TEX システムの日本語化

倉沢 良一
ASCII Corporation

昭和 62 年 3 月

概 要

現在、TEX システム の日本語化を進めているが、現時点で一応の動作が可能となった。そこで、ここにその報告と共にその内容に付いて解説する。ただし、ここに報告する内容については、決して最終的なものではなく、現時点での暫定的なものであることを承知してもらいたい。尚、本稿はこの日本語化された TEX システムで LATEX を用いて作成した。

目 次

1 TEX システム日本語化の方針 3

2 TEX 日本語化の実際 4

2.1 コード体系 4
2.2 プリミティブ 4
2.2.1 プリミティブの追加 4
2.2.2 プリミティブの拡張 4
2.3 ラインブレーク 4
2.4 禁則処理 6
2.4.1 禁則処理でのペナルティの応用 6
2.4.2 禁則テーブル 6
2.5 スペーシング 7
2.5.1 和文組版におけるスペーシング 7
2.5.2 TEX でのスペーシング 8
2.6 フォント 9
2.6.1 TEX での日本語フォントの取り扱い 9
2.7 TFM ファイルと char ノード 10
2.7.1 TFM ファイルの拡張 10
2.7.2 char ノードと JFM ファイルの対応付け 10
2.8 数式中での日本語の使用 11

3 今後の予定 12

A 禁則ペナルティの設定例 13

1 TEX システム日本語化の方針

TEX システムを日本語化するにあたり、次の事項に留意しその方針を決定した。

1. 完全なアッパーコンパチブルであること。
   - 日本語化により本来 TEX の持つ機能が損なわれるようなことがあってはならない。
   - 欧文、和文の区別が無く、同一のシステム（プログラム）で扱えること。当然これまでにある、又はこれから作られる欧文の原稿が扱えること。
   - 和文に対しても、欧文同様の機能が扱えること．（フォント、マクロ等）
2. 和文特有の特殊処理が行えること。
   - 行頭、行末禁則を自動的に行えること。
   - 和文、欧文間のスペーシングの調整等が自動的に行えること。
3. 仕上がりが美しいこと。
   TEX 本来の目的である “美しい出力” が得られなければ、日本語化を行う意味は全く無くなってしまう。高性能の出力デバイスと高品位のフォントを用いることにより、写植機に負けない出力が得られること。

1. に関しては、あくまでも “TEX システム” をそっくりそのまま日本語環境の下で使いたいという要望である。和文に対しても、欧文のように自由にフォントを切り換えたり、マクロ定義が行いたいというのは当然であり、それが TEX のような文書処理システムの魅力の 1 つであるからである。
2. に付いては、和文を扱う場合には必然的に付いてまわる問題で、これらの処理が確実に行われなければ、その文書処理系の利用価値は半減してしまうと考えられる。
3. の内容は少し大げさなようにも思えるが、Don Knuth の TEX 開発の目的からすれば、至極当然のことでもある。又、これは 1. や 2. の内容が大きく関わっているのは言うまでもないことである。

このような要求を、出来る限り満足するために次のような日本語化の方策をとることにした。

- 日本語化は、TEX そのもの (initex、virtex) に対して行い、plain や lplain 等のマクロには出来れば一切手を付けない。（付けたくない。）
- ただし、追加された機能の為のイニシャライズや、パラメータの設定、漢字フォントの定義等は、TEX のフレキシビリティを保つためにもマクロで行えるようにする。
- 欧文で行えることは、可能な限り和文でも行えるようにポーティングする。

2 TEX 日本語化の実際

上記の方針に基づき日本語化を行った結果、現時点で以下のように実現されている。

2.1 コード体系

インプリメントの容易さ、現在使用しているホストマシンのコード、将来のパーソナルコンピュータへの移植等を考え合わせた結果、内部コード、外部コード共に ASCII、シフト JIS の組み合せを用いた。ただし、DVI ファイル内の漢字コードだけは、JIS コードを出力するようになっている。また半角カナのサポートは行っていない。

