日本語ワープロにおける再変換処理について

河　上　喜代子

1. はじめに

情報化時代の発展に伴い，以前には想像もつかなかったような情報関連機器が，家庭や職場にも導入されるようになってきた。その中でも特に普及率の高いものの１つとして，ワードプロセッサ（以下ワープロと略す）がある。ひとロにワープロと言っても，それは大きく２つに分けられる。１つは，それ自体ワープロの機能しか持たない専用機で，もう１つは，パソコンのアプリケーションソフトとしてのワープロである。

ところで，欧米においては約100年ほど前からタイプライターが普及しており，すでにワープロの土壌はできあがっていた。そのため，コンピュータの発展とともに，自然にワープロが誕生した。しかし日本では，ごく最近になってやっと日本語ワープロが普及するようになったのである。このように開発が遅れた理由としては，日本語がなか漢字混じり文で構成されていること，同音異義語が多いことなどがあげられる。同音異義語の識別は人間にとっては容易であっても，コンピュータにとっては非常に困難なことである。つまり，日本語の持つ特質が日本語ワープロの開発を遅らせる原因であったといえる。

現在では，ワープロ専用機，パソコンのソフトとも，数多くの製品が市販されている。そのどれをとっても，かなまたはローマ字で入力した文を自動的に漢字に変換する機能を備えているので，ユーザーは容易に文書を作成，編集できるようになった。

本報告では，パソコンのソフトとしてのワープロに多くみられる再変換について述べる。再変換とは，文書作成中に誤って変換された部分を変換し直すという処理である。変換によって多くの文書情報が作成されるが，それらは文節を１単位として保存されているので，再変換処理では文節を意識する必要がある。しかし，文節の認識は人によってまちまちであり，日本語の文法についてかなりの知識を必要とすることもある。そこで，少しでもユーザーが文節を意識しなくてすむようなアルゴリズムを提案する。

2. データ構造

本報告で提案するアルゴリズムを説明する前に，その基本となるデータ構造について述べる。

一般に，漢字への変換処理とは，キーボードからかな（カナ）またはローマ字で入力されたよみがなをもとに行われるものである。この変換処理の時にさまざまな文書情報が作成され，それぞれ記憶領域上のきまった場所に格納される。ここでは，行単位で格納する方法をとっており，その格納場所の構造を示したものが図１である。

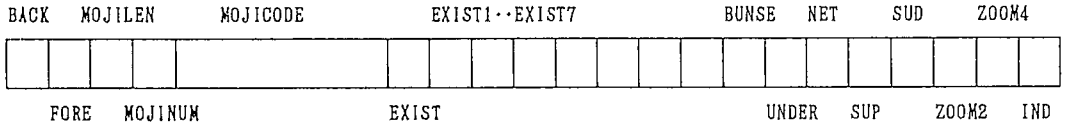
（ａ）行は，記憶領域上での１行の表現であり，このような構造で１行分の文書情報が格納されるようになっている。また，このように格納されたすべての行は，それぞれその前後の行と BACK, FORE という２つのポインタでつながれている。ポインタとは，次の情報が格納されている記憶場所を示す値で，その値を変えることによってデータを差しかえたり，データ構造そのものを変えたりすることができる。（ｂ）文節，（ｃ）編集は，

それぞれ行に含まれる文節，編集の情報を行頭に近いものの順に格納するポインタで，（a）行のBUNSE，UNDER～IND につながれている。各項目についての詳しい説明は，図1に添付しておく。

ここで，“私は短大の学生です„ という例文を用いて，文書情報がどのように格納されるかを説明する。

まず，“わたしはたんだいのがくせいです„ と，入力されたよみがなをもとに，あらかじめ用意された日本語辞書を探索する。そして，読みがなに対応するいくつかの候補の中で最適とされるものを取り出す。いま，この文が，“私は„，“短大の„，“学生です„ という3つの文節に分けられたとする。各文節ごとに，再変換可能判定値ABLE（初期値は0で再変換可能），文節開始位置STAPOS，文節長LEN，次の文節を示すポインタ値NEXT1が与えられ，読みがなYOMIGANAはJIS漢字コードに準拠したコード（表1参照）が与えられる。仮に，それぞれの文節の情報がA，B，Cという場所に格納されたとすると，文節の情報は表2のようにまとめられる。

