タッチタイプ入力方式による日本語エディタ

池田 勇二　中西 隆　郡司 隆男　大岩 元
（豊橋技術科学大学）

１．はじめに

ＯＡ化の波に乗って、わずか数年の間に日本語ワードプロセッサが広く使われるようになってきた。しかし、計算機による日本語処理はまだ研究の歴史も浅く、解決すべき問題点も多い。

まず問題となるのは日本語の入力方法であり、これまでに数種類の方式が提案され【１】、研究されている。最近では、ＪＩＳかな鍵盤あるいはＡＳＣＩＩ鍵盤からのローマ字入力による、かな・漢字変換方式が日本語入力の一般的な方法になっているが、この方式は、システムを使い慣れてくると疲労も多く、入力速度も決して速くはない。また近年、ワープロ専任タイピストの精神障害や視力低下などの問題が新聞などで取り上げられている。

一方、我々の研究室では、高速入力が可能で、疲労も比較的少ないタッチ・タイプ入力方式によるマルチ・ストローク・コードのＴＵＴコード【２】を開発し、これまでにＪ－ＴＹＰＥ【３】，Ｊ－ＥＤＩＴ【４】といった簡単な日本語編集プログラムを発表してきた。また、２ストローク・コードの１つＴコードを基礎とするエディタＫＥＤ【５】も発表されている。今後はさらに、タッチ・タイプ入力方式による日本語編集の操作性などの研究が必要である。

さらに、最近ではＬＡＮ（ローカル・エリア・ネットワーク）と共にワークステーションが大きな注目を集めており、将来、日本語による計算機との対話が期待される。計算機との対話において、エディタは、文字列（プログラムや文書）を編集レイアウトするための最も重要で基本的なツールである。現在見られる日本語エディタは、欧米で開発されたエディタ・システムに対して取り扱える文字セットを拡張しただけのものが多いが、本来日本語と欧米の言語はその特徴が大きく異っているので、日本語特有の言語構造を反映したエディタの開発が必要である。

以上の事から、以下の設計目標をたて、今後の日本語処理の研究の基礎となる日本語エディタＰＭＡＣＳを作成した。

（１）　タッチ・タイプ入力（ＴＵＴコード入力）、および日本語編集に適した操作性を持つ。
（２）　テキストとして日本文・欧文の両方が扱える。
（３）　エディタの機能が拡張・変更できる。

本エディタの作成に当っては、タッチ・タイプに関する豊富な経験を持つアメリカで、現在代表的なエディタであるＥＭＡＣＳ【６】を参考とし、プログラム開発用言語にはＵＣＳＤ Ｐａｓｃａｌ　ｖｅｒ４．１を用いた。ＥＭＡＣＳを参考として作られた日本語スクリーン・エディタとしては、武蔵野通研のＪＭＡＣＳがあるが、これは「日本語のテキストも英語と全く同じコマンドで編集できるＥＭＡＣＳ風のスクリーン・エディタ」【７】ということであり、我々の開発目的とは必ずしも一致しない。

２．ＰＭＡＣＳの特徴

ＰＭＡＣＳは、ＥＭＡＣＳを参考として設計を行い、エディターとしての基本的な部分はＥＭＡＣＳの優れた所を取り入れている。反面、日本語と欧米語では、明らかにその特徴が異っているので、データ構造やコマンドは、日本語エディタであるということに重点を置いて設計を行った。

この章では、ＰＭＡＣＳの主な特徴を示し、それぞれについて説明することによってＰＭＡＣＳの概要を与える。

ＰＭＡＣＳの特徴：

（１）　タッチ・タイプ入力方式のＴＵＴコードによる日本語入力
（２）　マルチ・バッファー機能
（３）　マルチ・ディスプレー機能
（４）　コマンドの拡張・変更機能
（５）　ミス・ストロークの修正機能
（６）　日本語の特徴を反映した各種コマンド