2.2 プリミティブ

日本語化に伴い、幾つかのプリミティブの追加および拡張を行った。これらは、禁則処理、オートスペーシングおよび漢字コードのためのものである。この時、新しく追加するプリミティブについては、日本語独特の新しい処理に対するものだけに止め、出来るだけ既存のプリミティブを拡張するように努めた。

2.2.1 プリミティブの追加

Table 1 に示すプリミティブが追加されている。(詳しくは、以降のセクションで解説する。)

2.2.2 プリミティブの拡張

既存のプリミティブに関しては、出来得る限り欧文、和文の区別をせずに扱えるよう拡張を行った。また内部で扱うトークンに対する日本語化を行うことによって、プリンティングキャラクタのみでなく、マクロ名やトークンリストにも全角文字を半角文字と混在して扱えるようになっている。
例.

```
\char”82A0
\accent”5F‘あ
\def\ 改ページ {\eject}
```

2.3 ラインブレーク

欧文と和文の処理で見かけ上最も大きな違いは、ラインブレーク処理であると思われる。“見かけ上” と言うのは、TEX 内部の処理にはさほど大きな変更はいらないからである。具体的には、次のように文中のどの位置をブレークポイントの対象とするかの問題である。

prebreakpenalty	指定するキャラクタコードの前方に挿入するペナルティの値を設定する。行頭禁則の指定として用いる。 例. \prebreakpenalty‘、=1000
postbreakpenalty	指定するキャラクタコードの後方に挿入するペナルティの値を設定する。行末禁則の指定として用いる。 例. \postbreakpenalty‘［=1000
jis	漢字コードの変換を行うためのプリミティブで、JIS コードから内部コードであるシフト JIS に変換する。 例. \prebreakpenalty\jis"2122=1000
kuten	\jis と同様に区点コードからシフト JIS へ変換する。 例. \postbreakpenalty\kuten"0146=1000
sjis	上記の２つと同様であるが、内部コードがシフト JIS であるので、そのままの値をかえす。
kanjiskip	全角コード間に自動的に挿入されるグルーを持つレジスタ。 例. \kanjiskip=0pt puls .3pt minus .3pt
xkanjiskip	全角–半角コード間に自動的に挿入されるグルーを持つレジスタ。 例. \xkanjiskip=2.5pt puls 1pt minus 1pt
autospacing	全角コード間へのグルーの自動挿入を行うための指定。
noautospacing	全角コード間へのグルーの自動挿入を行わないための指定。
autoxspacing	全角–半角コード間へのグルーの自動挿入を行うための指定。
noautoxspacing	全角–半角コード間へのグルーの自動挿入を行わないための指定。
xspcode	全角–半角コード間へのグルーの自動挿入の対象となる半角コードを指定する。現在デフォルトの値としてアルファベットおよび数字に対してスペースが挿入されるように設定されている。 例. \xspcode‘A=0

表 1: 日本語化に伴い追加されたプリミティブ。

- 欧文中でのラインブレークは、ハイフネーション処理等の特別な場合を除いて、単語中つまり連続する文字列中は、ブレーキングポイントとして選択されない。
- 和文中では禁則の例外を除いて、全ての文字間がブレークポイントの対象になり得る。

実際に TEX 内部ではパラグラフ単位でラインブレークの処理が行われ、それは *line_break* というルーチン内で、ブレーク可能な箇所を捜し出し、それを *try_break* という関数に渡すことにより行われている。(勿論こんな単純なものではないが、基本的な操作としてこうした形になっている。詳しくは、TEX: The Program PART38、39 Breaking paragraphs into lines. を是非読んでみることをお勧めする。) そこで、和文中の文字間もブレークポイントの候補として、*try_break* に渡すよう変更した。

又、本来ならばラインブレークの処理は、次項の禁則処理と密接な関係がありとても分けて考えられるものではない。しかしながら、TEX では事実上完全に分けて考えても差し支えがないような構造になっている。それは、ペナルティと呼ばれるブレーキング

のための一般的な評価値が導入されているためであり、ブレーキングルーチンではこの値を如何に評価するかだけを考えれば良く、禁則処理ルーチンでは、この値を如何に設定するかだけを考えれば良い。

