ソフトウェア工学 31－4
（1983. 8. 31）

既存ソフトウェアへの日本語テキスト処理機能の追加
－ SNOBOL4処理系の場合 －

牛島 和夫，吉田 和幸，黒坂 輝彦＊
（九州大学工学部）

1．はじめに

近年，ハードウェア技術の進歩により，計算機により日本語文字（漢字，ひらがな，カタカナ，ローマ字および日本語の文章に使用する特殊文字をまとめてこう呼ぶことにする）を扱うことが可能になってきた．最近国内で発表される計算機はそのほとんどが「漢字が扱えます」とうたっている．しかし，実際に日本語文字を扱うプログラムを書こうとすると，計算機の細部まで知っている必要があったり，面倒な手順を要求されたりする場合が多い．

筆者らの利用する九州大学大型計算機センターでは1980年から富士通の提供するJEF（Japanese processing Extended Feature）[1]と言う日本語情報処理機能を導入している．JEFの利用形態として最もよく行われているものの一つに，日本語ワードプロセッサとしての利用がある．しかしもっと高度な使い方（たとえば[2]にある文書作成支援ツールのようなこと）をしたければ，自分でプログラムを組む必要が出てくる．このためにはCOBOL，FORTRAN，PL／Iの三つの言語でJEFの機能を使えるようになっている．ところが、これらの言語は非常に「手続き的」であったり，固定したフォーマットのファイルしか扱えなかったりして，日本語文字処理を伴うプログラムを書くのは，かなりむずかしい．

SNOBOL4[3]は挿入，削除を伴ったパターンマッチを一文で書け，その組み合わせにより文字列に関するかなり複雑な操作も簡単に記述できる．従ってテキストファイル処理などに際して適当なツールがない場合その場でプログラムを作ってみるというラピッドプロトタイピングに適した言語である[4]．我々の研究室ではFORTRANソーステキスト中からサブルーチンを切り出すといった，汎用のテキストエディタの能力を超えた文字列処理のためのツールとしてSNOBOL4を便利に使用している．

そこでこのSNOBOL4に日本語文字処理機能を付加することで，日本語文字，EBCDIC文字が混在しているテキストの高度な処理手続きを簡単に記述できるようにすることを考えた．以下，その実現の過程と既存ソフトウェアの一部変更という形でこれを可能にした理由とについて述べる．第2章と第3章はその準備である．第2章ではJEFが提供する日本語文字処理環境の概要について述べる．第3章では対象となった既存のSNOBOL4処理系の概要を示す．第4章でこの処理系に追加した日本語テキスト処理機能の設計とその実現，第5章で処理速度の改善について述べる．第6章では改善の効果を実行例で示す．第7章では既存ソフトウェアを手直しすることによって日本語テキスト処理機能が比較的容易に実現できた理由を考察する．

なお，日本語SNOBOL4は九州大学大型計算機センターのFACOM M－200 OSⅣ／F4のもとで1983年3月から一般の利用に供している[5]．

2．日本語文字処理環境

OSⅣ／F4が提供する日本語文字処理のソフトウェアとそれが用いるハードウェアについて簡単に述べる．

（1）文字コード　従来から使われてきたEBCDICコード（1バイト）に加えて漢字等の日本語文字を表わすJEFコード（2バイト）が使われる．この2種類の文字コードの区別にはシフトコードを用いる．JEFコードは上位バイト，下位バイトに分けて考えることができる．上位，下位バイトはそれぞれ16進で41～FE，A1～FEの値をとる．

（2）データセット　JEFが扱うデータセットには2種類ある．一つは従来のエディタが扱うデータセットと同様の形式を持つもので日本語文字は右筆というエディタで編集する．右筆ではEBCDIC文字も当然編集の対象となっている．他の一つは文章処理専用のデータセットで従来のエディタや右筆では扱えない構造になっている．このデータセットの編集にはFDMS和文エディタという専用エディタが使われる．

（3）漢字端末　右筆と和文エディタが動作する端末はフルスクリーン型の端末で，そこでは漢字等の日本語文字はEBCDIC文字の2倍の幅で表示される．

＊現在エプソン株式会社

(4) 漢字プリンタ　ドットプリンタとレーザビームプリンタの2種類ある．ドットプリンタでは日本語文字はＥＢＣＤＩＣ文字の2倍の幅で印刷され，レーザビームプリンタでは日本語文字の幅はＥＢＣＤＩＣ文字の2倍と1．2倍の2種類の幅を選択できる．

(5) プログラミング言語による日本語処理　ＪＥＦではＦＯＲＴＲＡＮ，ＣＯＢＯＬ，ＰＬ／Ｉで日本語文字を扱う手段を提供している．これらの言語は手続き的であるため文字列処理を行うプログラムは複雑になりすぎる．また，ＪＥＦが提供する機能では，日本語文字とＥＢＣＤＩＣ文字との混在は固定的なものしか許されていない．そのため文章ファイルのような日本語文字とＥＢＣＤＩＣ文字が自由に混在しているデータの処理は一層複雑になる．そのため計算機を専門としない一般ユーザが文章ファイルを扱うプログラムをこれらの言語を用いて作成することはほとんど不可能であるといってもよい．ＪＥＦが提供する環境だけでは手軽に日本語文章を扱うプログラムを作成できるとは言い難い．以下では日本語文字とＥＢＣＤＩＣ文字が任意に混在するテキストを日本語テキストと称する．

