後　書

森本英夫先生の「古稀」に際し、お祝いの意味で「論集」をまとめようという話は、ずいぶん以前から仲間内で出ていた。先生のご内諾を頂くべくご相談したところ、「仰々しいことは嫌」と断られ、それなら先生の教え子たちが論文を発表する場の「名目」にとお願いし、お許しを頂いた。

本来ならば、刊行本として「古稀記念論文集」を捧げるべきところであるが、先生のお気持ちもあり、同人誌『周辺』と『TLLMF』の合併号と相成った。諸般の事情からも、同人全員が論考を提出できなかったが、とにもかくにも16名の論文が集まった。

これら二誌を創刊された先生のお気持ちからすれば、どんな事情があるにせよ、学問の道を行こうとするなら、論文の一本や二本、いつでも出せるよう準備しておくべきであったであろう。また当初の刊行期日が延び延びになったのも、研究者としてはいかがなものであろうか。

これからは『周辺』『TLLMF』の二誌が、創刊当初の、本来の姿に戻り、同人の研究成果発表の場であり続けられるよう努めることを、先生にお約束し、ご寛恕を賜りたいと願っている。

2004年5月

編集子　平山　弓月

福島　祥行

森本英夫古稀記念
『周辺』『TLLMF』合併号　　定価1000円

2004年5月31日　発行

発行所　シメール社
〒597-0062 貝塚市澤1009-6
TEL. 0724-31-8085

印刷所　協和印刷株式会社
〒615-0052 京都市右京区西院清水町13
TEL. 075-312-4010

『周辺』『TLLMF』合併号　目次

文　化

電子テキストの書式

——より快適な電算処理のために——

小栗栖　等

はじめに

高品質な電子テキストとは何か。それが本稿の主題である。ただし、ここでいう「電子テキスト」とは、写本・手稿をオリジナルとした電子校定本をも含むテキストファイルである。また、「品質」とは、オリジナル再現の忠実性ではなく、電算処理への適性、コンピュータ・プログラムとの親和性を意味する[1)]。

電子テキストと電算処理

コンピュータは自ら情報を創り出すことはできない。いかに複雑な処理を行った場合でも、事前に与えられたデータから特定の情報を取り出すに過ぎない。たとえば、ある単語の頻度数を調べる場合、頻度数そのものが電子テキストの中に書かれているわけではない。だが、頻度数に応じた個数の単語があらかじめ入力されていなければ、コンピュータにはなすすべがない。これは、常識の域であろう。しかし、コンピュータで単語の頻度数を調べるのが、極めて困難な作業だと言うと、多くの人は驚くのではないだろうか。

単語という概念は、コンピュータには存在しない。ワープロやエディタには検索機能があるが、単語を検索してはくれない。bas を検索した場合、bas だけではなく、bascule, abasourdissement, cabas などもヒットするのは、周知のとおりである。もっとも、正規表現では、「単語の区切り」という便利な概念が利用できる[2)]。単語の区切りとは、句読点やスペース、改行、タブなどアルファベ文字や数字以外を意味し、正規表現では \b と表現される。そこで、haubert だけを探したい場合には、\bhaubert\b とすれば良い。しかし、それで見つかるのは、haubert という文字のシーケンスであって、単語ではない。たとえば、hau-bert とハイフネーションされた場合があるとすれば、別の手間が生じ、正規表現は突如、\bhaubert\b|\bhau-\nbert\b という複雑なものとなる。haubert が連続している場合と、分割されている場合とを分けて処理しているのである。ハイフンの後ろにスペースやタブが存在する（もしくは存在し得る）場合に

は、さらに、\bhaubert\b¦\bhau\W＋\bert\b とする必要があり、この表現になる理由を端的に説明するのはもはや不可能になる。そして、複数形をも考慮した場合、正規表現は ¥bhauberts?¥b¦¥bhau¥W＋berts?¥b となる。

正規表現で数えられるなら問題はないではないか、と思う人もあるかも知れない。しかし、動詞や形容詞の不規則変化なども考慮すれば、正規表現は膨大な長さになる。それに、suis が être なのか、suivre なのかを見分けるような正規表現は絶対に作れない。ボタン一つ押せば、全語彙頻度数データを作ってくれるようなソフトは、存在しえないのである。

さて、ここで確認しておきたいのは、もし、ハイフネーションが使われていなければ、haubert の検索は、\bhauberts?\b という半分の長さで事足りたということである[3]。コンピュータは電子テキストに内在する様々な情報を取り出すが、その処理が単純になるかいなかは電子テキストの作り方で大いに変わる。電子テキストの書式が問題となるゆえんである。

むろん、コンピュータの都合に、闇雲に合わせろ、と言いたいわけではない。たとえば、複数形をテキスト内で使わなければ、正規表現はもっと簡単になる。しかし、それでは、テキストの本質そのものが損なわれる。一方、ハイフネーションは、限られた紙面への印刷という制約の結果、つまり、印刷機に都合に合わせた結果に過ぎない。ならば、これを取り除いて、電算処理を単純化したところで何の問題もない。もちろん、印刷技術は、人間の側の都合に合わせ、快適な読書を実現するために発展してきた。だが、電子テキストは読書のためにあるのではない。読書は手許の書物ですれば良い。電子校訂本に関しても、電算処理に適したテキストファイルならば、LaTeX を利用して市販の書籍のように組版するのはいとも容易い。電算処理と読書という別次元の事柄を同一地平に並べるべきではない[4]。