2.4 禁則処理

2.4.1 禁則処理でのペナルティの応用

前のセクションでも簡単に触れたように、禁則処理も TEX にとって全く新しい概念ではなく、ペナルティと言う形で導入されている。これは、上記のラインブレークやページブレーク処理時に、その箇所がブレークポイントとして、どの程度適切（負値のペナルティ）であるかあるいは不適切（正値のペナルティ）であるかを示すための評価値である。したがって、ただ単にその箇所でのブレーキングを禁止するだけでは無く、強制的なブレーキングや、禁則文字が連続するような場合の優先順位付けにも利用できる。

日本語の禁則処理にこのペナルティを応用するためには次のようにすればよい。

行頭禁則 禁則文字の前に正のペナルティを付ける。

行末禁則 禁則文字の後ろに正のペナルティを付ける。

こうした形でペナルティを自動的に挿入さえできれば、後は既存の TEX のルーチンをそのまま利用できるはずである。

2.4.2 禁則テーブル

このペナルティを自動的に挿入するために、TEX 内部に禁則文字とその文字に対応するペナルティ値とペナルティの挿入位置（その文字の前に挿入するか後に挿入するかの別）のテーブルを持つことにした。そしてこのテーブルにこれらの情報を登録する手段として、**\prebreakpenalty** と **\postbreakpenalty** の 2 つのプリミティブを追加した。**\prebreakpenalty** は指定した文字の直前にペナルティを挿入することを、**\postbreakpenalty** は指定した文字の直後にペナルティを挿入することを指定するもので、全角文字、半角文字の区別無しに指定できる。これらは、同一の文字に対して同時に両方のペナルティを挿入するようには指定できない。 もし同一の文字に対して双方の指定がされた場合、後からのものに置き換えられる。勿論これらは “{}” による、グルーピング規則が適用される。又、双方で指定できる文字数は最大で 256 文字までである。この値は、全角文字、半角文字を合わせて禁則対象となると考えられる文字数を、十分にカバーしていると思われる。

この禁則テーブルからの登録の削除は、ペナルティ値 ‘0’ を設定することによって行われるが、グローバルレベルでの設定でなければ、このテーブルの領域は解放されないことに注意して欲しい。なぜなら、ローカルレベルで領域を解放してしまえば、その外側のレベルに戻ったときに、そのレベルでの値を最設定するための領域が確保されている保証がないからである。

このテーブルができれば、各文字を読み込む度に、その文字がこのテーブルに登録されているかどうかを調べ、登録されていればそのペナルティを文字の前後適切な位置に

挿入して行けば良い。この禁則テーブルの検索にはハッシュ法を用いており、ハッシュ関数として

$$\begin{cases} (C2 + (C2 << (C1 - 0x81)))\, mod 256 & \text{漢字コード} \\ & C1 = \text{上位バイト}、C2 = \text{下位バイト} \\ \\ C * 2 + 1 & ASCII\ \text{コード} \end{cases}$$

を用いている。これは、おそらく禁則の対象となるであろう、全角の記号、ひらがなカタカナの小文字、ASCII コードの記号等を適当に散らばらせるように考えてある。又、このテーブルには最大 256 文字まで登録可能であるが、むやみに多くの文字を登録してテーブルを満たすことは、禁則文字検索のオーバーヘッドを大きくし、パフォーマンスに大きな影響を与えるので注意する必要がある。

こうしてペナルティを適切に挿入さえしてしまえば、後は TEX のラインブレークルーチンにおまかせしてしまえる。勿論、このテーブルにいかにペナルティを登録するかは、ユーザーの責任であると共に、ペナルティ値の変更により自由に禁則処理に変化をつけることができる。

2.5 スペーシング

これまでの変更で一通りの和文を出力することは可能である。ところがこれだけではまだ不十分で、このままでは決して美しい出力は得られない。なぜなら、欧文では単語毎にスペースが挿入され、そこでのスペーシング量を調整することにより、ジャスティファイケーションの処理が行われていた。ところが、これまでの変更だけでは、和文に対してそのような調整を行う部分がほとんど無いのである。この状態がいったいどのようなものであるか試しに、この段落を上記の*\noautospacing*と*\noautoxspacing*によって、スペーシングの自動挿入をしないで組んでみた。この操作ににによって、この段落中にとられるスペースはピリオド、カンマの後のスペースのみである。

