam.ttf:jfm=ujis} at 13.5\jQ
\tenipam\parindent=1\zw
\rm\tenipam abcほげほげ）（あいう本文本文……
```

■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 (JFM)」の組である．JFM 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もの（pTEX における JFM ファイルとほぼ同じ内容）であり，`jfm-`⟨*name*⟩`.lua` という名称のファイルに Lua コードの形で書かれている．

和文フォントを使うには，TEX の`\font` primitive と同様の書式を持った次の命令を用いて和文フォントを定義する：

```
\jfont⟨font⟩={⟨font_name⟩:⟨features⟩} ⟨size⟩          % local に定義
\globaljfont⟨font⟩={⟨font_name⟩:⟨features⟩} ⟨size⟩ % global に定義
```

- ⟨*font_name*⟩ の指定について
  内部で luaotfload パッケージを読み込んでいる．大きくわけて，以下の 4 種類がある．このうち，前の 2 つは luaotfload パッケージの機能である．
  - `file:`⟨*file_name*⟩
    TrueType/OpenType フォントをファイル名 ⟨*file_name*⟩ で指定．
  - `name:`⟨*font_name*⟩
    システム内のフォント名を ⟨*font_name*⟩ に指定することも可能．
  - `psft:`⟨*PSfont_name*⟩
    PS フォント名 ⟨*PSfont_name*⟩ を直接指定することもでき，この場合はフォントは名前だけ（非埋込）となる．例えば，本文章では，
    ```
    \jfont\tenmc={psft:Ryumin-Light:jfm=ujis} at 13.5\jQ
    \jfont\tengt={psft:GothicBBB-Medium:jfm=ujis} at 13.5\jQ
    ```
    のような定義をし，標準和文フォント Ryumin-Light, GothicBBB-Medium を用いている．
- **JFM** の指定：⟨*features*⟩ 欄に次を指定する．
  - `jfm=`⟨*jfm_file*⟩: JFM として `jfm-`⟨*jfm_file*⟩`.lua` を用いることを示す．必須．
  - `jfmvar=`⟨*varkey*⟩
    和文文字間の空白の挿入処理は，使用する JFM と，この ⟨*varkey*⟩ の値とのペアによって行われる．
- luaotfload package の他の機能，例えば各種の feature，を用いてもよい：
  ```
  \jfont\tenipam={file:ipaexm.ttf:script=latn;+jp90;jfm=mt}
  ```
  なお，非埋込の場合は，GSUB/GPOS テーブルとして小塚明朝 Pr6N R のものが用いられる（`test01-noembed.pdf` を参照）．
- 不良な JFM を用いた場合，エラーを返すとともに，⟨*font*⟩ の意味は`\relax` となる．

■「和文文字の範囲」の設定

- `\defcharrange{`⟨*number*⟩`}{`⟨*char_range*⟩`}`: 文字範囲の新規定義/追加．
  - ⟨*number*⟩: 文字範囲の番号で，1–216 の整数値で指定．
  - ⟨*range*⟩: 文字コードの範囲を`"100-"200`，`800`，`1701-`のように指定する．
    - ASCII code の範囲 (`0x00`–`0x7F`) は指定できない．
    - 既に他の n 番の文字範囲に使われている領域を指定した場合は，その領域は n 番の文字範囲からは除外される．即ち，後に定義した文字範囲の方が優先度が高い．
  - 定義した文字範囲ごとに，「和文扱い」「欧文扱い」を local に指定できる．

- デフォルトでは，`U+0100` 以降の文字は全部和文扱いであり，さらに文字範囲として，

```
\defcharrange{1}{"80-"FF}
\ltjsetparameter{jacharrange={-1}}
```

  と指定している（つまり Latin-1 Supplement の範囲は欧文扱い）．

■組版パラメタの調整

日本語組版用の各種パラメタの調整には，次の命令を用いる．

```
\ltjsetparameter{⟨key⟩=⟨value⟩, ...}       % local に変更
\ltjglobalsetparameter{⟨key⟩=⟨value⟩, ...} % global に変更
```

⟨*key*⟩ に許される値は次の通りである．

1. ★がついているものは，段落や水平ボックス構成時の値が全体に影響するものである．
2. ◎がついているものは，自動的に global な代入となってしまうもの．

- `prebreakpenalty={`⟨*chr_code*⟩`,` ⟨*penalty*⟩`}`★
  pTEX の`\prebreakpenalty` に対応した設定項目である．
  - ⟨*chr_code*⟩: 文字コードを指定する．一旦補助カウンタに代入されるので，16 進法での数値の指定 (`"abcd`) や，文字トークンによる指定 (‘あ) も可能である．
  - ⟨*penalty*⟩: penalty の値を −10000 から 10000 までの整数で指定する．