２．１　タッチ・タイプ入力方式のＴＵＴコードによる日本語入力

日本語の入力方法にタッチ・タイプ方式を採用することにより、エディタ自身の性格も他の入力方式を採用しているエディタのそれと大きく異ってくる。そのことは、以下のＴＵＴコードの特徴を見ることによっても容易に理解できるだろう。

ＴＵＴコードの特徴

（１）　２ないし３ストロークで、ひらがな・漢字合わせて約２５００文字を直接入力できる。
（２）　コードの割り当ての最適化を行ってあり、高速入力が可能である。
（３）　ひらがなについては、構造的にコードが割り当てられており覚えやすい。

ＴＵＴコードについて、さらに詳しいことは別文献【２】に述べられている。

２．２　マルチ・バッファー機能

マルチ・バッファー機能とは、編集中に複数のバッファーを同時に持ち、それぞれのバッファーには異ったテキストを保持できる機能をいう。この機能は、先に上げた特徴の（３）と組合せて使うことにより、編集効率を大きく向上させることができる（次節２．３参照）。

ＰＭＡＣＳでは、マルチ・バッファーを実現するために図１に示すような環境を持っている。

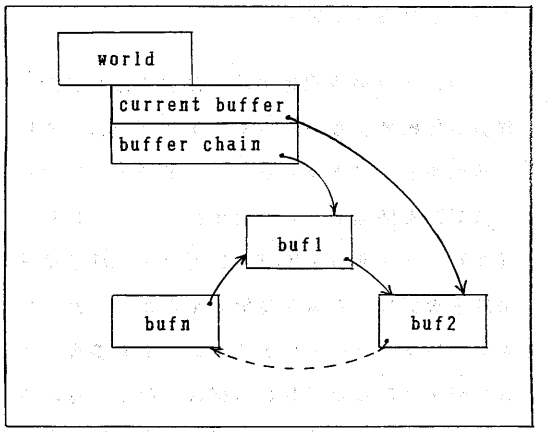


図１．　マルチ・バッファーの環境

ｗｏｒｌｄは、バッファー全体の環境を示す。複数のバッファーは図の様にリング状に接続されている。ｃｕｒｒｅｎｔ＿ｂｕｆｆｅｒは現在編集を行っているバッファーを指すポインタで、ｂｕｆｆｅｒ＿ｃｈａｉｎのポインタを入

れ換えることにより、編集の対象となるバッファーを変更する。バッファーの数ｎは、空いているメモリーの大きさによって制限される。

２．３　マルチ・ディスプレー機能

マルチ・ディスプレー機能は、前節のマルチ・バッファー機能による複数のバッファーの内容を同時に画面に表示するための機能である。

この機能を実現するために、各々のバッファーには図２に示す様な表示サイズが与えられ、バッファーの内容は表示サイズで指定された画面の中に表示される。

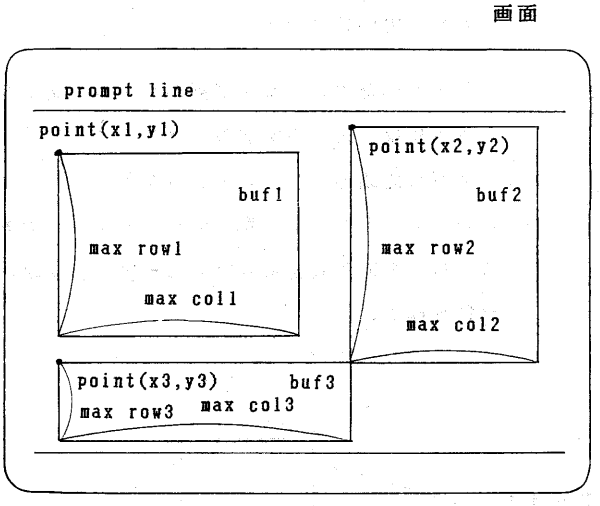


図２．　おのおののバッファーの表示サイズ

表示サイズは、ｐｏｉｎｔ、ｍａｘ＿ｒｏｗ、ｍａｘ＿ｃｏｌの３つのパラメータを持ち、ｐｏｉｎｔは表示画面の左上角の座標（ディスプレーの絶対座標）、ｍａｘ＿ｒｏｗは表示行数、ｍａｘ＿ｃｏｌは表示桁数を与える。テキストの一行が表示桁数を越える場合には、表示桁数に合わせてラッピング（ｗｒａｐｐｉｎｇ）が起こる。表示サイズは随時変更可能である。