一目して解るように、行末に 1 文字以上の不揃いが出来てしっまている。これは、パラグラフ全体をダイナミックにブレーキングするといった、TEX の長所が悪い方に働いてしまい、数行先の禁則処理の影響がそのまま反映されているのである。禁則ペナルティのパラメータの値の悪さも手伝ってか、悲惨な結果となってしまっている。

2.5.1 和文組版におけるスペーシング

これに対して一般に行われている和文の組版では、行単位のみでのブレーキングが行われており、最悪の場合でもこのようなことは起こらないようである。また組み幅は、全角の正数倍にとられるため日本語のみの行では必ず行末がきれいに揃う仕掛になっている。ただし、禁則が生じたり、途中に欧文が挿入されたりした場合は、やはりどこかで調整が必要になってくる。こうした調整は次のような箇所で行われているようである。

- まずこうした調整に迫られた場合、追込みを考える。追込みのために、カンマ、ピリオド、括弧等の文字を半角幅にまで縮めてみる。

- それでも調整しきれず、行中に欧文が存在する場合は、欧文間、和文–欧文間のスペースを調整する。
- さらに調整が必要な場合は、かな文字間のスペースを調整する。
- これでもまだ駄目な場合、追い出しを試みる。

説明が遅れてしまったが、和文–欧文間には見栄えを良くするために、4 分開けと呼ばれる全角幅の 1/4 のスペースが挿入される。実際問題としては、禁則文字が連続でもしない限り、行中に 2 つ以上のカンマ、ピリオドの類の文字が存在すれば、最初の段階で調整は付いてしまう。

2.5.2 TEX でのスペーシング

これを TEX に応用することを考えてみる。

まず組み幅の問題であるが、TEX の場合、何文字幅といった指定も簡単に実現可能ではあるが、既存のマクロ等の利用を考えると組み幅ばかりでなく、インデント等の組み幅に関係する全てのパラメータも同時に変更する必要があり、出来れば自由な組み幅の指定が出来た方が良い。そうなると、ピリオド、カンマ等の文字が一文字も表れないような行に対しても組み幅を調整する手段が新たに必要となる。しかも、行中にそう何文字も表れないような文字に、こうした帳尻合わせを集中するのは、あまり美しいことではない。

次に、和文–欧文間のスペース等は原稿中でいちいち気にしながら挿入するのでは非常に面倒である。 こうした類のものは折角コンピュータを使用しているのであるから、自動的に行ってほしい。

このような要求を満たすために次のような変更を加えることにした。

1. 全角文字間にもグルーを自動的に挿入できるようにする。これによりピリオド、カンマ等の文字が表れない行でもジャスティファイケーションが行える。またカンマ等の持つスペースだけでなく、行全体でその調整分を分割するため、より自然な組みが実現できる。勿論、ここで行う調整は 0. 数 pt といった極めて微量の調整である。さらにこのグルーの導入により、1 歯詰めなどといった組み方も可能になる。
2. 和文–欧文間のスペーシングは当然 TEX が自動的に行ってくれるものだと考え、そのまま導入した。ただし、全ての和文–欧文間（全角文字–半角文字間）に挿入して良いわけではない。例えば、半角のピリオド、カンマ、括弧類記号と全角文字間には余計なスペースは入れない方がよい。
3. オリジナルの TEX では、改行キャラクタを読み込むことにより単語の切れ目と判断し、スペースが挿入される。しかし、和文ではこうしたスペースを挿入されるのは不都合である。 そこで全角文字の後の改行はただ単に読み飛ばすだけの処理に変更する必要がある。

1. の実現方法として、**\kanjiskip** というグルーレジスタを新たに設け、このグルーを全角文字間に挿入するようにした。ただし、実際に他のグルーと同じように処理したのでは、全角文字 1 文字に付き 1 グルー分のメモリを消費することになり、余りにもメモ