- `postbreakpenalty={`⟨*chr_code*⟩`,` ⟨*penalty*⟩`}`★
  同様に，pTEX の`\postbreakpenalty` に対応した設定項目である．pTEX では，同一文字に対して`\prebreakpenalty`, `\postbreakpenalty` の両方を定義することはできなかったが，LuaTEX-ja では可能である．

- `kcatcode={`⟨*chr_code*⟩`,` ⟨*kind*⟩`}`★
  文字コード ⟨*chr_code*⟩ の文字が和文文字扱いされている時，「和文文字の種類」を `0`–`"7FFFFFFF` の自然数値 ⟨*kind*⟩ で指定する．
  - 最下位 bit はウィドウ防止用 penalty の挿入処理に関係する．
  - デフォルトでは，次の 3 つの Unicode の範囲において，`kcatcode=1` としている．
    - `U+2000`–`U+206F` (General Punctuation)
    - `U+3000`–`U+303F` (CJK Symbols and Punctuation)
    - `U+FF00`–`U+FFEF` (Halfwidth and Fullwidth Forms)

- `jaxspmode={`⟨*chr_code*⟩`,` ⟨*mode*⟩`}`★
  pTEX の`\inhibitxspcode` に対応した設定項目である．⟨*mode*⟩ で許される値は，
  - `0`, `inhibit`: 前後の欧文文字との間の `xkanjiskip` 自動挿入を禁止．
  - `2`, `preonly`: 前の欧文文字との間の `xkanjiskip` 自動挿入のみを許可．
  - `1`, `postonly`: 後の欧文文字との間の `xkanjiskip` 自動挿入のみを許可．
  - `3`, `allow`: 前後の欧文文字との間の `xkanjiskip` 自動挿入を許可．

- alxspmode={⟨*chr_code*⟩, ⟨*mode*⟩}★
  同様に，pTEX の\xspcode に対応した設定項目である．⟨*mode*⟩ で許される値は，
  – 0, inhibit: 前後の和文文字との間の xkanjiskip 自動挿入を禁止．
  – 1, preonly: 前の和文文字との間の xkanjiskip 自動挿入のみを許可．
  – 2, postonly: 後の和文文字との間の xkanjiskip 自動挿入のみを許可．
  – 3, allow: 前後の和文文字との間の xkanjiskip 自動挿入を許可．

- yalbaselineshift=⟨*dimen*⟩: pTEX の\ybaselineshift に対応したものであり，欧文文字のベースライン補正量を dimension で指定する．
  – 正の値を指定すると，その分だけ欧文文字は下にずれることとなる．
  – 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 yalbaselineshift だけずれたように見える）．

- yjabaselineshift=⟨*dimen*⟩: 和文文字のベースライン補正量を dimension で指定する．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，この値を適切に調整することで，異なる文字サイズの文字を「上下中央揃え」で組むことも可能である．

- kanjiskip=⟨*skip*⟩★
  和文文字同士の間に入る空白量を指定．pTEX の同名の命令と同様に，基本的には段落/hbox 終了時の値が，その段落/hbox 全体に適用される． つまり，

```
\ltjsetparameter{kanjiskip=3pt}あい%
{\ltjsetparameter{kanjiskip=10pt}うえ}}
```

  の組版結果は（段落終了時の値が 3 pt のため），

あ い う え

  となる．
  但し，kanjiskip の自然長が \maxdimen = 16383.99998 pt の場合は，
  –（和文文字の間それぞれについて）JFM に指定されている kanjiskip の値を採用する．
  – 左側 JFM，右側 JFM のどちらかにおいて指定されていれば，そちらを使用する．
  – どちらにおいても指定されていない場合は，0 とみなす．

- xkanjiskip=⟨*skip*⟩★
  和文文字と欧文文字の間に入る空白量．pTEX の同名の命令と同様．これも，自然長が\maxdimen の場合は，JFM で指定された値を使う．

- jcharwidowpenalty=⟨*penalty*⟩★
  段落において「最後の 1 文字のみが次の行に」くることを抑制するための penalty 値．この penalty は，段落内にある，最後の「kcatcode の最下位 bit が 1 でないような和文文字」の直前に挿入される．

- autospacing[=⟨*bool*⟩]
  和文文字間の glue（kanjiskip）の自動挿入をするかしないかを制御．pTEX では段落/hbox 単位での設定となったが，LuaTEX-ja ではそうではなくなった．挿入可能な箇所の両端のノードにおいて，片方でもこの値が false なら自動挿入は行われる．つまり，例えば

```
あ\ltjsetparameter{autospacing=false}いう
```

  の場合，「あい」の間には kanjiskip が挿入され，「いう」の間には入らない．