このマルチ・ディスプレー機能により、エディタの使い勝手はたいへん良くなる。例えば、複数の文書を相互に参照しながら、別の文書を書くとか、複数の文書の内容を引用（ｃｏｐｙ）して別の文書を作成するといったことが、全て一つの画面の中で行えるのである。

２．４　コマンドの拡張・変更機能

ＥＭＡＣＳの最大の特徴の一つは、エディタの拡張可能性である。ＥＭＡＣＳはＬｉｓｐで記述されており、非常に高い拡張性を容易に実現している。それに対して、ＰＭＡＣＳは記述言語がＰａｓｃａｌであることから、この機能の実現に当って大きな制約を受けることになった。

Ｐａｓｃａｌでは、実行時にプログラムを変更したり、プログラムの動的リンクができないので、それらをプログラム内で擬似的に行わなければならない。そのために、ＰＭＡＣＳでは２つのテーブルを用意し、さらにマクロ機能を持った簡単なインタプリタを作った。

テーブルの一つはキーとコマンドを結合するテーブルで、もう一つは基本ファンクションを実行するためのテーブルである。コマンドは基本ファンクションとコマンドのマクロによって記述される。

前者のテーブルは実行時に変更することができ、キーの定義や、コマンドの定義を変更したり、新たなコマンドを作成したりできる。従ってエディタのカスタマイズも容易に行える。詳しいことは５章で述べる。

２．５　ミス・ストロークの修正機能

この機能は、タッチ・タイプ方式によるマルチ・ストローク・コード（ＴＵＴコード）ならではのものである。マルチ・ストローク・コードは、数個のキー（キーボード上の）の組合せに対して一つの日本語の文字を割り当てている。例えば、『一つ打ち間違えると大違い。』と入力する場合には、'ｋｆｄｕｄ．ｄｉｋｊｚ／ｒｈｇｕｄｊｉｄｚ／ｒｉｆ　'と連続してキーをたたけばよい。しかし、ここで一つのキーが抜けてしまったり、余計に打ってしまった場合には、それ以後のコードが全て誤ったものになってしまう。

一つ打ち間違えると大違い。
kf du d. di kj z/ rh gu dj id z/ ri f␣
　　↑――― d が抜けると

一本支校友豊円本車街豊中
kf ud .d ik jz /r hg ud ji dz /r if ␣
　　└――― 有効情報 ―――┘

図3．　ＴＵＴコードの入力誤り

タッチ・タイプ方式では、かな・漢字変換などのようにひとつひとつ入力文字を確認せず、入力文書に目を置いたままで入力を行うことから、上に述べたような誤りが時々起こる。入力テキストにこの誤りがあった場合は、その部分はもう一度打ち直さなければならないことになるが、ここで図３の誤ったテキストを見てみると、そのテキストには復元可能な有効情報があることがわかる。この有効情報を復元し、誤ったテキストを可能な限り正しく修正することをミス・ストロークの修正機能という。

この機能のアルゴリズムを図４に示す。この図は、横須賀通研で開発されたＨＣＰチャート【８】で書かれている。

復元されたＴＵＴコード列から正しい文字列を得るには、正しい文字列になるまでＴＵＴコード列の先頭の１ストローク分を削ってゆけばよい。ＴＵＴコードは最大が４ストロークであるから、３回先頭を削る間に必ず正しい文字列が得られる。