質の低い電子テキスト

I¶
·····A·icel·jor·ke·la·dolor·fu·grans¶
·····Et·la·bataille·orible·en·Aliscans,¶
·····Li·quens·Guillaumes·i·soufri·grans·ahans.¶
·····Bien·i·feri·li·palasins·Bertrans,¶

5····Gaudins·li·bruns·et·Guichars·li·aidans,¶
·····Girars·de·Blaives, ·Gautier·li·Tolosans,¶
·····[Hunaut·de·Seintes]·et·Foukiers·de·Mellant.¶
·····Sor·tos·les·autres·s'i·aida·Vivians : ¶
10···En·. XXX. ·leus·fu·rous·ses·jazerans,¶
(以下、省略)(·はスペース文字を¶は改行文字を表す)[5]

上記の電子テキストは見た目には、非常に美しい。ほぼ、印刷本の体裁になっている。しかし、電子テキストとしての質は非常に低い。

まず、詩節番号を消去するのが、かなり困難である。人間には簡単に見分けられる、ローマ数字表記の番号は、コンピュータには一大障害となる。上の詩節番号を表現する正規表現は、^M*C*M*C*D*C*X*C*X*L*X*I*X*I*V*I*$ (MMMCMXCIX 以上の数字はコンピュータでは簡単に表現できない) であるが、

(1) ローマ数字が常に行頭に位置している
(2) ローマ数字の末尾には常に改行しかない
(3) 本文中にローマ数字のみの行がない
(4) ローマ数字には大文字のみが使用されている

といった条件を満たす場合にしか利用できない。これらの制限への違反が多ければ多いほど、先ほどの haubert の場合と同様、正規表現は長くなっていく。場合によっては、正規表現で対処しきれなくなる。

第二に、行番号が厄介である。本文中にもアラビア数字が使われていた場合、通常の一括置換で行番号をとりのぞくことはできない。正規表現では、^\d+\s+|^\s+を使うことで、行頭の余分なスペースもろとも行番号を消去できる。だが、番号が括弧などで括られている場合や、複数行に同一行番号を与えるために、1 a, 1 b などとなっている場合には、この表現は利用できない。

第三に、行番号と本文の間のスペースの数が一定しないのも困る。たとえば、行番号と本文の間をタブで区切りたいと思った場合、通常の一括置換での

対応はほぼ不可能である。正規表現では^(\d＋)\s＋|^\s＋を$1\t に置換することで、行番号と本文をタブで仕切り、行番号のない行の冒頭のスペースもタブに置換できる。だが、先に述べた行番号の不規則表現があった場合には、利用できない。

最後に、第六行から第十行までの間に四行しか詩行がないのも問題となる。もちろん、これは入力ミスの例ではない。原本のナンバリングにミスがあったり、写本で行脱落があったりすると、上のようなことが起こる。つまり、行の脱落そのものは電子テキスト作成者の責任ではない。問題は、行番号と行の実際の位置との乖離である。こうしたずれがあると、多くの電子テキスト処理ソフトは正常な動作を望めない。たとえば、ソフトを使って索引を作っても、参照番号が電子テキストの行番号と一致しなくなる。こうしたずれを解消するには、スクリプトを使うか、つらい手作業を行わねばならない。

上記の通り、多くの問題は正規表現で対処できる。しかし、だからといって、問題を放置して良いということにはならない。ある電子テキストが、どのような問題を抱えているのかを知るためには、電子テキストを詳細に検討しなければならない。そのうえ、問題解決には、正規表現を、個別に作りあげることになる。入力時に電算処理に適した書式を採用してさえいれば、これらの作業は生じてこないのである。電算処理はしばしば複数の電子テキストを対象とする。処理する行が、十万から百万のオーダーに達すれば、上記のような細かな事柄が、大問題となり得る。

電算処理に適した書式——原則——

原理はきわめて単純である。テキストの論理構造単位が、一義的に、コンピュータに判断できるようにするだけでよい。たとえば、武勲詩に固有の論理構造単位は、詩行と詩節である。本文中では使われない記号で、これらの単位を仕切れば、コンピュータはテキストを適切に処理してくれる。そこで、基本的には、

1. 各詩行の末尾には改行を入れる
2. 詩節の始まりにはタブを入れる[6]
3. 行番号、詩節番号は入力しない

という三つの規則を守るだけでよい。ワープロやエディタには、パラグラフ番号を表示・印刷する機能があるので、行番号は簡単に得ることができる[7]。改行文字を別の記号に置換し、タブを改行に変換すれば、詩節番号を得ることもできる。詩節番号と行番号が同時に得られないのが不便な場合は、番号を入力した電子テキストを別途用意すればよい。そのための PERL スクリプトは筆者のホームページ上で入手できる。スクリプトはテキストファイルなので、中身を確認すると良いだろう。あまりの短さに驚くはずである。人間が読みやすいテキストから、電算処理しやすいテキストを作るのは、時に難事業となるが、逆は、極端に容易い。番号を手入力するのが、いかにむなしいか理解されよう。