3．ＳＮＯＢＯＬ４原処理系の概要

日本語テキスト処理機能の追加作業を説明する準備として，本章では既存のＳＮＯＢＯＬ４処理系の構成を簡単に述べる．原著者の許可を得て入手したＳＮＯＢＯＬ４処理系は，ＳＩＬ（ＳＮＯＢＯＬ４　Ｉｍｐｌｅｍｅｎｔａｔｉｏｎ　Ｌａｎｇｕａｇｅ）という抽象言語で書かれたもの（ＯＳ３６０用）である［6］．

このＳＮＯＢＯＬ４処理系はトランスレータ，インタプリタ，記憶管理，システムインタフェースという四つの部分から構成される［6］．トランスレータはＳＮＯＢＯＬ４プログラムをプレフィックスコードに変換し，それをインタプリタが解釈実行する．記憶管理とシステムインタフェースはトランスレータとインタプリタの両方から使われる．記憶管理は領域の割り当て，データの記憶，記憶域の再構成等を行う．システムインタフェースは入出力やプログラム割込み等を受け持つ．

このＳＮＯＢＯＬ４処理系を記述しているＳＩＬという抽象言語は図1のような階層構成により実現されている［6］．最も外側（図1（ａ））がＳＩＬの世界で１３１個の命令を持つ．その内側（図1（ｂ））が１３１個の命令をアセンブラのマクロ機能を用いて定義したＳＩＬマクロの層である．その内側に入出力等ＯＳに依頼する部分を実行時ルーチンとしてまとめた層がある（図1（ｃ））．これらのルーチンはアセンブリ言語で記述され，ＳＩＬマクロから呼び出される．実際の入出力はＦＯＲＴＲＡＮの実行時ルーチンに処理を依頼し，異常終了時処理，時間管理等のＯＳに関係する部分は直接ＯＳのマクロ命令を用いているものもある．これらが最も内側の層を構成する（図1（ｄ））．

4．日本語テキスト処理機能の追加

4．1　日本語テキスト処理機能の設計

ＳＮＯＢＯＬ４への機能追加にあたり次の目標を設定した．

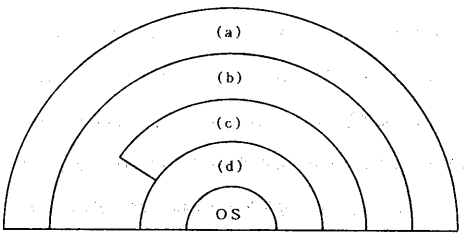
(1) 日本語文字とＥＢＣＤＩＣ文字を同等に扱えるようにすること．シフトコードのような印刷されない文字をＳＮＯＢＯＬ４利用者が意識しなくてもすむようにする．

(2) 従来のエディタや右筆で作成したデータセットと同様にＦＤＭＳ和文エディタで作成したデータセットも扱えること．このためには，可変長ファイルを自由に読み書きできる必要がある（和文エディタが作成するデータセットは可変長ファイルである）．原処理系ではＦＯＲＴＲＡＮ実行時ルーチンを使用しているために入力データは固定長ファイルに入っているものしか扱えない．

(3) 読み易いプログラム，わかりやすいプログラムを書くための一つの手段としてプログラム中の変数，ラベル等に日本語文字が使用できるようにする．

(4) ＴＳＳ環境で使用するのに便利な機能を加える．

このうち，（1），（2）が重要な目標である．これらを実現するた



(a) ＳＩＬ言語
(b) ＳＩＬマクロ定義
(c) 実行時ルーチン
(d) ＯＳのマクロ命令，ＦＯＲＴＲＡＮ実行時ルーチン

図1．ＳＩＬ処理系の階層構成

めには，

（a）文字列処理のアルゴリズムの実現はかなり面倒なのでＳＮＯＢＯＬ４原処理系のアルゴリズムをなるべくそのまま使えるようにする．そのためには，日本語文字１字をＥＢＣＤＩＣ文字１字と同等に扱えるようにする．

（b）入出力機能を強化し，可変長ファイルを扱えるようにするほか，和文エディタのファイルではその形式に合わせて読み書きできるようにする．

の２点を解決しなければならぬ．

この目標を実現するため，次の変更設計を行った．

（1）文字の内部表現　ＳＮＯＢＯＬ４処理系の文字列処理のアルゴリズムをなるべくそのまま使えるようにするためには，文字コードはすべて同じビット長である必要がある．ＳＮＯＢＯＬ４処理系内部で日本語文字とＥＢＣＤＩＣ文字を同等に扱えるようにＥＢＣＤＩＣ文字の左側（上位バイト側）に１バイトのパッディングバイト（X'00'）を加え，形式的に２バイトで表現することにする．たとえばA，B，CをＥＢＣＤＩＣ文字とすると文字列