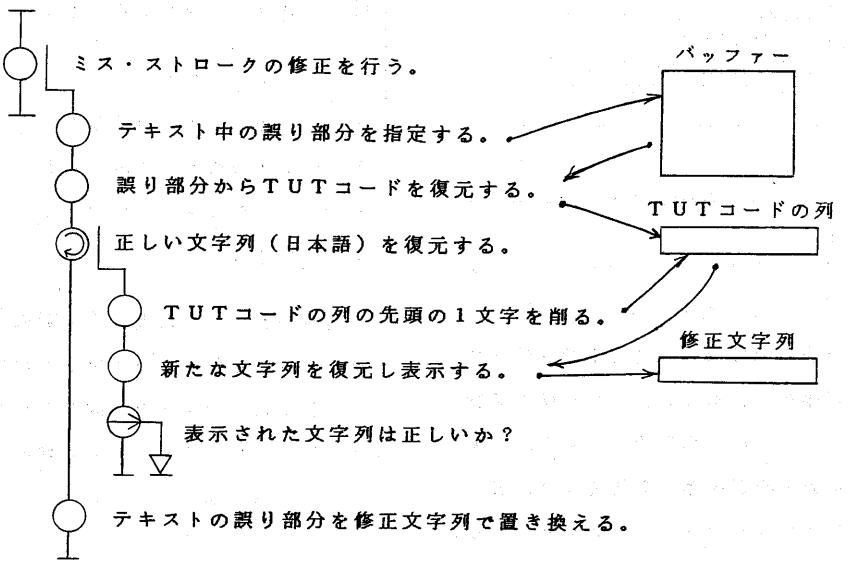


図４．　ミス・ストロークの修正アルゴリズム

テキストの誤りは、ＴＵＴコードにないキーの組合せが入力された時点で自動的に回復する。例えば、ＴＵＴコードではブランクで始まるコードは無いので「。」（ｆ　）や「、」（ｄ　）が入力された時点で誤りから抜け出す。したがって、誤りの文字列の長さはそれほど大きくはならない。

２．６　日本語の特徴を反映した各種コマンド

日本語の特徴を反映するということはすなわち、日本語の文章をいくつかの論理的なかたまりに分けてとらえるということである。例えば、欧米でよく知られている汎用の英文エディタ・フォーマッタのＷａｒｄＳｔａｒでは、英文の中のワード（単語）という論理単位でいくつかのコマンドを設けている。

しかし、英文での論理単位をそのまま日本語に適用することは困難であり、そこから、欧米のエディタの文字セットを拡張しただけの日本語エディタの問題点が生じる。

日本語の文章をいくつかの論理単位に分けてみると、章、節、段落、文、句、…などがあげられる。本エディタでは、段落以下の小さな論理単位について着目し、例えば、段落間、文間のカーソル移動、漢字（熟語）間のカーソル移動、段落、文単位の削除、…などのコマンドを設けた。しかも、これらのコマンドは２．４節で述べたコマンドの拡張・変更機能を利用して簡潔に記述されており、新たなコマンドの作成も容易に行うことができる。

日本語の特徴を反映したエディタを作成するには、そのデータ構造についても検討しなければならないが、この議論は３章で行う。

３．データ構造

ＰＭＡＣＳでは、データ構造に日本語の特徴を反映させている。これは、将来フォーマッタと有機的に結合させるためであり、同時に日本語の特徴を反映したコマンドを作成するためでもある。

３．１　テキスト・タイプ

日本語の特徴として、その論理単位を考えた場合、章、段落、文、句、…などの可能性がある。ＰＭＡＣＳではその中で段落に着目し、テキスト・タイプという新しい型を定義した。テキスト・タイプにおいては、段落が双方向にリンクされており、さらに段落は、適当な大きさの配列を双方向にリンクした形で構成されている（図5b）。ｐａｓｃａｌの定義文を図5aに示す。

```
type  ref_sub_par= ^sub_par;
      sub_par=record
              itself : packed array[1..max_sub_par]
                         of ku_ten_code;
              leng     : integer;
              pre_sub  : ref_sub_par;
              next_sub : ref_sub_par;
            end;
      ref_par    = ^paragraph;
      paragraph  = record
                     itself       : ref_sub_par;
                     tail_sub_par : ref_sub_par;
                     pre_par      : ref_par;
                     next_par     : ref_par;
                   end;
      text_type  = ref_par
```