散文テキストの場合には、次のように、上記の規則を読み替えれば良い。

1. オリジナルテキストの行の変わり目で改行を入れる
2. 段落の始まりにはタブを入れる[6]。
3. 行番号、段落番号は入力しない

できあがる電子テキストは、行末が揃わず、見苦しくなるが、索引ソフトを用いた場合、段落番号と行番号で索引を作ることができる。散文の一段落は時に非常に長くなるので、段落番号のみの索引では非常に扱いにくくなるだろう。行がどの単語で折り返しているのかは、重要な情報なのである。

ところで、散文テキストの場合には、テキストの論理構造単位は、行と段落だけではない。章、節、項などの区別が必要となることもある。その場合には、章の区切れ目には$、節の区切れ目には\、項の区切れ目には&などといった具合に、記号を使い分ける。もちろん、これらの記号を本文中で使ってはならない。章名、節名、項名なども入力する必要がある場合には、LaTeX 方式を用いて、\chapter{章名}\section{節名}\subsection{項名} などとし、区切れ目に入れておく。なお、入力の際には、見やすさのために、} の後に改行を入れてもかまわないが、電子テキスト完成のおりには、「}＋改行」を} に置換しておかねばならない。改行は、オリジナルテキストの行の境界を示すためにのみ用いるべきだからである。

最後に、不可視文字（タブ・スペース・改行）について、注意を促しておこ

う。

改行の方は間違って、キーを二度押ししたとしても、画面上でそのことがはっきり確認できるが、タブの場合には、気づかない場合も多い。というのも、タブを一回入力しても、画面上で全く変化が起こらないこともあるからである。

また、タブの後ろや、改行の前に、無意識に入れてしまったスペースが、電算処理の際に思わぬ障害となることもある。：や；と本文の間にスペースを入れたり、入れなかったりといった、首尾一貫性のなさも、また然りである。テキスト電算処理専用のソフトを作る場合、こうしたありがちな不統一が支障を来さないよう、テキストを下準備する仕組みを用意するのが常識である。だが、エディタやワープロの検索置換機能を使って検索を行う場合には、そうした配慮は望めない。

以上を考慮すると、電子テキストを入力・加工する際には、不可視文字が画面上で確認できるテキストエディタを使うべきである（たとえば、MacOS 上では、Jedit, YooEdit, Tex-Edit Plus などが、不可視文字を明示できる）。

電算処理に適した書式——例外処理——

詩行番号とパラグラフ番号のずれ

すでに述べた通り、韻文テキストでは、「各詩行の末尾に改行を入れる」というのが原則である。これにより、原本の行番号（以下、ノンブル）と電子テキストのパラグラフ番号にずれが生じないようにするわけである。ところが、原則に従ったがために、原本のノンブルとパラグラフ番号がずれる、という事態が生じ得る。原本のノンブルに、何らかの理由で、脱落や重複があった場合である[8)]。

上記のような場合、空行を入れたり、複数詩行を一つのパラグラフに入れるなどして、パラグラフ番号を操作する必要がある。

空行の設定

電子テキストに空行を設定しなければならないのは、次の二つの場合である。

1. 写本の行脱落にあわせ、校訂者が意図的にノンブルをずらした場合
2. 誤植などが原因のノンブルミス

写本に行脱落があった場合、第何行目が脱落しているのかは、通常校訂者が明示しているので、該当するパラグラフ番号の行を空行にすれば良い。

一方、ノンブルミス場合、原本のノンブルとパラグラフ番号の矛盾が生じる行の直前を空行にする。たとえば、ノンブル 20 が実際には第 19 詩行を表している場合、第 19 パラグラフを空行にし、第 20 パラグラフにノンブル 20 が付された詩行を入れる。四行おきのノンブルの場合パラグラフ 17 を、五行おきのノンブルの場合、パラグラフ 16 を空行にする方法も理論的にはあり得るが、お進めできない。たとえば、ノンブル 1-4 あるいはノンブル 1-5 で、早くもミスがあった場合、作品第一行目を空行にしなければ一貫性を欠くことになる。ファイル冒頭の空行は処理ミスの原因となりやすい。

詩行の合体

一パラグラフに二詩行をまとめなければならないのは、次の二つの場合である。

1. 校訂の底本にしたがったノンブル処理を行いつつ、他の写本からノンブルなしで詩行を補った
2. 誤植などが原因のノンブルミス

ノンブルのずれが写本の事情に基づく場合には、16, 16 a, 16 b. . . といった具合にすることが多い。こうした場合、16 a, 16 b. . . はノンブル 16 のテキストの後に詰め込む。アルファベットつきのノンブルを使わない校訂本もあるが、その場合でも、どの詩行を補ったかは校訂者が指示しているはずなので、対処法は、同じである。いずれにせよ、こうした処理を行った場合、それぞれの詩行の区切り目を何らかの記号で示しておいた方がよい。筆者の場合には、二本のスラッシュ（//）で表している。

誤植が原因の場合には、空行を補う場合と同じく、原本のノンブルとパラグラフ番号の矛盾が生じる行の直前の行に、余分な行をすべて詰め込む。たとえば、実際には、第 21 詩行なのに、ノンブルが 20 となっている場合には、パラグラフ 19 に二詩行を詰め込み、ノンブル 20 のテキストがパラグラフ 20 に位置するようにする。