AB漢字C

の１６進数表現はデータセット内で

C1C228B4C1BBFA29C3（9バイト，下線部はシフトコード）

処理系内部で

00C100C2B4C1BBFA00C3（10バイト）

のようになる．データセット内と処理系内とで表現の差ができるので，ＳＮＯＢＯＬ４実行時ルーチン（図１（ｃ）に属する）の入出力部でその変換を行う．また，１文字が２バイトになるためＳＩＬマクロ定義（１３１個）のうち文字列を扱うもの（２０個）を書き替える．そのうち９個はＳＩＬマクロ定義内の変更だけでよく，他の１１個は実行時ルーチン中にある下請けルーチンの変更が必要である．たとえば，前者の例ではデータ領域を確保するＳＩＬマクロＢＵＦＦＥＲの定義

```
        MACRO
&LOC    BUFFER  &N
&LOC    DC      (&N)X'40'
        MEND
```

を次のように変更する．

```
        MACRO
&LOC    BUFFER  &N
&LOC    DC      (&N)X'0040'
        MEND
```

ここでMACRO，MENDはマクロ定義の開始，終了を，BUFFERはマクロ名を表す．&のついた名前はマクロの仮引数である．DCはアセンブリ言語のデータ定義命令でここでは１６進定数を定義している．

さらに文字コードを８ビットから１６ビットに変更するのでコンパイル時および実行時に使用する字句解析のための表もそれに伴って変更する必要がある．コンパイル時に使用する表は２４個あり，それらは変数名，ラベル名，文字列等をソースプログラムから切り出すために使用される．表の内容は変更されない．実行時に使用する表は上の２４個のほかパターンマッチ処理のために実行中に書き替えながら使用する動的な表が１つある．表に関係するＳＩＬ命令は３個ある．そのうち２個はマクロ定義内の変更が必要であり，１個はその下請けルーチンの変更が必要である．変数名，ラベル名等として日本語文字も使用できるようにするためにもこれらの表を変更する必要がある．

（2）入出力　ＳＮＯＢＯＬ４原処理系ではＦＯＲＴＲＡＮ実行時ルーチンの入出力機能を借りて実際の入出力を行っているが，ＦＯＲＴＲＡＮ実行時ルーチンの仕様は公開されたものではないのでその機能の細部はわからない．したがって和文エディタで作成したファイル等の可変長ファイルからデータをＦＯＲＴＲＡＮ実行時ルーチンを用いて入力したとき，そのデータの長さを知る方法がない等の問題が生じる．そこでＳＮＯＢＯＬ４実行時ルーチン自身（図１（ｃ））が直接ＯＳのデータ管理マクロを使用して入出力を行うように書き改めることにした．これによって図１（ｄ）の層からＦＯＲＴＲＡＮ実行時ルーチンはなくなる．

（3）ＴＳＳ環境での使い勝手　ＴＳＳ環境での使い勝手をよくするために次の機能を追加する［5］．

（a）ＳＮＯＢＯＬ４プログラム中でＴＳＳコマンドからの実行時パラメータを受け取ることができる．

（b）ＳＮＯＢＯＬ４プログラム中からＴＳＳコマンドの実行ができる．

このうち（a）はＳＮＯＢＯＬ４処理系の初期設定時にＴＳＳコマンドからのパラメータをＳＮＯＢＯＬ４プログラムから読めるように処理系内部にコピーすることで実現する．（b）の実現にはＴＳＳコマンドを実行する外部関数を作成し，ＳＮＯＢＯＬ４の外部関数呼び出し機能を用いてＳＮＯＢＯＬ４プログラムと結合する．そのためＳＮＯＢＯＬ４処理系内部の変更はない．

入出力ルーチンを書き改めたことによりＳＮＯＢＯＬ４の組み込み関数の一つであるＯＵＴＰＵＴの仕様を次のように変更し，ほとんど使われず，実現が面倒なＢＡＣＫＳＰＡＣＥ関数を廃止する必要がある．

ＯＵＴＰＵＴ（出力変数，論理機番，制御文字）

従来はＯＵＴＰＵＴ関数の第３パラメータにはＦＯＲＴＲＡＮのＦＯＲＭＡＴ記述子を書き，改ページ，重ね打ち等を自由に行っていたが，入出力を実行時ルーチン中で直接行うようにするので，ＦＯＲＭＡＴ記述子を自由に使えなくなるためである［5］．ＯＵＴＰＵＴ関数の第３パラメータは省略可能なパラメータである．ＳＩＬで書かれたもの以外のＳＮＯＢＯＬ４処理系（ＳＰＩＴＢＯＬ等）のＯＵＴＰＵＴ関数の第３パラメータにはＦＯＲＭＡＴ記述子を使用しないので，互換性を考えて作った，既存のＳＮＯＢＯＬ４プログラム（たとえば［7］）にはこの仕様の変更は影響しない．