図5a　テキスト・タイプの定義

多少複雑になっているが、これはメモリーを有効に使おうとした結果である。さらに、挿入と削除のアルゴリズムを工夫することによって、メモリーの効率化をはかっている。

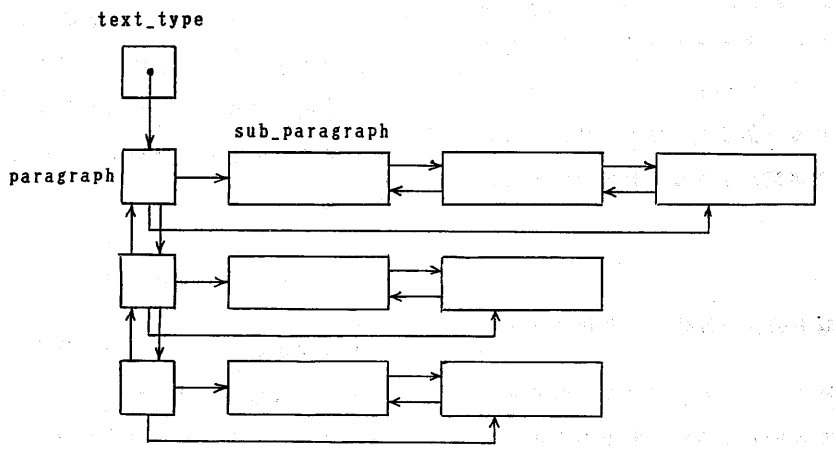


図5b　text_type の構造

また、構造の複雑さは、ともすればプログラムの複雑さにつながりかねないが、テキスト・タイプのハンドリング用ツールによってこの問題を回避している。

バッファーの実体や、エディタ内の引数、コマンドの定義部、削除リストの実体などがテキスト・タイプを構成する。エディタ内の引数とは、基本ファンクションに対する引数で、ａｒｇｕｍｅｎｔとｓｕｂ＿ａｒｇｕｍｅｎｔの２つが用意されている。コマンド定義部は、キーとコマンドを結合するテーブルの一部であり、これについては、５章で説明する。削除リストは、バッファーからブロック単位で削除された部分の過去１０回分までのリストで、これを用いてテキストのコピーが実現されている（ブロックとはポイントによって指された２点間のことをいう）。

３．２　内部表現

日本語エディタでは、テキストの内部表現も重要なポイントの一つである。ＯＳによっては、１バイト・コード（ＡＳＣＩＩコード）と２バイト・コード（ＪＩＳ漢字コード、あるいはＳＪＣコード【９】）を混在させて使っているが、ＰＭＡＣＳでは全てのテキストをメモリー上において編集するという方針から、テキストは全てＪＩＳの２バイト・コードで表現している。

これによって、１バイト・コードと２バイト・コードが混在している場合に起こる幾つかの弊害から逃がれることができ、テキストの扱いが容易になっている。

4．基本ループ

ＰＭＡＣＳの基本ループを図６に示す。基本ループは、コマンド実行しその結果を表示するという単純なものであるが、ここで再表示の最適化を行っている。すなわち、コマンド実行をコマンドが入力されている間は繰り返すことによって、ユーザは編集の途中経過を不必要に見せられることがなく、結果だけを見ることができる。

このことは、タッチ・タイプ入力方式を採用している本エディタにとって好ましいことである。というのは、タッチ・タイプ入力では入力文字を画面でいちいち確認する必要がないので、途中経過を表示することはかえって使用者にとって不用なチラツキとなるからである。また、ＣＰＵもコマンドの実行だけに専念できることから結果として処理も早くなる。