詩節番号や段落番号の脱落と重複

すでに述べた通り、適切にタブが用いられている限り、詩節（段落）は、自動的に番号をつけることができる。しかし、何らかの原因で原本に、詩節（段落）番号の脱落がある場合、原本の番号とコンピュータの付した番号にずれが生じる。そうしたずれを解消するには、ずれが生じる詩節（段落）の直前に、タブを一つではなく、二つ入れる。

一方、原本で詩節（段落）番号が重複している場合には、ずれが生じる詩節（段落）のタブの前に専用の記号を付す。記号には何を用いても構わないが、他の用途に用いられていないことが前提である。

上記のようになっている場合、自動処理のためのプログラムを若干書き換えるだけで、正常な処理を見込むことができる。たとえば、タブが連続して並んでいる場合には、詩節（段落）番号を余分にカウントして、後続の詩節（段落）番号が原本のノンブルとずれないようにすることができるし、タブの前に専用の記号が入っている場合には、直前の詩節（段落）と同一の詩節（段落）番号を再設定することもできる。

行番号や詩節番号が正常に並んでいない場合

これは場合により解決法が異なる。たとえば、行番号が5, 15, 10と並んでいて、誤植だということが明らかな場合には、ノンブルとオフセットのずれは無視して構わない。むしろ、原本のノンブルを書き換えておいた方がよい。一方、誤植ではなく、写本の並び順を校訂者が変更したことを、ノンブルが示している場合もある。その時には、写本の並び順を尊重するなら、電子テキストの行を入れ替えるべきであるし、校訂本の並び順を尊重するなら、そのままにしておくべきである。あるいは、二種類の電子テキストを作る。ただ、こうした厄介なテキストの場合、プログラミングに心得のある人と相談するのが一番良い。筆者自身は、そうした相談に、できる限りの対応をする心づもりがある。

ハイフネーション

電子テキスト内でハイフネーションを避けるべきなのは、すでに述べた通りである。もちろん、grand-mère のように常にハイフンが必要な語はその限りで

はない（その場合でも、この語が二行にまたがらないようにする）。ハイフネーションされた単語は、常に、次行に単語全体を移動する。というのも、段落の最終行がハイフネーションされた単語の後半部分だけからなる場合、単語を前の行に移動すると、段落の最終行が空行になってしまうからである。こうした空行は、電算処理の際ミスを引き起こしがちである。たとえば、テキスト末尾の改行マークが、空行になった最終行を示すのか、それとも、単に、テキストの締めくくりを表すのかは、電子テキスト内部では決定できない。

例外処理の後処理

ハイフネーションの解除以外の例外処理をした場合、処理内容や、処理に使用した記号は、テキストファイルに記入しておき、電子テキスト本体とひとまとめにして、フォルダ（ディレクトリ）に入れておく。そうした添付書類は他人に電算処理を依頼する場合に役立つばかりではない。多くの場合、電子テキストを作り上げて、一年もすれば、自分が行った例外処理など思い出せなくなる。忘れたまま、電算処理ソフトを使った場合、思った通りの結果が得られないばかりか、不具合の原因を特定するために、時間を浪費することにもなりかねない。

文字・記号の使用——一般論として——

欧文テキストで用いることのできる文字は、ASCII コードでは、127 番をのぞいた、33 から 255 番までの 222 文字である。33 から 127 番までは LowASCII コードと呼ばれ、Windows, Macintosh 間でも文字化けが生じない。一方、128 から 255 番までは HighASCII コードと呼ばれ、プラットフォームにより割り当てられた文字が異なるため、文字化けがおこる。しかも、どの文字を収録するかが異なっているため、Macintosh では表示できても、Windows では表示できない文字が存在する（逆もある）。したがって、クロス・プラットフォーム（異なった OS 間でのデータの互換）を考慮するなら、HighASCII 文字の使用には十分注意すべきである。両プラットフォームでコードが割り当てられている文字以外は、問題を引き起こすので用いるべきではない。また、記号類を、無用の用途に用いてはならない。テキスト内の事象は多様であり、いたずらに浪費できるほどに多くの文字が与えられているわけではないからである。たとえば、先に述べた、章や節の区切りに利用できるのは、本文内で用いられない記号だけである。残された文字は、こうした特殊な用途のために用いるべきで

ある。

一般論で考えると、本文内で使用される可能性のない文字は、それほど多くない。アルファベット文字（合字、アクサン付きを含む）、句読点、それにデータ互換上問題のある文字をのぞいた、記号文字を検討してみよう。

まず、LowASCII コードでは以下の文字しか残らない（数字は ASCII 十進コード、以下、LowASCII 文字は常に下記の表記法で示す。）。

34 ", 35 #, 36 $, 37 %, 38 &, 39 ', 40 (, 41), 42 *, 43 +, 47 /, 60 <, 61 =, 62 >, 64 @, 91 [, 92 \, 93], 94 ^, 95 _, 96 `, 123 {, 124 ¦, 125 }, 126 ~

上記のうち、通常、文書内で用いられることがないのは、#, *, @, `, {, }, ¦, ^, ~くらいのものだろう。

次に、HighASCII 文字では、通常、以下の文字の用法は決まっている（数字は Macintosh の ASCII 十進コード、() 内の数字は Windows の ASCII 十進コード。以下、常にこの規則に従って HighASCII 文字を示す）。

163(163)£, 177(177)±, 180(215)¥, 199(171)«, 200(187)», 201(133)..., 208(150)–, 209(151)—, 210(147)“, 211(148)”, 212(145)‘, 213(146)’, 214(247)÷, 228(137)‰