４．２　プログラム例

図２に日本語ＳＮＯＢＯＬ４プログラムの例を実行回数情報（図の左側の３列の数字が右側の各文の実行回数，成功回数，失敗回数［8］）付きで示す．変数名とラベル名に日本語文字を用いた．この例は入力の日本語テキスト中に現われる漢字列を抜きだしてその出現回数を計数するプログラムである．このプログラムでは'亜'が文字コードの上で漢字以外の文字との境界にある漢字として漢字の判定に利用している．このプログラムの入力テキストと出力結果の一部を図３に示す．

```
FACOM OSIV/F4 SNOBOL4(BELL TELEPHONE LABORATORIES) VER.3.11 + PROFILE + JEF (-830707-)     DATE 83/08/06
EXECUTION =  SUCCESS +  FAILURE :          S O U R C E   P R O G R A M

                                00000100*-------------------------------------*
                                00000200*     例題 : 漢字列計数               *
                                00000300*-------------------------------------*
        1           1         0 00000400            INPUT(.JINPUT,31,300)
        1           1         0 00000500            OUTPUT(.JOUTPUT,4)
        1           1         0 00000600            回数 = TABLE(30,10)
        1           1         0 00000700            亜 = '亜'
       58          58         0 00000800読み込み
       58          57         1 00000900            テキスト = JINPUT ' '                    :F(印刷)
       57          57         0 00001000            テキスト LEN(8) REM . テキスト           :F(読み込み)
     5522        5465        57 00001100文字判定     テキスト LEN(1) . 文字 =                 :F(読み込み)
     5465        5198       267 00001200            LGT (亜, 文字)                           :S (文字判定)
      267         267         0 00001300            単語 =
      441         441         0 00001400単語探索     単語 = 単語 文字
      441         441         0 00001500            テキスト LEN(1) . 文字 =
      441         267       174 00001600            LGT(亜,文字)                             :F(単語探索)
      267         267         0 00001700            回数<単語> = 回数<単語> + 1              :(文字判定)
        1           1         0 00001800印刷        JOUTPUT = '＊＊＊＊ 漢字列計数結果 ＊＊＊＊'
        1           1         0 00001900            JOUTPUT =
        1           1         0 00002000            回数 = CONVERT(回数,'ARRAY')             :F(終了)
        1           1         0 00002100            I = 1
      120         119         1 00002200単語印刷     JOUTPUT = 回数<I,1> ' = ' 回数<I,2>     :F(終了)
      119         119         0 00002300            I = I + 1                                :(単語印刷)
        1           1         0 00002400終了
        1           1         0 00002500END
NO ERRORS DETECTED IN SOURCE PROGRAM
```

図２．プログラム例（漢字列計数プログラム）

```
01014002Ｌ@CE@@CE@@CE@@CE@@CE@@CE@@CE@@CE@@CE@@CE@@CE@@RE+@@
01015000ファイルは@UL@ディレクトリ@UE@である．@NL@　　0　このファイ
01021000である．@NL@@LN=2@　@TB=S5,K,FL8+L20+@@MS@表５．ディレクトリ
01023000@RE@@DN@@TE@@DN@@NB@３．@UL@ＯＳ　／Ｆ４のもとでのＵＮＩＸフ
01024000　作業の手順を表６．に示す．@NL@@TB=SO,K,FL1+R4+L22+L16+,H@@
01024001@PT@ＪＳＤ－　ＧＥＮＥＲ@PE@@LN=2@@CE@
01024002ＵＮＩＸ配布用磁気テープファイルをブロック長を変えてディスク
01025000@LN=2@表１．の２番～４番のデータをそれぞれ@NA@直接アクセス・
01025003Ｍ＝Ｆ，ＤＳＯＲＧ＝ＤＡ）@PE@@RE@３@CE@@PT@ＵＮＩＸ－ＰＯ@P
01025004この時，ディレクトリ名を含めた完全ファイル名も同時に作成する
01026000@CE@@PT@@LN=1@Ｆ０８８１．ＵＮＩＸＯＢＪ@NL@Ｆ０８８１．ＵＮ
01026002@PE@@RE@４@CE@@PT@ＳＥＴＵＰ@PE@@CE@@LN=2@ＡＳＣＩＩ　ＥＢＣ
01026004@PE@@CE@@LN=2@　@@@・・・・　を分離符号として順編成から区分
01026005　ＤＳＯＲＧ＝ＰＯ）@PE@@RE@@TE@
01027000　@PT@ＪＳＤＧＥＮＥＲ@PE@は富士通のデータセットユーティリテ
01027001８１行のプログラムである．
01028000@PT@ＵＮＩＸＤＡ，ＵＮＩＸＰＯ，ＳＥＴＵＰ@PE@のソースプログ
01030000@PE@に入っている．@NB@４．@UL@ＯＳ　／Ｆ４におけるＵＮＩＸソ
01031000　@PT@ＵＮＩＸＰＯ@PE@が作成したファイル名一覧表が@PT@Ｆ０８
01032000目的のファイル名がｎ行目に表れたとする．ｎが４ケタに満たない
01033000その名前を@PT@ＵＮＩＸ##########@PE@とすると，目的のファイルは
01034000　なお，@PT@ＵＮＩＸ０００１～ＵＮＩＸ０３３７@PE@は，ドキュ
01035000@PT@ＵＮＩＸ０９３３～ＵＮＩＸ１２２２@PE@はオブジェクトディ
01036000@PT@'Ｆ０８８１．ＵＮＩＸ．ＴＥＸＴ'@PE@には対応するメンバ
01037000　ＳＨ　ＲＥＵ@NL@ＪＦＯＲＭＥ　'Ｆ０８８１．@ＵＮＩＸ'@P
01039000ＡＬＬＯＣ　Ｆ（ＳＹＳＰＲＯＣ）　ＤＡ（'Ｆ０８８１．ＴＳＳ
```