5．　コマンドの拡張・変更機能の実現

コマンドの拡張・変更機能が、２つのテーブルと簡単なインタプリタによって実現されていることは２．４節で述べた通りである。この章では、このことについてさらに詳しい説明をする。

5．1　コマンド結合テーブル

コマンド結合テーブルは、キーボード上のキー（キーの組合せも含む）とコマンドを結びつけるためのテーブルで、図７の様に定義されている。

```
ref_k_tbl  = ^k_table;
k_table    = record   { Key Rebind Table }
               key_name : name_type;
                          { string[20] }
               com_name : name_type;
               com_list : text_type;
               next     : ref_k_tbl;
             end;
```

図7．　コマンド結合テーブルの定義

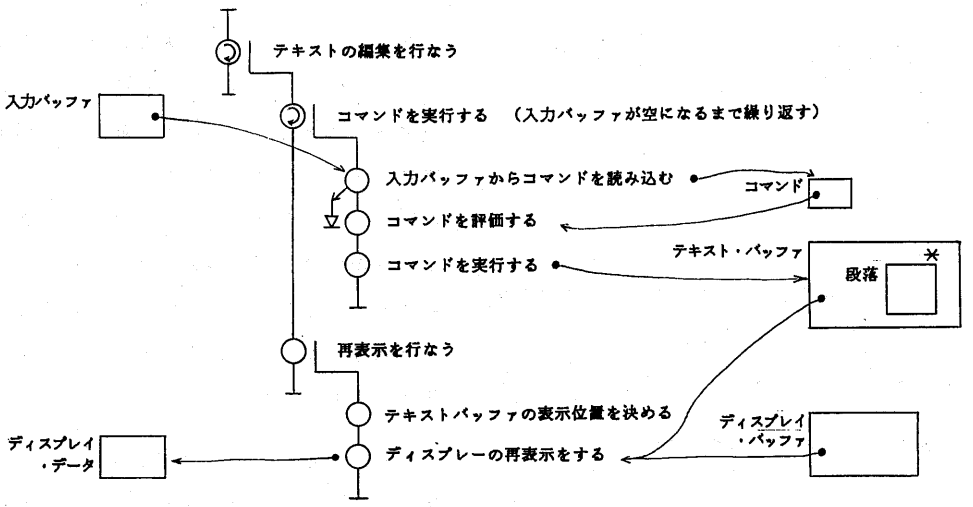
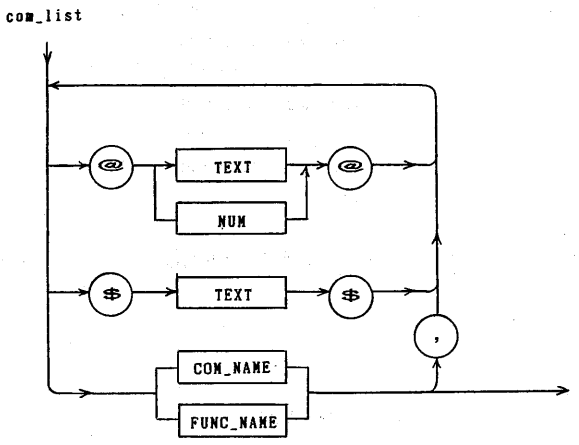


図６．　基本ループ

ｋｅｙ＿ｎａｍｅはキーの組合せによるオートマトンの記述で、ｃｏｍ＿ｎａｍｅはｋｅｙ＿ｎａｍｅに結合されたコマンドのマクロ名、ｃｏｍ＿ｌｉｓｔには図８の構文に従ってコマンドが記述される。これらがリスト状になってテーブルを構成する。