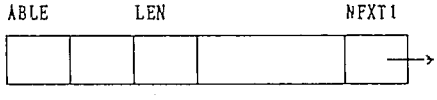
文節以外の情報については，この文だけで文書が構成されていることから前後の行を示すポインタBACK，FOREはNIL（NILとは次の情報が存在しないことを示す）である。また，この文が全角9文字であることから文字数MOJINUMは9，文字長MOJILENは18となり，文字のコードを格納するMOJICODEには表3に示したコードがそのまま代入される。アンダーライン，網掛けなどの編集の情報はそれぞれ0であるので，そのポインタはすべてNILである。これらを図1（a）行の決められた場所に格納した状態を示したものが図2である。文書情報はこのような形で



BACK,FORE	前後の行の格納場所を示すポインタ
MOJILEN	行の長さ（半角を1、全角を2として数える）
MOJINUM	行に含まれる文字数
MOJICODE	文字コード（最大120文字分）
EXIST	行に含まれる文節数
BUNSE	行の先頭文節の情報の格納場所を示すポインタ

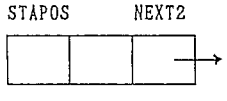
EXIST1‥EXIST7	行に含まれる編集のそれぞれの数
UNDER‥IND	編集の情報の格納場所を示すポインタ
編集・・アンダーライン、網掛け、上付き、下付き、倍角、4倍角、インデント	

（a）行



ABLE	再変換可能判定値
STAPOS	行頭から数えた文節開始位置
LEN	文節の長さ（半角を1、全角を2とする）
YOMIGANA	文節の読みがな
NEXT1	次の文節の情報の格納場所を示すポインタ

（b）文節



STAPOS	行頭から数えた編集開始位置
TERPOS	〃　終了位置
NEXT2	次の編集の情報の格納場所を示すポインタ

（c）編集

図1：データ構造

表１：よみがなコード表

よみ	わ	た	し	は	た	ん	だ	い	の	が	く	せ	い	で	す
ｺｰﾄﾞ	246F	243F	2437	244F	243F	2473	2440	2424	244E	242C	242F	243B	2424	2447	2439

（注） 文字コードは１６進４桁で表される

表２：例文の文節データ

文　節	格納場所	ABLE	STAPOS	LEN	YOMIGANA	NEXT1
私は	A	0	1	4	246F243F2437244F	B
短大の	B	0	5	6	243F2473244024244E	C
学生です	C	0	11	8	242C242F243B2424244724 39	—

表３：変換された文字コード表

文字	私	は	短	大	の	学	生	で	す
ｺｰﾄﾞ	3B64	244F	433B	4267	244E	3358	4038	2447	2439

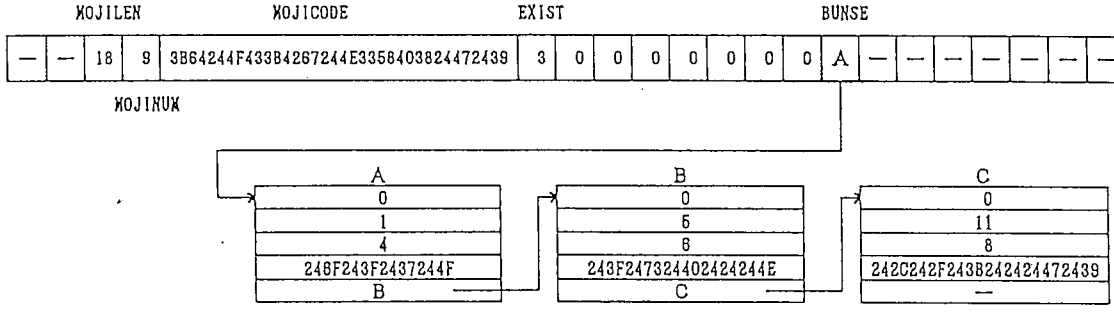


図２：例文の文書情報格納状態

格納されている。

このデータ構造を用いることの利点として，以下のことが挙げられる。

1. 行，文節，編集のデータ構造が，それぞれ１行，１文節，１編集単位のポインタ形式なので，必要に応じて使用メモリを広げればよい。つまり，あらかじめ確保しておく必要がないので，メモリの節約になる。
2. 挿入，削除にともなう文書情報の修正が容易である。

3. アルゴリズム

ここで提案する再変換のアルゴリズムを図３，図４に示す。これは，ユーザーが少しでも文節を意識せずに処理できるようにと考えたものである。このアルゴリズムでは，再変換キー（再変換の機