また、226(130), 227(132), 246(136), 247(152), 248(175), 171(180), 251(176), 252(184), 254(139) は特殊文字で、ワープロやエディタでは、非常に扱いづらい（文字を示さないのは、印刷過程でのトラブルを防ぐためである）。それゆえ、記号として利用可能なのは、次のものくらいだろう。

160(134)†, 162(162)¢, 164(167)§, 165(149)·, 166(182)¶, 168(174)®, 169(169)©, 170(153)™, 172(168)¨, 181(181)μ, 187(170)^{a}, 194(172)¬, 196(131)*f*, 224(135)‡, 225(183)·

とはいえ、先にも断った通り、これは一般論である。ある程度、記号の自由度を増すために、言語や分野ごとに、どの記号が自由に使えるかを決めた方が

現実的だろう。だが、その場合でも、まず、#, *, @, `, ¦, ¦, ¦, ^, †, ¢, §, ·, ¶, ®, ©, ™, ¨, μ, ª, ¬, f, ‡, ·といった文字を利用し、それでまかなえない場合に、当該言語・分野で不用と思われる記号を利用すべきである[9]。

文字・記号の使用——フランス語、中世フランス語に関して——

前項で述べた通り、記号の用法は、各分野で検討を加える必要がある。以下では、筆者の専門である、中世フランス語に関して、記号の用法を吟味する。とはいえ、中世フランス語のテキストは、多かれ少なかれ、現代語風に校訂される。まず、現代フランス語にもあてはまる事項から検討しよう。

まず、多義的に用いられがちな記号としては、39 ' がある。39 ' は、時に引用や単語の強調に用いられるが、これはエリジョンなどの文字の省略を表す場合にのみ用いるべきである。引用や強調には、212(145)‘, 213(146)’ を用いるべきである。これらの記号は、左右の区別があるため、引用箇所や、強調箇所のみを抜き出すことが簡単にできる。同様の理由で、二重引用符には 210(147)“, 211(148)” を用いる。ところで、213(146)’ と 39 ' の混用の例は非常に多い。ワープロやテキストエディタで、39 ' の入力を自動的に 213(146)’ に置き換える機能を持つものがあるためである（Macintosh 用の Tex-Edit Plus のスマート引用符など)。両者は異なった文字なので、混用があると、電算処理の結果が不正確になる。細心の注意を払うべきである。これは、211(148)” と 34 ″にもあてはまる[10]。

終止符（46 .）も、多義化することがある。省略を表すのに、201(133)...ではなく、...と終止符を三回入力してしまう人は案外多い。また、中世フランス語のテキストに関して言えば、写本では、ローマ数字の前後と位取りを黒点で示していることが多い。筆者がかつて作成したものも含め、多くの電子テキストでは、その黒点を表現するために、終止符が用いられている。しかし、終止符は、コンピュータに対し、「文」の概念を定義するための、大切な指標である。冠詞の·I·さえも、そのまま温存する今日、ローマ数字にまつわる終止符は、電算処理の一大障害となりえる。225(183)·を用い、·M·C·などとするべきだろう。

次に、一つの事象が複数の記号で表される場合としては、登場人物のセリフ

がある。時に、ギュメ（99(171)«, 200(187)»）、時に、ティレ（(151)－）、ある時には、“と”、ある時には、‘と’、さらには、’ や ” が使用される場合もあるという、恐ろしいほどの多様ぶりである。最後の文字記号の混同を別にすれば、微妙なニュアンスの差を込めて、各記号が使用される場合があるのは否定しない。特に、近現代の作家の場合、引用符の用法は、重要な意味を持ち得る。だが、中世フランス語のテキストの場合、引用符は、通常、校訂者が付すもので、オリジナルテキストには属さない。したがって、セリフを常にギュメではさみ込み、電算処理の都合を優先させたほうがよい。dit il などの挿入句は地の文なので、. . . » dit il, « . . .とするとよい。引用符の多様性を維持するためには、«－とすれば、印刷する時には、«－を－に置換すればティレになる上、電算処理の都合にもかなう。後にティレや他の引用符は続かないものの、印刷時に不用となる、ギュメは、«@や@»といった具合に、何らかの記号で、そのことを表すようにすればよい。ただし、前節最終項の「例外処理」の場合と同様、どういう処理をしたのかを添付書類内に書き留めておくべきである。なお、セリフの中のセリフは、先にあげた二重引用符と一重引用符を使えばよい。これらの記号は、地の文で用いられていれば、引用か強調であり、セリフの中（つまり、ギュメの間）で用いられていれば、セリフ内のセリフということになる。電算処理で両者を区別するのは容易い。なお、『千夜一夜物語』のような、重層構造をもつテキストでは、セリフの階層が深くなるたびに、ギュメを«, ««, «««, «««« と増す方式をとるか、«1, «2, «3, «4 のようにして、他の引用符を使わない方が、結果的に電算処理でも、入力でも楽ができる。なお、階層を表す数字とギュメの間にはスペースは入れないようにしなければならない（セリフ本体の数字と区別がつかなくなる可能性がある）。