入力データ

```
＊＊＊＊ 漢字列計数結果 ＊＊＊＊

配布用磁気 = 6
構造 = 6
上 = 5
年 = 1
月 = 1
吉田 = 1
和幸 = 1
解析 = 4
検索法 = 2
付録 = 1
名一覧表 = 4
形式 = 3
記録形式 = 1
次 = 5
磁気 = 2
中 = 6
大 = 1
約 = 1
部分 = 2
分 = 2
表 = 16
内容 = 7
番号 = 8
下 = 1
```

出力結果

図３．漢字列計数プログラムの入力データと出力結果

5．処理速度の改善

日本語ＳＮＯＢＯＬ４処理系でプログラムを実行するときにプログラム中でパターン関数ＢＲＥＡＫ，ＳＰＡＮ，ＡＮＹ，ＮＯＴＡＮＹを多用していると実行速度が極端に遅くなることがわかってきた．これらの関数は上述の動的な表を書き替えて使用し，次の文字列とパターンマッチする．

ＢＲＥＡＫ　：引数に現われない文字から成るなるべく長い文字列．

ＳＰＡＮ　：引数に現われる文字から成るなるべく長い文字列．

ＡＮＹ　：引数に現われる文字のうちのどれか１文字．

ＮＯＴＡＮＹ：引数に現われない文字のうちのどれか１文字．

これらの関数は表を検索するＳＩＬ命令ＳＴＲＥＡＭを用いている．個々の計算機が実現しやすいアルゴリズムを採用できるように表の検索ルーチンは実行時ルーチン中（図１（ｃ））にあり，ＳＴＲＥＡＭ命令はそのルーチンを呼ぶように定義されている．

以下処理速度を遅くしている原因を究明するために表の構造と機能について調べる．

５．１　表の構造と日本語文字への適用

ＳＮＯＢＯＬ４原処理系（ＯＳ３６０用）が字句解析に用いる表の構造は機械命令ＴＲＴ（翻訳テスト命令）に大きく依存している．ＴＲＴ命令は，第１オペランドの文字列を先頭から１バイトづつ第２オペランドで指す表を用いて変換して行き，変換結果が０でなくなったところまたは文字列が尽きたところでやめる．したがって第２オペランドで指定される表は２５６個のエントリを持つ．たとえば先頭番地ＳＮＡＢＴＢと指された表で空白文字に対応する値（文字に対応するコマンド）はＳＮＡＢＴＢ＋Ｘ'４０'に入れておく．ＳＮＯＢＯＬ４原処理系では動的な表は図４のように，２５６バイトの項目部とそれに続く３個の機械命令からなる命令部で構成される．図４でｓｔｏｐやｅｒｒｏｒがコマンドである．

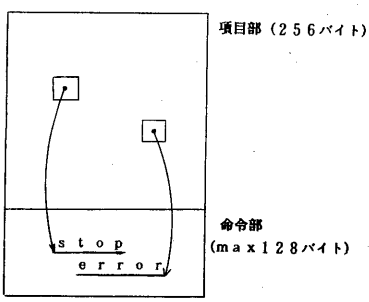


図４．ＳＮＯＢＯＬ４原処理系の字句解析表

この方法を日本語文字にそのままあてはめると，文字コードが２バイトなので一つの表が６４Ｋバイト（２＊＊１６）になり，表全体では１．６Ｍバイト（＝６４Ｋバイト×２５）にもなってしまう．そこで，表の圧縮を試みた．ＴＲＴ命令は１バイトを単位として処理するため使用できない．

字句解析ルーチンがコンパイル時に実際に区別すべき字種は演算記号，カッコ等２０種程度であるので，表には文字コードと対応するコマンドを並べて記入し，先頭から順に比較していくというアルゴリズムを採用して表の圧縮を行った．ただし，日本語文字，英字，数字は文字種としてはそれぞれをひとまとめにして考えてよいが，その数が多く，表中に全文字を入れることは大変なので表中の文字コードの項には日本語文字，英字，数字を表わすマークを書いておき，表検索ルーチン中でそのマークをみて当該字種の上限および下限と比較することで文字種の判定を行う．

静的な表はこれでよいが，このような構成では実行中に表を書き替えることは難しい．まず動作するものをつくるという方針に従って，動的な表の管理はＳＮＯＢＯＬ４原処理系のアルゴリズムを（ＴＲＴ命令は使えないが）そのまま用いることにし，動的な表は６４Ｋバイトの領域をとりあえず持つことにした．この表を実行中に書き替える際は，まず表全体をクリアした後に必要なところに値を書き込むようになっている．このためにこの表を使うＡＮＹ，ＮＯＴＡＮＹ，ＢＲＥＡＫ，ＳＰＡＮの実行には非常に時間がかかるのであった．