５．２　ファンクション・テーブル

このテーブルは、ｃｏｍ＿ｌｉｓｔに書かれるｆｕｎｃｔｉｏｎ＿ｎａｍｅとｐａｓｃａｌの実行文の中のプロシジャを対応づけるためのテーブルである（図９）。



TEXT :　全ての文字
NUM　:　{ 0 | 1 ... | 9 }
COM_NAME　: { a | ... | z }
FUNC_NAME : function table　に定義されているfunctionの名前

@　～　@　はargumentへ、
$　～　$　はSub_argumentへ渡される。

図８．　com_listの構文図

ファンクションテーブル

NO.	function_name
1	insert
2	delete
3	back_one
' '	' '
n	' '
n+1	end_of_table

```
case No of
  1 : insertion;
  2 : deletion;
  3 : backward_one;
  '
  '
  n :
end;
```

図９．　ファンクションテーブルとpascal実行文

５．３　コマンドの実行

キーボードから入力されるキーによって、ｋｅｙ＿ｎａｍｅに記述されたオートマトンを遷移する。オートマトンの最終状態になった時点で、ｋｅｙ＿ｎａｍｅに結合されたｃｏｍ＿ｌｉｓｔがインタプリタに渡される。インタプリタは、ｃｏｍ＿ｌｉｓｔの記述を解釈し実行する。ｃｏｍ＿ｌｉｓｔ中にｆｕｎｃｔｉｏｎ＿ｎａｍｅが見つかった場合にはファンクション・テーブルを引き、それをＰａｓｃａｌの実行文によって実行する。また、マクロ名が見つかった場合には、コマンド結合テーブルからそのマクロ名を持つコマンドを探し、再帰的に呼ばれたインタプリタにｃｏｍ＿ｌｉｓｔを渡す。これがコマンド実行のアルゴリズムである。ただし日本語の文字については、テーブルに持つにはあまりに多すぎるので、特別に論理キーという概念を導入し、入力された一文字を挿入するという形にした。

上述のアルゴリズムでは、マクロのネストは許されるが、再帰的なマクロ呼出しは禁止される。

５．４　コマンドの拡張・変更

以上のことから、実行時にコマンド結合テーブルの追加・変更を可能にすることによって、本機能は実現される。

ｋｅｙ＿ｎａｍｅ、ｃｏｍ＿ｎａｍｅについては問題ないであろう。ｃｏｍ＿ｌｉｓｔは３章で述べたテキスト・タイプであり、再帰的に呼ばれるエディタによって直接編集される。

６．　おわりに

ＰＭＡＣＳは現在エプソンＱＣ－１０上で動いている。本原稿はＰＭＡＣＳで作成した。ハードウェアに制約のため、再表示のスピードに改善の余地があるが、ＴＵＴコードによって高速に入力できることから、打ち込んだ後の疲労も少なく、修正も各種コマンドによりスムースに行えるなどの使い勝手を実現している。

参考文献：

【１】　高橋延匡：　日本文入力の現状と展望、情報処理vol.23,　No. 6pp.518～528 (1982)

【２】　高嶋孝明：　タッチタイプ入力方式とその練習法、情報処理学会日本文入力方式研究会資料4-3 、pp. 10 (1982)

【３】　桑原　他：　パーソナル・コンピュータによる日本語タッチタイプ入力システムの試作、昭和57年度電気関係学会東海支部連合大会(1982)

【４】　池田勇二　他：　タッチタイプ入力による日本語エディタの試み、情報処理学会第27回全国大会(1983)

【５】　平賀譲：　日本語テキストエディタＫＥＤの紹介、東京大学大型計算機センターニュース、 vol. 13, No. 12 (1981)

【６】　Craig A. Finseth：　A Cockbook For An EMACS、MIT/LCS/TM-165 (1980)

【７】　斎藤康己　他：　日本語スクリーン・エディタＪＭＡＣＳの概要、情報処理学会第24回全国大会(1982)

【８】　花田収悦：　プログラム設計図法、企画センター(1983)

【９】　阿部雅人：　漢字処理の現状、インフォメーションサイエンス、vol. 2、No. 4 、pp. 81～87(1983)