能を割り当てられたキー）が押された時に，カーソルが文節のどの位置にあっても処理できるようになっている。

いま１つの文書があったとき，その文字の位置を点座標（X，Y）で表すことにする。ただし，原点を（１，１）とし，Y軸は一般の座標とは異なって下向きを正とする。この座標系におけるカーソル位置のX座標と文節開始位置とを比較することで，処理の対象となる文節を探索し，その文節が再変換可能かどうか判断する。判断する理由は，文節の途中に文字が挿入されたり，逆にある文字が削除された場合に，その文節長は変更可能だが，文節の読みがなを変更するのは困難だからである。つまり，変換に必要な読みの情報が格納できないため，再変換の対象から外されるのである。以下，カーソル位置の文節を第１文節，次の文節を第２文節，処理の対象となる文節の数を処理文節数と呼ぶことにする。

まず，カーソル位置から第１文節を探索する。第１文節の位置は，前の行の最終文節である場合と，カーソル行のいずれかの文節である場合の２つに大きく分けられる。どちらの場合でも，第１文節が再変換不可能なら処理文節数は０，そうでなければ１で，さらに第２文節が再変換可能なら２となる。（第２文節が存在しなければ１）そして，処理文節数が１，または２の時，その文節の先頭文字のコードが格納されている配列番号を計算する。第２文節を探索する理由は，第１文節の区切りが文法的に間違っている場合に，区切りを延長することもありうるからである。ただし，第２文節が存在しない場合や再変換不可能な場合は，第１文節内でのみ区切れの変更が可能なだけである。

次に，文節の読みをもとに変換処理を行い，もとの文書情報と置き換える。変換処理とは，前に示したように，読みに対応する候補の中で最適とされるものを取り出すことである。そして，他の文節の情報を作成し，文字のコードとともに決められた場所に格納するのである。その際，誤って必要な情報を消去することがないように，まず，不要になった情報を消去してから新しい情報を挿入するという処理を行うようになっている。また，１度再変換された文節でも再変換可能であればくり返し処理できる。

4. おわりに

実際，本報告のアルゴリズムを用いたとしても，文節を全く意識せずに再変換を行うのは難しいと考えられる。というのは，この機能の本来の目的は，間違って変換された部分の修正を効率よく行うというところにある。そのため，再び辞書を探索しなくてすむように，変換の時に得られた候補の情報を保存している。しかし，それは，あくまでも区切れ，つまり文節が変わらないということを前提としている。意味的に間違って区切られているなら，保存されている候補は役にたたない。そこで，区切れの位置を変更して新たに変換することになるが，この時に文節の意識が必要である。ただ，文節を意識するといっても，意味的におかしくない場所で区切るという程度のものなので，それほどユーザーの負担にはならないはずである。