リ効率が悪くなる。 そこでこのグルーに関しては、ラインブレークや、ボックスへの組み込みルーチン (*hpack*) 内での組み幅の計算だけで処理するようにしてある。

2. に関しては、TEX が数式中で行っているように、必要な箇所に実際にグルーを挿入する。 挿入するグルーは、1. と同じように **\xkanjiskip** というレジスタを設け、その値を使う。さらに、全角文字の前後に現れる半角文字の内の何れに対して、スペースを挿入すべきかを指定するために、内部に 128 文字分のテーブルを設け、そこに登録されている内容によってスペース挿入の有無を決定している。現在デフォルトでアルファベットおよび数字がスペース挿入の対象となっている。 このテーブルへの登録は、**\xspcode** というプリミティブによって行う。各 ASCII キャラクタに対して 0 を設定することにより、全角との間のスペースの挿入を禁止し、それ以外の値で許可するようになっている。

尚、1.2. に共通する処理として、これらのレジスタの値は、そのパラグラフ、あるいは *hbox* に入った時点での値を保持して使用し、その中でどんなに変更を加えても処理には影響が及ばないようになっている。又、これらの処理は、**\autospacing**、**\noautospacing**、**\autoxspacing**、**\noautoxspacing** によって ON、OFF できるようになっている。

3. は TEX のスキャンニングルーチンを変更することで、スペーシングキャラクタとして、先のルーチンに渡って行かないようにした。

2.6 フォント

フォントに関しても、できるだけ欧文のものと同じ扱いが出来るように配慮した。

2.6.1 TEX での日本語フォントの取り扱い

ユーザインタフェースからみた日本語フォントの取り扱いは、*\mathcode* 等の数式関係のフォントの定義中には含めることができないことを除くとほとんど同じようにして扱える。異なる点は、半角文字と全角文字とを使い分ける度にフォントの切り換えを行うのは非常に繁雑なので、カレントフォントとしてこれまでの半角文字用の他に全角文字用も独立して持つようにしたことである。これによって特にフォントの変更を行う必要がなければ、全角、半角の区別を意識することなしに原稿を入力することが出来る。具体的には次のように指定すれば良い。

```
\font\jfont=min10
\jfont
```

これを見てわかるように、これまでと全く同じようにして指定できる。この時指定された \jfont が漢字フォントであれば、漢字用のカレントフォントを変更する。漢字フォントと英字フォントの判別は、TFM ファイルにより行っている。

又、数式中でのフォントを指定する、ファミリーに付いても同様に、全角用のカレントファミリーを拡張した。これにより、全角文字に対しても、上付き文字や下付き文字の取り扱いが可能になる。勿論、半角、全角を意識する必要はない。ただ全角用カレントファミリーが半角用と異なる点は、半角用のカレントファミリーは、マスモードに切り

替わった時点で-1 にセットされるのに対し、全角用のカレントファミリーは、そのままの値を保つことである。(マスモード中でのフォント切り換えのメカニズムに付いては、TEXbook p.154 等を参照のこと。) これは、数式関係のフォント定義を日本語フォントにまで拡張していないための措置であるが、数式用フォントに関しては、全角文字に拡張してもほとんど意味がないので、こうした形をとった。

2.7 TFM ファイルと char ノード

TEX 内部では印字データを char ノードと呼ばれるノード中にそのフォントの種類と共に格納する。又、TEX システムには、TFM ファイルと呼ばれるファイルがフォントの種類毎に用意されており、そこに各フォント内に含まれるキャラクタの、各種サイズ等の情報が置かれている。

TEX のラインブレーク等の処理は、この char ノード中のキャラクタデータから TFM ファイルの字幅情報等を検索して処理する仕組みになっている。したがって、TEX を日本語化する際には、これらをどのように拡張し、どのように対応付けるかがキーポイントとなる。

2.7.1 TFM ファイルの拡張

まず TFM ファイルであるが、この中には *char_info* と呼ばれるそのフォント中に存在するキャラクタのコードと 1 対 1 に対応するテーブルが置かれている。このテーブル中には *width*、*height*、*depth* 等の他のテーブルへのインデックスが設定されており、このインデックスを用いてそれぞれのパラメータを参照する仕組みになっている。