- autoxspacing[=⟨*bool*⟩]
  和欧文間の glue（xkanjiskip）の自動挿入をするかしないかを制御．その他は autospacing と同様．

- differentjfm=(large/small/average/both)★◎
  異なる (⟨*jfm*⟩, ⟨*varkey*⟩) である 2 つの和文文字の間の glue/kern の計算方法を設定する．左側文字由来，右側文字由来のものが両方存在した場合にのみ効力をもつ．
  - large: glue/kern の幅が両者のうち大きい方になるように定める．
  - small: 両者のうち小さい方．
  - average: 両者の相加平均．
  - both: 両者の合計値の幅をもつ glue/kern を挿入する．この指定は，JFM 由来の kanjiskip の値が左右の和文文字で異なったときの挙動にも適用される．

- jacharrange={⟨*range_num*⟩,⟨*range_num*⟩,...}: |⟨*range_num*⟩| 番の文字範囲の文字を和文扱いするか欧文扱いするかを設定する．
  - |⟨*range_num*⟩| > 216 の場合，「どの文字範囲にも属さない U+0100 以降の文字」に対しての処理を行う．
  - 正値が指定されたら，|⟨*range_num*⟩| 番の文字範囲は和文文字扱いとなる．
  - 負値が指定されたら，|⟨*range_num*⟩| 番の文字範囲は欧文文字扱いとなる．
  - ⟨*range_num*⟩ = 0 の場合は，単に無視される．

■組版パラメタの取得

日本語組版用の各種パラメタの取得には，次の命令を用いる．

\ltjgetparameter{⟨*key*⟩} or \ltjgetparameter{⟨*key*⟩}{⟨*chr_code*⟩}

戻り値は全て「空白は\catcode=10，それ以外の文字は全て\catcode=12」の文字列である．

⟨*key*⟩ に指定できる値は，説明がない限りは\ltjsetparameter で指定できる ⟨*key*⟩ と同じで，次の通りである，

- kanjiskip, xkanjiskip, yalbaselineshift, yjabaselineshift, jcharwidowpenalty
  それぞれ値を表現する文字列を返す．

- differentjfm: 戻り値は large, small, average, both の 4 つの文字列のいずれか．

- prebreakpenalty, postbreakpenalty, kcatcode, jaxspmode, alxspmode
  これらは各文字コード別に設定される値であるので，文字コード ⟨*chr_code*⟩ を第 2 引数にとる．指定されている整数値を表す文字列を返す．

- jacharrange: 文字範囲の番号 n を第 2 引数にとり，n 番の文字範囲が和文文字扱いされていれば 0，欧文文字扱いされていれば 1 を返す．$n \notin [1, 216]$ の場合は，「どの文字範囲にも属さない U+0080 以降の文字」に対しての結果を返す（しかし，前に述べたデフォルトでの設定のため，実際には U+0100 以降となる）．

- chartorange: 文字コード ⟨*chr_code*⟩ を第 2 引数にとり，それが属している文字範囲の番号を返す．
  - ⟨*chr_code*⟩ が Unicode の範囲外または 0x80 未満ならば，-1 を返す．
  - 上の場合以外で，⟨*chr_code*⟩ の文字がどの文字範囲にも属さない場合は，217 を返す．

■その他の命令

- 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 mm.
- \inhibitglue: 指定箇所での JFM 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）．

JFM 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 Lua ファイルである．見本として，jfm-ujis.lua を入れてある．

```
ltj.define_jfm {
   dir = 'yoko', zw = 1.0, zh = 1.0,
   [0] = {
      align = 'left', left = 0.0, down = 0.0,
      width = 1.0, height = 0.88, depth = 0.12, italic=0.0,
      glue = {
         [1] = { 0.5 , 0.0, 0.5  },  [3] = { 0.25, 0.0, 0.25 }
      }
   }, ...
  [1] = {
      chars = {
         0x2018, 0x201C, 0x3008, 0x300A, 0x300C, 0x300E, 0x3010, 0x3014,
         0x3016, 0x3018, 0x301D, 0xFF08, 0xFF3B, 0xFF5B, 0xFF5F
      },
      align = 'right', left = 0.0, down = 0.0, ...
   }, ...
   [5] = {
      ...,
      glue = {
         [1] = { 0.5 , 0.0, 0.5  },
         [3] = { 0.25, 0.0, 0.25 }
      },
      kern = { [5] = 0.0 }
   }, ...