謝辞

本研究を進めるにあたり，懇切丁寧なご指導をくださいました信州大学工学部岡本助教授に深く感謝いたします。

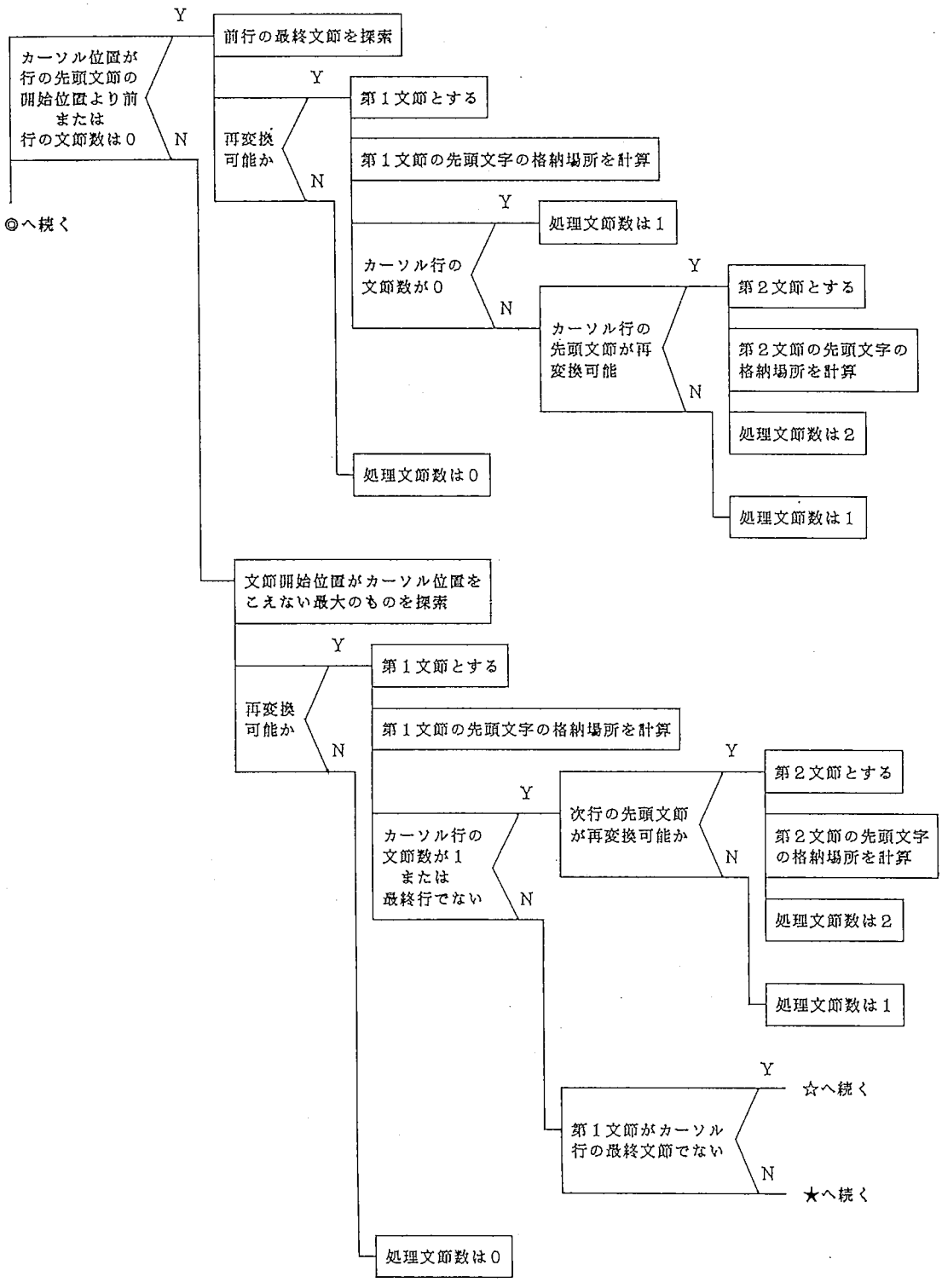


図3：再変換のアルゴリズム（その1）

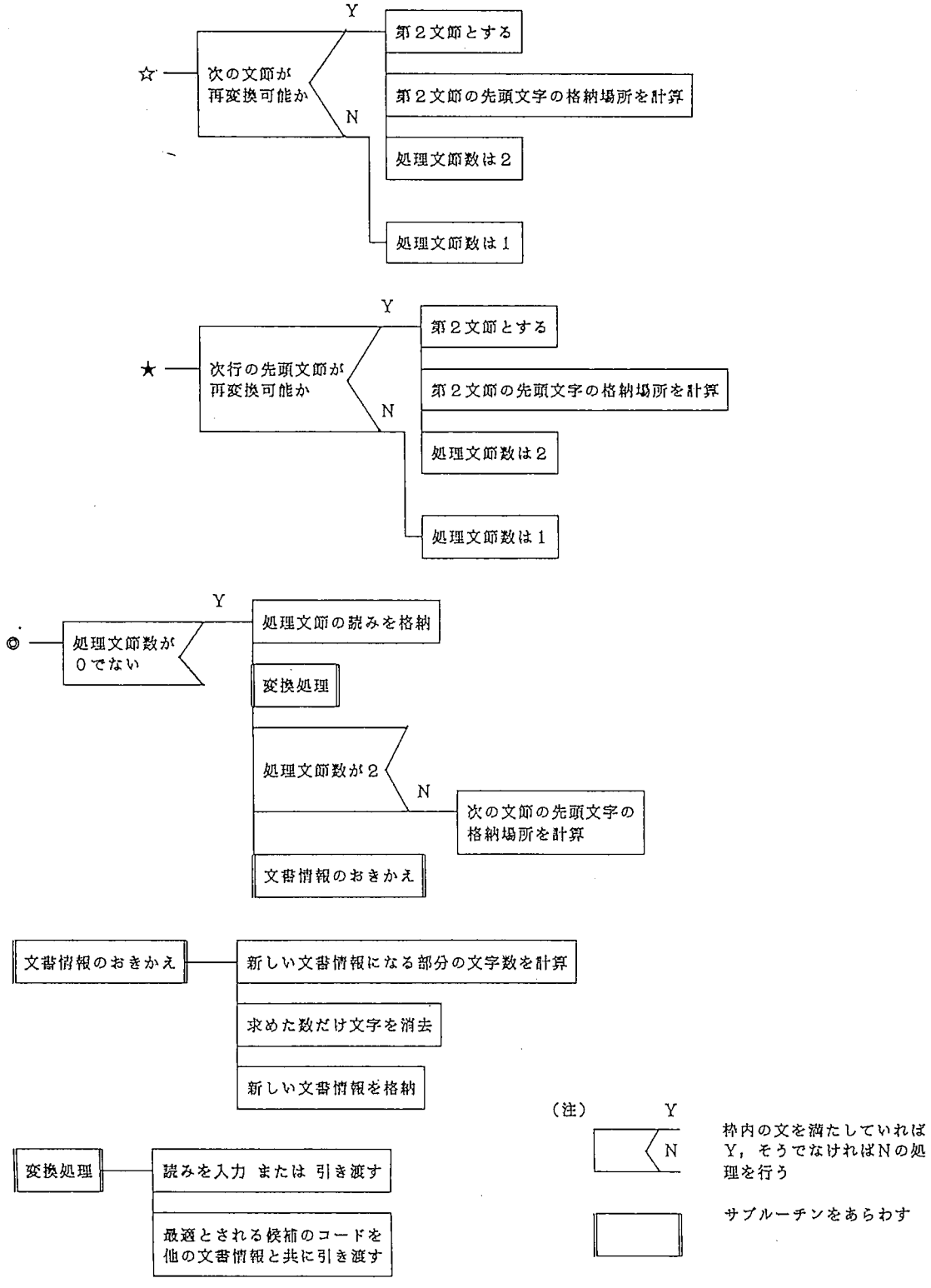


図４：再変換のアルゴリズム（その２）

付　録

《再変換のプログラムリスト》

```
procedure  saihen;
  var flag1,flag2,flag3,flag4 : boolean;
      i,j,len,num,old_len,old_num,new_len,new_num,sai_start1,sai_start2 : word;
  begin
    flag1:=false;
    flag2:=false;
    flag3:=false;
    flag4:=false;
    if (retsu < curl^.bunse^.stapos)or(curl^.exist = 0) then
       begin
         curl1:=curl^.back^.bunse;
         while curl1^.next1 <> nil do
           curl1:=curl1^.next1;
         if curl1^.able = 0 then
            begin
              kazu:=curl^.back^.mojinum*2-1;
              len:=curl^.back^.mojilen;
              while len > curl1^.stapos do
                begin
                  if (curl^.mojicode[kazu] = 85h)or
                     (curl^.mojicode[kazu] = 86h)and
                     (curl^.mojicode[kazu+1] = 40h) then len:=len-1
                  else len:=len-2;
                  kazu:=kazu-2
                end;
              if len < curl1^.stapos then kazu:=kazu+2;
              sai_start1:=kazu;
              if curl^.exist = 0 then sai_num:=1
              else if curl^.bunse^.able = 0 then
                      begin
                        flag1:=true;
                        sai_num:=2;
                        keep1:=curl^.bunse;
                        kazu:=1;
                        len:=0;
                        while len < keep1^.stapos-1 do
                          begin
                            if (curl^.mojicode[kazu] = 85h)or
                               (curl^.mojicode[kazu] = 86h)and
                               (curl^.mojicode[kazu+1] = 40h) then len:=len+1
                            else len:=len+2;
                            kazu:=kazu+2
                          end;
                        sai_start2:=kazu
                      end
                   else
                      begin
                        sai_num:=1;
                        flag4:=true
                      end
            end
```

```
      else sai_num:=0
    end
  else
    begin
      curl1:=curl^.bunse;
      while (retsu > curl1^.stapos)and(curl1^.next1 <> nil) do