中世フランス語のテキストに特有の事象に関しては、次のようなものがある。

まず、nel などのアンクリーズを明示する校定本があるが、この場合、ne·l というふうに、165(149)·を使うと良い。これは、印刷本の習慣とも一致する処理である。ただし、ローマ数字の 225(183)·と混同しないよう十分注意されたい。225(183) の方は、ローマ数字の黒点よりも、若干大きく、選んだフォントによっては（MacOS 9.2 以前の Courier など）、かなり大きな黒丸となる。なお、古い校定本に見られるように、アンクリーズも ne'l とアポストロフで

表せば良い、という意見があるかも知れない。しかし、エリジョンが一貫して明示されるのに対し、アンクリーズはその総てが明示されるわけではない。たとえば、前置詞と冠詞の縮約まで de·l とすることはない。したがって、エリジョンとアンクリーズを同じ記号で表すのは避けた方が良い。

次に、Paratax 構文を表す記号も提案しておこう。筆者は、Edition électronique de la Chanson de Roland の暫定版で、√を使用したが、考えが足りなかったようである[11]。というのも、この文字は Windows では ASCII 文字に含まれないし、HTML 形式での記述法もないからである。現在、HTML 形式では、やむなく、§を使用しているが、これも問題がある。§は慣習上明確な意味をもっており、文中に混在したこの記号は非常に場違いな印象を与えるからである。電子テキストでは、見た目は重要ではないというのは事実であるが、これまで、広く認められた Paratax 構文を表す記号はなかった以上、記号そのものがころころと変わるのはまずい。今後は一貫して 194(172)¬を使うことにしたい。HTML 形式では、この文字は、&#172；(十進法表記) ——&#x00AC；(十六進法表記)、&not；(実体参照表記) ——で、LaTeX では、$\neg$もしくは、$\lnot$ で表される。

なお、中世フランス語テキストの記号使用法については、*Edition électronique de la Chanson de Roland* の第一版で、体系化する。各種記号の HTML, LaTeX 表記もあわせて記述する予定である[12]。

マーキングについて

コンピュータは神にも比すべき計算力と記憶力を備えた三歳児である。我々にとっては自明の、詩行（行）、詩節（段落）といった概念でさえ、手取り足取り教えてやらねばならない。以下では、コンピュータに情報を提供する適切な手段を考察してみよう。

中世フランス語で書かれた作品の校定本では、通常、アクサン・グラーブは用いられない。しかし、前置詞の a や en にアクサンをつければ、コンピュータは、それらの語を、avoir の a や中性代名詞と区別できるようになる。一方、縮約している前置詞にアクサンをつけるのは、習慣に完全に反している。だが、àl, às, èl, ès などとしておけば、前置詞の a, en の用例をかなり網羅的に取り出すことができる。さらに、固有名詞を、冒頭の文字だけでなく、語の全体にわたって大文字にしておけば、テキストから固有名詞だけを抽出すること

もできる。

コンピュータは、すべてを外見で判断する。前置詞の a と avoir の a を見分ける術は持たない。そこで、両者の外見を変える必要が生じてくる。これがマーキングである。上に挙げたのは、ごく初歩的で、手間のかからないものばかりであるが、時には、相当大掛かりな作業を強いられることもある。たとえば、テキスト内の動詞の活用形を網羅的に抽出する場合などが、それである。こうした時には、単に、動詞の活用形の前に©といった記号をつけただけでは、満足なデータを得ることができない。動詞の活用形は文脈から切り離すと、法、時称、人称、時には不定法形態を判断することさえも困難になるからである。一方、これらの情報をコンピュータに教えてやれば、かなり質の高いデータベースを構築することができる。その方法の一つとして、テキストに次のような書き込みをするやり方がある[13]。電子テキストは簡単にコピーできるので、書き込みをしたところで問題はない。

A icel jor ke la dolor fu {estre} {indpres} {3 p} grans
Et la bataille orible en Aliscans,
Li quens Guillaumes i soufri {sofrir} {passsimp} {3 p} grans ahans.
(以下省略)

ところで、上の例に従う場合、気をつけないといけないのは、amer {inf} などとしてはならないということである。この場合、amer {amer} {inf} || が正しい記述となる。一番目の || には不定法形態を、二番目の || には法・時称を、三番目の || には人称を入れる、というのが、当面の規則だからである。

テキスト内の単語に続く、|| はデータベースのフィールドに相当にする。データベースについては、別の機会に述べたので、詳細はそちらに譲り、ここでは要点だけを繰り返しておこう。まず、同一フィールドには常に同質のものが入らなければならない。すなわち、|| をその順により、第一フィールド、第二フィールド、第三フィールドとするなら、第一フィールドに不定法形態、第二フィールドに法・時称名、第三フィールドに人称番号を入れるとし、単語ごとにフィールドの数を増減させたり、フィールドの順番を換えたりしてはならない。つぎに、空白のフィールドもそれ自体情報価値をもつので、削ってならない。

上記に加え、次の点にも注意しなければならない。まず、フィールド内に同綴異義語・異綴同義語が生じないようにしなければならない。たとえば、nier と noyer のいずれの意味の場合でも区別せずに、不定法形態の noiier を第一フィールドに入れると、電算処理に問題が生じるのは明らかである。noiier 1, noiier 2 などというふうに区別しなければならない。逆に、同じ動詞の活用形に関して、異なった不定法形態を第一フィールドに入れてもいけない。同様に、第二フィールドで、同一の法・時称を表すのに、異なった表記を用いてはならない。第三フィールドで 3 P を三人称単数と三人称複数の意味で用いてしまうというのもありがちな失敗である。