５．２　表の構造の変更

実行速度が遅くなった原因は，６４Ｋバイトの表をクリアするところにあると予想されるので，表自身を小さくする必要がある．表検索ルーチン中で文字コードを上位，下位バイトに分けて考え，表もそれに合わせて上位バイトで検索する主表とそれの各項からポインタで指される下位バイトのための副表に分けた（図５参照）．ＪＥＦコード（およびそのもととなっているＪＩＳ漢字コード）は上位バイト，下位バイトそれぞれだけみると連続しているが１６ビット全体として見ると不連続である．このため１６ビットをひとまとめにして表を引くよりも上位，下位バイトに分け，上位，下位の順で表を引くほうがよりＪＥＦコード（ＪＩＳ漢字コード）にあった方法であろう．

この方法では，表の書き替えは上位バイトによって主表を引き副表を探し下位バイトによってその副表を書き替える．表中にＪＥＦコー

ドの文字割当がない部分や下位バイトで区別する必要がない部分の副表は作らずにすませるので，６４Ｋバイトあった表を大幅に圧縮できる．主表に書き込む値は原処理系と同様なコマンドか副表へのポインタ（X'80'～X'FC'）かである．主表にｃｏｎｔｉｎｕｅ等のコマンドが直接書き込まれている場合には下位バイトの値に関係なく（副表は作らずに）コマンドの指示通りに文字列の解析を続ける．副表にはコマンドが書き込まれる．

この方法を採用して作った動的な表と先に別の方法で圧縮していた静的な表との大きさの上での差はほとんど解消できる．２種類の違う構造の表を管理することは保守上から好ましくないので静的な表もすべて動的な表と同じ構造にした．

上記の変更に伴う処理系の書き替えについて触れる．４．１節（１）文字の内部表現の項でも述べた，表に関係する３つのＳＩＬ命令は次のものである．これらは，次の動作をする．

ＣＬＥＲＴＢ：　表のクリア

ＰＬＵＧＴＢ：　表の文字に対応する位置に値を入れる．

ＳＴＲＥＡＭ：　表を用いて文字列を解析する．

このうちＳＴＲＥＡＭマクロだけが上で述べてきた表検索ルーチンを呼ぶ．そのルーチンのインタフェース仕様を変更しないのでＳＴＲＥＡＭマクロも変更の必要がない．

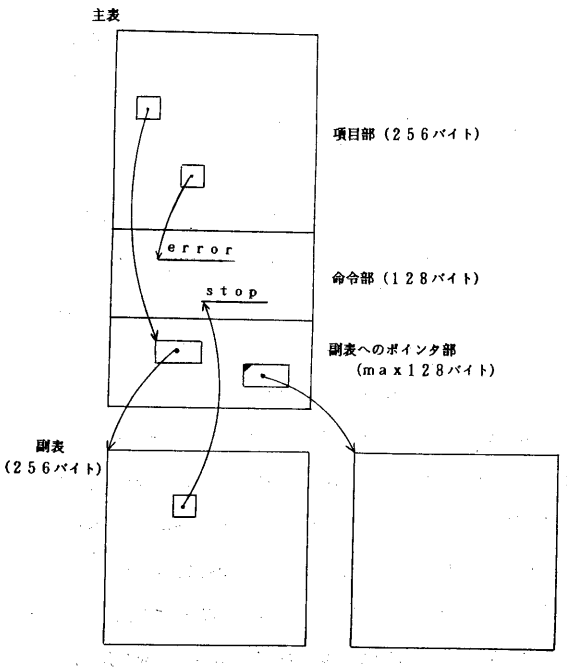


図５．日本語ＳＮＯＢＯＬ４処理系（改良版）の字句解析表

ＣＬＥＲＴＢは主表２５６バイトのクリアする命令と副表へのポインタ領域のクリアする命令とをマクロ内で展開する．主表２５６バイトをクリアする部分は原版とほぼ同じ形にもどった．

ＰＬＵＧＴＢは従来の単一構造の表では文字コードに対応する位置に値を書き込むだけであったのでマクロ内で展開していたが，表を階層構造にしたために，値の書き込みが多少複雑になり，副表の管理も行うようになった．そのためその本体を実行時ルーチン内に移しマクロはそのサブルーチンを呼び出すように変更した．

６．ＳＮＯＢＯＬ４処理系の実行速度の比較

ＳＮＯＢＯＬ４処理系への日本語テキスト処理機能の追加，字句解析表の変更による処理系の実行効率の変化を調べるために６個のＳＮＯＢＯＬ４プログラムについてその実行時間の比較を行った．ここで比較した処理系は実行回数計数機能追加版ＳＮＯＢＯＬ４（Ｓ４Ｄと略す，原処理系に実行回数計数機能を追加したもの［８］），日本語ＳＮＯＢＯＬ４（Ｓ４Ｊ，ＪＡＰＡＮＥＳＥ），表構造を変更したＳＮＯＢＯＬ４（Ｓ４Ｊ（改良版））の３つの処理系である．コンパイル時間，実行時間の測定には各処理系の統計情報出力機能（ＳＵＭＭＡＲＹ）を用いた．さらにそれぞれの処理系で実行が正しく行われたことを確認する意味で実行回数計数機能を動作させている．なお，これらの測定はＦＡＣＯＭ　Ｍ－２００　ＯＳⅣ／Ｆ４のもとで行った．コンパイル時間，実行時間の測定結果を表１に示す．