        begin
          keep1:=curl1;
          curl1:=curl1^.next1
        end;
      if (retsu <= curl1^.stapos)and(not((keep1 = curl^.bunse)and
         (curl1^.next1 = nil))) then curl1:=keep1;
      if curl1^.able = 0 then
         begin
           kazu:=1;
           len:=0;
           while len < curl1^.stapos-1 do
             begin
               if (curl^.mojicode[kazu] = 85h)or
                  (curl^.mojicode[kazu] = 86h)and
                  (curl^.mojicode[kazu+1] = 40h) then len:=len+1
               else len:=len+2;
               kazu:=kazu+2
             end;
           sai_start1:=kazu;
           if curl^.exist = 1 then
              if (curl^.fore <> nil)and(curl^.fore^.exist <> 0)and
                 (curl^.bunse^.able = 0) then
                 begin
                   flag2:=true;
                   keep1:=curl^.fore^.bunse;
                   kazu:=1;
                   len:=0;
                   while len < keep1^.stapos-1 do
                     begin
                       if (curl^.mojicode[kazu] = 85h)or
                          (curl^.mojicode[kazu] = 86h)and
                          (curl^.mojicode[kazu+1] = 40h) then len:=len+1
                       else len:=len+2;
                       kazu:=kazu+2
                     end;
                   sai_start2:=kazu;
                   sai_num:=2
                 end
              else sai_num:=1
           else if curl1^.next1 <> nil then
                   if curl1^.next1^.able = 0 then
                      begin
                        flag3:=true;
                        keep1:=curl1^.next1;
                        kazu:=sai_start1;
                        len:=curl^.stapos-1;
                        while len < keep1^.stapos-1 do
                          begin
                            if (curl^.mojicode[kazu] = 85h)or
                               (curl^.mojicode[kazu] = 86h)and
```

```
                    (curl^.mojicode[kazu+1] = 40h) then len:=len+1
                  else len:=len+2;
                  kazu:=kazu+2
                end;
              sai_start2:=kazu;
              sai_num:=2
            end
          else if (curl^.fore^.exist <> 0)and
                  (curl^.fore^.bunse^.able = 0) then