以上のことから理解される通り、実際に電子テキストに書き込みを始める前に、十分な準備を行わねばならない。第二、第三フィールドの場合、あり得る文法事項を網羅的な一覧表として作りあげておけば良い。たとえば、00inf, 01indpres, 02subjpres, 03imper, 04indimpf, 05partpres, 06futur, 07cond, 08passsimp, 09subjimpf, 10partpass ; 1 P, 2 P, 3 P, 4 P, 5 P, 6 P といった具合である。後は入力ミスを防ぐだけである。

マーキングの処理

本稿の趣旨からは若干逸脱するが、最後に、簡単なマーキング処理法を紹介しておこう。プログラム処理をすれば、動詞活用表を HTML 形式や LaTeX 形式で書き出したり、電子辞書に加工したりすることができるが、ここでは、索引作成ソフト（筆者のホームページで入手可能）と表計算ソフトを使った処理法に話を留める。ソフトの使い方は説明しないので、各ソフトのマニュアルを参照されたい。

電子テキスト内のすべての動詞活用形態にマーキングを終えたら、次に索引作成ソフトにかける。すると、

aime {amer} {indpres} {3 P} （以下、タブと参照番号）
aime {amer} {subjpres} {3 P} （以下、タブと参照番号）
（以下、省略）

といった具合に見出し語がたてられることになる。マーキングにより、コンピュータが、同綴語を文法機能で区別できるようになったわけである。

ところで、上記の手順で作った索引には不要な見出し語が多数含まれている。マーキングをしなかった名詞や形容詞も見出し語になるからである。これを解決するには次の手順をとる。まず、索引の |をタブに、| を空の文字列に一括置換し、次に、索引をタブ区切りデータとして表計算ソフトに読み込ませる。すると、表計算ソフトは、活用形態を A 列に、不定法形態を B 列に、人称を C 列、参照行番号を D 列に読み込んでくれる。だが、そうなるのは、動詞の活用形態だけである。他の見出し語の場合は、A 列に見出し語、B 列に参照行番号がはいる。したがって、C 列でソート（並べ替え）を行えば、簡単に動詞活用形態の見出し語だけを取りだせる。動詞以外の語は C 列にデータがないので、昇順で並べ換えれば表の上方に、降順なら下方に全部集まるので、それらを一括選択して削除すればよい。

上記のようにして表計算ソフトに読み込んだデータは様々な操作が可能である。B 列でソートすれば、不定法形態ごとに活用形態がまとまるし、C 列でソートすれば、電子テキスト内の全ての直説法現在形態を取り出すこともできる（なお、前項で示した通し番号の人称番号、00-10 という番号を伴った法・人称名、記述法は、表計算ソフトでの並べ替えを念頭に置いている）。

同様の方法で、電子テキストの全語彙集を作ることもできる。ただし、フィールド数が増えることになる。たとえば、次のようなフィールド定義が必要になる。

第一フィールド：代表形態（不定法形態、原形など）
第二フィールド：品詞
第三フィールド：法・時称
第四フィールド：人称
第五フィールド：性・格・数（名詞、代名詞、形容詞の場合に使用）

amor が par amor という表現で使われていて、par～などと記述したい場合や、語義を書き込みたい場合には、さらにフィールドが増える。そうなると、フィールドの位置を間違わないようにするのが重要になってくる。|1| |2| |3| |4| |5| |6|...といった具合に、フィールド番号を挟んだ || を、コピー機能や仮名漢字変換ソフトを利用して、簡単に入力できるようにした方が良いだろう。あるいは |代表形態| |品詞| |法時称| |人称| |性格数| |語義| |熟語|

としても良い。要はフィールドにラベルをはっておくのである。もちろん、入力がすべて終了した時点で、一括置換などを使って、これらのラベルを剝がすのを忘れてはならない。

注

1）電子テキストの書式については、「文学と電子テキスト－電子テキストの可能性、技術的問題について」（『LUTECE』、30号、2002年）で、若干言及した。しかし、その後、経験を積むにつれ、当時の考えを変更せざるを得なくなった。上記論文と現在の筆者の立場が異なる場合は、必ず、注記を加える。なお、筆者が開発するプログラムは、上記論文の記述に従った電子テキストも処理できる。

2）本稿では、正規表現と言えば、常に、PERLの正規表現を表す（PERLはUNIX, Windows, Macintoshで利用可能なスクリプト言語である）。正規表現は様々なソフトで利用できるが、ソフトにより方言がある。たとえば、Macintoshでは、JeditやYooEditが正規表現に対応しているものの、いずれも本稿の正規表現を正しく解釈できない。また、本稿では、改行を\nで表す。パラグラフ末尾を指示する「改行」は、システムごとに異なっており、UNIXでは\n、Macintoshでは\r、Windowsでは\r\nとなる。ただし、MacOS 9.2以下のシステムで動作するMacPerlでは\nで改行を表せる。MacOSXでshellを介した場合には、\rで表現しなければならない（\は日本語環境では¥になる）。