}
```

全体は，「関数 ltj.define_jfm にテーブルを引数として与える」という構造になっている．以下に，テーブルの中身を述べる．

- 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．
- zw, 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（必須）．通常は両方とも 1.0 となるだろう．
- kanjiskip, xkanjiskip: それぞれ和文文字間グルー量，和欧文間グルー量を

  {⟨*width*⟩, ⟨*stretch*⟩, ⟨*shrink*⟩}

  という形で，フォントサイズ単位で指定する．
- 数字の index i を持った値：i 番の文字クラスについての情報を記述する．
  - glue: 現在の文字クラスの文字の後に挿入する glue を指定する．この項目の内容は

    { [⟨*j*⟩] = { ⟨*width*⟩, ⟨*stretch*⟩, ⟨*shrink*⟩ }, ... }

というテーブルであり，各要素は

i 番の文字クラスの文字と j 番の文字クラスの文字の間に，自然長 $\langle width \rangle$，伸び $\langle stretch \rangle$, 縮み $\langle shrink \rangle$（フォントサイズ基準）なる glue を挿入

という意味である．

– `kern`: 現在の文字クラスの文字の後に挿入する kern を指定する．この項目の内容も，

`{ [`$\langle j \rangle$`] =` $\langle width \rangle$`, ... }`

というテーブルであり，各要素は

i 番の文字クラスの文字と j 番の文字クラスの文字の間に，幅 $\langle width \rangle$ の kern を挿入

という意味である．

– `chars`: 文字クラスに属する「文字」達をリストの形`{...}`で記述する．文字の指定には，Unicode におけるコード番号か，その文字 1 文字だけからなる文字列（’字’ のように）を用いる．どの文字クラスにも属さなかった文字は，0 番の文字クラスに属するとみなされる．そのため，0 番以外の文字クラスではこの項目は必須である．

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．

- `’lineend’`: 行末．この「文字」を 0 以外の文字クラスに設定することで，ぶら下げ組のような組版も可能になる．
- `’boxbdd’`: 水平ボックスの先頭/末尾，段落の先頭/末尾．
- `’jcharbdd’`: 和文文字達の連続とそれ以外のもの（例えば欧文文字）との境界．
- `’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．

以下の **3** 項目は，文字クラスに属する文字が「仮想的な文字」達だけでない場合に必須．

– `align`: `’left’`, `’middle’`, `’right’` のどれかを指定する．

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．そのような場合，`align=’right’` と指定することで，「半角幅の領域に（右詰めで）フォントの実際の字形を入れる」といったことができる．

– `width`, `height`, `depth`, `italic`: それぞれ幅，高さ，深さ，そしてイタリック補正値をフォントサイズに対する割合で指定する．例外として，`width=’prop’` の場合には，その文字クラスは特に幅を定めず，実際のフォントにおける文字幅そのままで出力する．

– `left`, `down`: それぞれ左右，上下方向のずらし量を指定する．`align` 項目のため，`left` はほとんどの場合 0 で良いだろう．

大まかな処理の流れ

LuaTEX-ja パッケージでは，次のような流れで実際の処理を行っている．

- 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` **callback**

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，「\tenrm 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 \ltj@curjfnt 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 \ltj@curjfnt の値に置換．
– p の language field を\ltj@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 \ltj@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）

- **JFM 由来 glue/kern の挿入: pre_linebreak_filter, hpack_filter callbacks**

ここで，JFM 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（を表す node）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 JFM 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 を計算している（\ltjsetparameter 中の differentjfm キーを参照）．
– もうちょっと詳しく書くと,本処理前において,和文文字を表す 2 つの連続した glyph_node Q, P の間には，次の node 達が挿入される：

$$\ldots,\ Q,\ (\texttt{\textbackslash kern}\ w\,\mathrm{pt}),\ (\texttt{\textbackslash penalty}\ p),\ (\texttt{\textbackslash kern}\ (k-w)\,\mathrm{pt}),\ P,\ \ldots$$

上に書いた全ての node が挿入されるとは限らず，また 4 つめの kern も glue に変わる可能性がある．上に出てきた量の意味は次の通りである：

- w: Q が行末にきたときに，行末からどれだけずらすかを指定した量．
- p: Q の行末禁則用 penalty と P の行頭禁則用 penalty の合計値．ウィドウ防止用の \jcharwidowpenalty が挿入される時は，値はここに加算される．
- \kern k: 本来の Q と P の間に入る空き（glue であることも）．w のために自然長を補正している．

- **kanjiskip, xkanjiskip の挿入: pre_linebreak_filter, hpack_filter callbacks**

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．

– 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 -1 の文字」で判定している．
– 合字の周囲の空白挿入については，構成要素の文字列を通じて判断している．例えば，「漢 ffi 字」という文字列に対して，
  - 「漢」と「ffi」間の空白挿入：「漢」と「f」間に入るかで判断
  - 「ffi」と「字」間の空白挿入：「i」と「字」間に入るかで判断

- **ベースライン補正: pre_linebreak_filter, hpack_filter callbacks**

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 \ltj@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\ltj@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，\ltj@yablshift へと値を移し変えているので，この段階では\ltj@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 **callbacks**

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．

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 (**de**vice-**i**ndependent) 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 $25.4/(72.27 \times 2^{16})\,\mathrm{mm}$（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（**C**omprehensive TEX**A**rchive **N**etwork、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/DVIPDFMx 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。