ＳＴＩＮＧＹは入力データを編集して出力するプログラムである．このプログラムのＳ４Ｊでの実行時間はＳ４Ｄでの時間に対して１割程度の増加で収まっている．Ｓ４Ｊがすべてのプログラムに対してこの程度の実行時間の増加で実行できるならばよかったのであるが，実は前章で述べたようにそうではなかった．ＬＡＢＥＬはＦＯＲＴＲＡＮプログラムの文番号をつけなおすプログラム，ＰＯＬＹＮＯＭＩＡＬは多項式の加算を記号的に実行するプログラムである．この２つのプログラムではＳＰＡＮ関数，ＢＲＥＡＫ関数を多数使用している．そのためＳ４Ｊでの実行時間はＳ４Ｄでの実行時間と比べて極端に大きくなっている．ＳＰＡＮ関数，ＢＲＥＡＫ関数の実行回数が多いＰＯＬＹＮＯＭＩＡＬプログラムの方がその差は著しい．

ＳＰＡＮ関数等が実行速度を遅くする原因であることを確かめるために作成したプログラムがＳＰＡＮＴＥＳＴプログラムである．この

プログラムはＳＰＡＮ関数等を用いたパターンマッチを繰り返し実行するプログラムである．ＳＰＡＮＴＥＳＴプログラムではＳ４ＤとＳ４Ｊとでの実行時間が３５倍も違う．

Ｓ４Ｊ（改良版）での実行時間はＬＡＢＥＬプログラム，ＰＯＬＹＮＯＭＩＡＬプログラム，ＳＰＡＮＴＥＳＴプログラムもＳＴＩＮＧＹプログラムと同様にＳ４Ｄでの実行時間とほぼ同じである．なお，ＳＴＩＮＧＹ，ＬＡＢＥＬでＳ４Ｄでの実行時間よりもＳ４Ｊ（改良版）での実行時間が短くなっているのは入出力ルーチンをアセンブリ言語で書き直した（４．１節参照）ためであると考えられる．また，すべてのプログラムでコンパイル時間が短くなっているのも同じ理由と考えられる．

日本語テキスト処理機能を用いたプログラムとして日本語の文章中の英字列の相互参照表を作成するＸＲＥＦＡプログラムと漢字列計数プログラム（図２に示したもの）とのコンパイル時間，実行時間を示す．これらのプログラムはＳ４Ｄでは処理できない．漢字列計数プログラムの作成時にはＳＰＡＮ関数等が実行時間に大きく影響することがわかっていたのでその作成に当たってそれらのパターン関数を使用しないようにしている．そのためＳ４ＪとＳ４Ｊ（改良版）との間で実行時間の大きな差は認められない．なお，漢字列計数プログラムではＳＰＡＮ関数を用いることによって２１９６ミリ秒かかっていた実行時間を６１８ミリ秒に短縮できた．

表１．ＳＮＯＢＯＬ４プログラムの各処理系における実行時間の比較

上：コンパイル時間（ミリ秒）
下：実行時間（ミリ秒）

プログラム名＼処理系	Ｓ４Ｄ	Ｓ４Ｊ	Ｓ４Ｊ（改良版）
ＳＴＩＮＧＹ（３９行）	６５ ２７９５	１１１ ３０５９	５４ ２３４９
ＬＡＢＥＬ（１９２行）	２１８ １５４０２	３７４ ２２２６４	１４２ １３０２４
ＰＯＬＹＮＯＭＩＡＬ（１３５行）	１５４ ６６９	２３７ ９７８１	９７ ７０３
ＳＰＡＮＴＥＳＴ（９行）	４４ ４００	４９ １４３４４	２９ ４１６
ＸＲＥＦＡ（２６２行）	―― ――	３９０ １１３４３	１４２ ４０４９
漢字列計数（２５行）	―― ――	７０ ２３９９	３４ ２１９６

７．議論

７．１　処理系の設計と日本語テキスト処理機能の追加

日本語機能の追加ではＳＮＯＢＯＬ４原処理系のＳＩＬで書かれた部分（図１（ａ）に属する）は変更の必要がなかった．ＳＩＬで書かれたＳＮＯＢＯＬ４原処理系は１０年以上も前に設計実現されたにもかかわらず，日本語機能の追加作業をこのように比較的容易に遂行できたのは，ドキュメント［６］がよく整備されていたこと，ＳＩＬシステムの設計が明快でよく機能分割されていたことが大きな理由と考えられる．以下に要点をまとめる．

（１）ＳＩＬシステムの構成　３章にまとめたように階層的に構成されており，機械またはＯＳに独立な部分と依存する部分が明確に分離されている．たとえば４．２節で述べた字句解析を行うための表（その構造は機械，文字コードに依存する．）ではそれを操作するＳＩＬ命令は３個とそれらのマクロの下請けサブルーチンだけが表の構造に依存する．このようにＳＮＯＢＯＬ４処理系の作成に当たってデータ抽象化の概念が取り入れられているため一部の修正がプログラムの広域に影響を及ぼすことがなかった．