ただし、このインデックスのサイズから、1 フォント内に含まれるキャラクタのサイズの種類が制限されてしまっている。しかし、漢字フォントは 1 フォント内の数は多いが、そのサイズは記号等を除いてほとんど同じであり、この制約内に十分に収まると判断した。そこでこの *char_info* とその他のテーブルの仕組みは、そのまま利用することにし、*char_info* テーブルと漢字の 2 バイトコードを、対応付けるための方策を考えることにした。

その手段として漢字コードと *char_info* へのインデックスを持つテーブルを付け加えることにした。ところが、漢字フォントは 1 フォント中に約 7000 文字もの数があり、そのままテーブルに持つのは得策ではない。この解決策として、1 フォント内のキャラクタの大部分が同じサイズを持つという漢字の特徴から、デフォルトサイズを規定し、これと異なるものだけを、そのコードと *char_info* へのインデックスの対という形でテーブルを持つようにした。さらに、検索のオーバヘッドを軽減するために、テーブルを漢字コード順にソーティングした形で持つように規定した。

又、従来の TFM ファイルと区別するためにサフィックスを "**JFM**" と変更した。

2.7.2 char ノードと JFM ファイルの対応付け

char ノードは、上記したとおりそこにフォントとキャラクタコードを持ち、さらに次のノードへのポインタを含んでいる。これを日本語のために拡張するためには、ノード

サイズを変更するか、複数のノードによって 1 つのキャラクタを表現するかの 2 通りが考えられる。処理の効率から考えると、前者の方がより良いのであるが、そのためには TEX のメモリ管理ルーチンの大変更をしなければならず、今回は断念した。

ここで採用した方法は、TEX: The program にも簡単に書かれていたとおり、連続する 2 つの char ノードを使って日本語 1 文字を表すものである。具体的には、2 つ目のノードにキャラクタコードを格納し、1 つ目のノードには、フォントと TFM の *char_info* へのインデックスを格納するようにしてある。つまり、char ノードを生成する段階で、TFM ファイルに拡張したテーブルから、*char_info* へのインデックスを検索してしまうのである。こうすることにより、この 1 つ目の char ノードは、既存の欧文キャラクタの char ノードと全く同一に扱える。2 つ目のノードは、最後に DVI ファイルに出力する時や、エラーが生じた場合に必要になるだけで、一般の内部処理では読み飛ばしてやりさえすれば良い。

ある char ノードが日本語用のものであるかどうかは、char ノード内のフォントを調べ、それが漢字フォントであるかどうかによって判断する。この処理を簡単に能率良く行うために、フォントが漢字かどうかのテーブルを TEX 内部に設けてある。

2.8 数式中での日本語の使用

フォントの項でも簡単に触れたとおり、数式中においても日本語が扱えるように変更を行ってある。これは、数式中に直接日本語が含まれるような使い方は希であるが、2.4.2 中の式のように、簡単な説明等を付け加えたいことは良くあるからである。

3 今後の予定

現時点までの TEX の日本語の状態を一通り表面的な部分のみであるが解説してみた。勿論、まだ不十分な点は多々あるが、実際にこの原稿を作ってみて、十分使用に耐えることは確信できた。特に LATEX がそのまま日本語で使用できることは、感激に価する。今のところ、LATEX のコマンドで使用できないものは無いようである。

今後の予定として次のようなことを考えている。

- 自動的に挿入される、全角間および全角–半角間のスペースは、**kanjiskip**、**xkanjiskip** の値が必ず使われてしまうが、これは JFM ファイル中に、デフォルト値を設定出来るようにして、フォントの変更と共に最適値を使用できるようにしたい。
- 全角の記号等は、全角幅固定ではなく、基本的にはその文字の持つ実際の幅で取り扱えなければ、行頭や行末におかれる場合、行ぞろえが正確に行えず美しくない。こうした記号類は漢字フォントであっても、独自の字幅情報を持つようにしたい。
- 全角文字に対しても、文字の組み合せによるカーニングを行うべきである。 特にひらがなカタカナの後に、ピリオド、カンマの類の文字が続いた場合の、スペースの空き方が不自然なことが場合があり、調整できるようにする必要がある。