                  begin
                    flag2:=true;
                    keep1:=curl^.fore^.bunse;
                    kazu:=1;
                    len:=0;
                    while len < keep1^.stapos-1 do
                      begin
                        if (curl^.mojicode[kazu] = 85h)or
                           (curl^.mojicode[kazu] = 86h)and
                           (curl^.mojicode[kazu+1] = 40h) then
                           len:=len+1
                        else len:=len+2;
                        kazu:=kazu+2
                      end;
                    sai_start2:=kazu;
                    sai_num:=2
                  end
                else sai_num:=1
        else sai_num:=1
    end
  end;
if sai_num <> 0 then
  begin
    for i:=1 to 20 do
      sai_yomi[1,j]:=curl1^.yomigana[j];
    if sai_num = 2 then
       for i:=1 to 20 do
         sai_yomi[2,j]:=curl1^.yomigana[j];
    color(15);
    locate(0,21);
    write('再変換');
    color(7);
    beta;
    if sai_num = 2 then
       begin
         old_len:=length(curl1^.yomigana);
         new_len:=0;
         i:=1;
         while i <= b_suu do
           begin
             new_len:=new_len+length(table[i].yomi);
             i:=i+1
           end;
         sai_start2:=kazu;
         return_text
       end
    else
```

```
        begin
          len:=0;
          while len < curl1^.stapos+curl1^.len-1 do
            begin
              if (curl^.mojicode[kazu] = 85h)or
                 (curl^.mojicode[kazu] = 86h)and
                 (curl^.mojicode[kazu+1] = 40h) then len:=len+1
              else len:=len+2;
              kazu:=kazu+2
            end;
          sai_start2:=kazu;
          return_text
        end
    end;
```

《文書情報のおきかえのプログラムリスト》

```
procedure return_text;
  begin
    now:=sai_start1;
    retsu:=curl1^.stapos;
    if (flag1 or flag4) then
       old_num:=(curl^.back^.mojilen-sai_start1+sai_start2) div 2
    else if flag2 then
            old_num:=(curl^.mojilen-sai_start1+sai_start2) div 2
         else old_num:=(sai_start2-sai_start1) div 2;
    if (flag1 or flag4) then gyoudown;
    i:=0;
    while i < old_num do
      begin
        i:=i+1;
        del_syori
      end;
    code_insert
  end;
```