（２）ＳＩＬ言語の定義　ＳＩＬ言語の文法に関するドキュメント［６］によりＳＩＬ言語の修得が容易であった．そのため文字の内部表現の変更によるＳＩＬマクロの書き替えもＳＩＬ言語の外部仕様に合わせるだけでよくなかば機械的に行うことができた．

（３）実行時ルーチンの仕様　１つの実行時ルーチンは１つのＳＩＬマクロに対応しており，その外部仕様がＳＩＬマクロ定義やそのドキュメント［６］によりはっきりわかる．さらに実行時ルーチン間の副作用がないため書き替えが必要なルーチンは容易に書き替えられた．

７．２　入出力の設計

ＳＮＯＢＯＬ４原処理系では入出力はＦＯＲＴＲＡＮ実行時ルーチンを用いて行っていたが，今回，可変長ファイルに対して読み書きできるようにするために入出力部を全面的に書き替え，ＦＯＲＴＲＡＮ実行時ルーチンをＳＮＯＢＯＬ４処理系の中から排除した．これはＦＯＲＴＲＡＮ実行時ルーチンの利用に関する情報がほとんどなく，むしろＯＳのデータ管理マクロのマニュアル等の方が完備しているので入出力を行うプログラムを書き易いためである．

このことに関してＳＮＯＢＯＬ４処理系の原著者であるＧｒｉｓｗｏｌｄは［９］の中で「ＳＮＯＢＯＬ４の設計時には移し換えを考慮

してどの計算機にもあるＦＯＲＴＲＡＮ実行時ルーチンをＳＮＯＢＯＬ４の入出力部として採用した．ＦＯＲＴＲＡＮはよく知られた言語であるのでＳＮＯＢＯＬ４を移し換えようとする程度に計算機を知っている人ならばだれでもＦＯＲＴＲＡＮ実行時ルーチンを手軽に使用できると考えたためである．しかし現時点で考えてみると，ＦＯＲＴＲＡＮ実行時ルーチンはだれでも使えるというものではない．むしろ，ＳＮＯＢＯＬ４用の仕様の入出力ルーチンを設計した方がよかったのではないか．」と述べている．今回の日本語機能追加ではこのことがはっきりと現われた．

7．3　日本語ＳＮＯＢＯＬ４の移換性

ＳＮＯＢＯＬ４原処理系は移し換えを考慮して設計されている．日本語テキスト処理機能の追加により移換性が損なわれていないかどうかを調べることが今後の課題である．互換機といわれているＦＡＣＯＭ，ＨＩＴＡＣ，ＩＢＭの計算機ではユーザ側から見たアーキテクチャやＯＳは同じであるがそのうえで作成されたソフトウェアには同じような機能を全く別の仕様で実現しているものもある．日本語処理機能もその一つであり，日本語テキスト保存ファイルの形式，日本語文字コード，日本語文字とＥＢＣＤＩＣ文字を区別するシフトコード等に違いがあるため，日本語処理機能には互換性がない．日本語ＳＮＯＢＯＬ４処理系がこれらのことに容易に対処できることが移し換えの容易さの点で要求されることである．

8．むすび

ＳＩＬで書かれたＳＮＯＢＯＬ４処理系に日本語テキスト処理機能を追加し，さらにその効率改善を行ってきた．ＳＩＬの設計は元来，多くの計算機システムの上でＳＮＯＢＯＬ４処理系を実働させることが第一義であったものと思われる．しかしその設計の故に，機能拡張を容易に行うことができたことを強調しておく．

本研究の一部は文部省科学研究費補助金（一般研究課題番号５７４６０２１１）によった．

参考文献

[1] 武富，高木，川崎他：日本語情報システムＪＥＦの使用法，九州大学大型計算機センター広報，Ｖｏｌ．１３，Ｎｏ．４，ｐｐ．４０６－４６８，１９８０．

[2] L.Cherry: Computer Aids for Writers, ACM SIGPLAN NOTICES, Vol.16, No.6, pp.61-67, 1981.

[3] R.E.Griswold, J.F.Poage, I.P.Polonsky: The SNOBOL4 Programming Language, Prentice-Hall, Inc., 1968.

[4] 木村：ＳＯＦＴＷＡＲＥ　ＴＯＯＬとは何か，情報処理，Ｖｏｌ．２０，Ｎｏ．８，ｐｐ．６７３－６８０，１９７９．

[5] 牛島，黒坂他：ＪＥＦ日本語処理機能を追加したＳＮＯＢＯＬ４について，九大大型計算機センター広報，Ｖｏｌ．１６，Ｎｏ．２，ｐｐ．１５５－１７９，１９８３．

[6] R.E.Griswold: The Macro Implementation of SNOBOL4, W.E.Freeman & Co., 1972.

[7] J.F.Gimpel: Algorithms in SNOBOL4, John Wiley & Sons, 1976.

[8] 吉田，牛島：実行回数計数機能を追加したＳＮＯＢＯＬ４処理系とその移し換えについて，情報処理学会論文誌採録決定，１９８３．

[9] R.E.Griswold: A History of the SNOBOL Programming Languages, ACM SIGPLAN NOTICES, Vol. 13, No.8, pp.275-308, 1978.