A　禁則ペナルティの設定例

禁則ペナルティの設定例として、このテキスト制作時に使用した設定を上げておく。

```
\prebreakpenalty‘.=1000    \prebreakpenalty‘,=1000    \prebreakpenalty‘}=1000
\postbreakpenalty‘{=1000   \prebreakpenalty‘)=1000    \postbreakpenalty‘(=1000
\prebreakpenalty‘]=1000    \postbreakpenalty‘[=1000   \postbreakpenalty‘!=500
\postbreakpenalty‘#=500    \postbreakpenalty‘$=500    \postbreakpenalty‘%=500
\postbreakpenalty‘&=500    \postbreakpenalty‘≒500     \prebreakpenalty‘|=500
\prebreakpenalty‘;=500     \prebreakpenalty‘?=500     \prebreakpenalty‘:=500
\prebreakpenalty‘、=1000   \prebreakpenalty‘。=1000   \prebreakpenalty‘, =1000
\prebreakpenalty‘. =1000   \prebreakpenalty‘・=1000   \prebreakpenalty‘：=500
\prebreakpenalty‘；=500    \prebreakpenalty‘？=500    \prebreakpenalty‘！=500
\prebreakpenalty‘） =800   \postbreakpenalty‘（=800   \prebreakpenalty‘} =800
\postbreakpenalty‘ ｛=800  \prebreakpenalty‘［=800    \postbreakpenalty‘］=800
\postbreakpenalty‘ ‘ =1000 \prebreakpenalty‘ ’ =1000 \prebreakpenalty‘ー=200
\prebreakpenalty‘＋=200    \prebreakpenalty‘－=200    \prebreakpenalty‘＝=200
\postbreakpenalty‘＃=200   \postbreakpenalty‘＄=200   \postbreakpenalty‘％=200
\postbreakpenalty‘＆=200   \prebreakpenalty‘ぁ=150    \prebreakpenalty‘ぃ=150
\prebreakpenalty‘ぅ=150    \prebreakpenalty‘ぇ=150    \prebreakpenalty‘ぉ=150
\prebreakpenalty‘っ=150    \prebreakpenalty‘ゃ=150    \prebreakpenalty‘ゅ=150
\prebreakpenalty‘ょ=150    \prebreakpenalty‘ァ=150    \prebreakpenalty‘ィ=150
\prebreakpenalty‘ゥ=150    \prebreakpenalty‘ェ=150    \prebreakpenalty‘ォ=150
\prebreakpenalty‘ッ=150    \prebreakpenalty‘ャ=150    \prebreakpenalty‘ュ=150
\prebreakpenalty‘ョ=150
\prebreakpenalty\jis"212B=1000  \prebreakpenalty\jis"212C=1000
\prebreakpenalty\jis"212D=1000  \postbreakpenalty\jis"212E=1000
\prebreakpenalty\jis"2139=250   \prebreakpenalty\jis"2144=250
\prebreakpenalty\jis"2145=250   \postbreakpenalty\jis"2146=1000
\prebreakpenalty\jis"2147=1000  \postbreakpenalty\jis"2148=1000
\prebreakpenalty\jis"2149=1000  \postbreakpenalty\jis"214C=800
\prebreakpenalty\jis"214D=800   \postbreakpenalty\jis"2152=800
\prebreakpenalty\jis"2153=800   \postbreakpenalty\jis"2154=800
\prebreakpenalty\jis"2155=800   \postbreakpenalty\jis"2156=800
\prebreakpenalty\jis"2157=800   \postbreakpenalty\jis"2158=800
\prebreakpenalty\jis"2159=800   \postbreakpenalty\jis"215A=800
\prebreakpenalty\jis"215B=800   \prebreakpenalty\jis"246E=150
\prebreakpenalty\jis"256E=150   \prebreakpenalty\jis"2575=150
\prebreakpenalty\jis"2576=150
```