3）実際には、PERLの正規表現は非常に柔軟性に富んでおり、haubertを検索したいのなら、¥bhau¥W*berts?¥bで事足りる。コンピュータは「300語で要約せよ」という意味での単語（mots）は比較的簡単に認識できる。単語境界を定義するだけでよいからである。しかし、「フランス語の単語を300個知っている」という時の意味での単語（vocable）を認識するには、膨大な定義が必要となる。本文内での冗長な正規表現は、これをわかりやすくするためのものと理解されたい。

4）現在では、LaTeXで組版された書籍が多数書店に並んでいる。LaTeXによる組版の品質については、福岡大学人文学部ドイツ語学科、永田良久氏のホームページ（http : //www.lg.fukuoka-u.ac.jp/~ynagata/）「他言語処理用TeXのインストール法」を参照されたい。PDF形式での見本も多数収録されている。

5）テキストは*Aliscans,* éd. par MM. F. Guessard et A. De Montaiglon, Librarie A. Franck, Paris, 1870を利用したが、論述の都合上、行番号は改変している。

6）本節の記述は、「文学と電子テキスト」41頁の記述とは異なる。後者では、詩節の区切れ目を何らかの記号で表す、としたが、タブを使うのが、常識に照らしても適切だろう。

7）パラグラフという概念に注意されたい。これは、文書冒頭と改行文字ではさまれた文字列、改行文字と改行文字にはさまれた文字列、改行文字と文書末に挟まれた文字列、のいずれかと定義できる。簡単にいえば、リターンキーを押せば、新

しい段落が開始する。したがって、韻文電子テキストの一詩行は一パラグラフに相当する。ワープロやエディタでは、行がある程度の長さになると、画面上で自動的に行が折り返す（ワードラップ）が、この場合の行末には改行文字は入らない。それゆえ、リターンキーを押して、パラグラフを改めることは、ワードラップとは本質的に異なる。本稿では、パラグラフを上記の意味でのみ用いる。後に出てくる散文テキストの段落と混同しないよう注意されたい。

8）電子校訂本でも、既存の校訂本のノンブルと歩調を合わせたいということが起こりえる。しかし、この場合にも、校訂本のノンブルを参照するわけなので、以下では、オリジナルが既存の校訂本だと仮定して話を進める。

9）ある種の分野では、@, ®, ©, ™ も用途が決まっているが、文学作品の電子テキストを扱う限り、これら比較的新しい記号類の使用が問題になる可能性はきわめて低い。一方、‖ は、テキスト内で集合数式が引用された場合を考えれば、多少、問題がある。しかし、それは極めてまれな場合であって、別途対処法を考えるべきであろう。むしろ、後から見る通り、‖ のように左右で対になる記号が、一組確保されていることの方が、電算処理の観点からは、ずっとメリットが大きい。

10）この記述は、「文学と電子テキスト」45 頁の記述、「シングル・クォーテーションやダブル・クォーテーションの使用も勧められない」とは完全に相反する。たしかに、これらの記号は、様々な混乱を引き起こし得る。だが、左右の別のある記号は非常に貴重なので、使用を諦めるのは得策ではない。もちろん、使用には細心の注意が必要である。たとえば、「文学と電子テキスト」45 頁 9 行目の「アポストロフや" を引用符として」は誤植である。「アポストロフや〃 を引用符として」が正しい。原稿では間違っていないのだが、編集段階で文字が入れ替わったらしい。本稿で文字の直前に ASCII 番号を示したのは、そうした事情による。

11）Edition électronique de la Chanson de Roland については、筆者のホームページ、Hruodlandus-et-Alda（http : //www.eonet.ne.jp/~ogurisu/）を参照されたい。

12）文字コードの問題を考える場合、Unicode が重要なトピックになることは、筆者も承知している。にも関わらず、論考を ASCII コードに関してのみ進めたのには、端的に、MacOSX（10.3）上では、Unicode が、まだ十分に安定していないからである。たとえば、Jedit（Rev 4.2.3 r 0）は、Unicode 変換が全く機能せず、OS のクリップボードからのペーストで文字化けがおこるばかりか、自分で Unicode 保存した書類さえまともに開くことができない。一方、TexEdit-Plus（4.52）は、HighASCII 文字の Unicode 変換に全角文字が混入する。MacOSX に付属する TextEdit は Unicode 保存がそもそもできない。

なお、MacOS と WindowsXP 間でのテキスト交換に、プレイン・テキスト・ファイルは、もはや使えない。WindowsXP 上では、多くの HighASCII コードが、文字を呼び出すことができないからである。現在、両者でデータを交換する最も確実な方法は、RTF フォーマットを使うことである。MacOS 上で作成した RTF

ファイルは、WindowsXP 上では、MS-WORD ならば、そのままで、文字コードを変換すれば、ノートパッドでも開くことができる。

ただし、RTF 形式であっても、本文中で述べた、Windows, MacOS のいずれかに対応する文字のないコードは、当然ながら、互換性がない。また、RTF の文字コードは、ASCII 16 進コード+1 に\' をつけて表現される。本稿が ASCII コードだけに話をしぼったゆえんである。

13) 本節の記述は、「文学と電子テキスト」51 頁の記述とは異なる。そこでは、マーキングの全体を ‖ ではさみ、各フィールドを¦で区切る方法を提案した。だが、その方式と本稿の方式に、本質的な違いは何もない。本節および次節は、むしろ、「文学と電子テキスト」の記述を拡充したものと理解されたい。