リスト処理言語 LIPX のストリング処理について

奥乃 博　　竹内郁雄
(日本電信電話公社　武蔵野電気通信研究所)

I. はじめに

言語の構文解析一特に日本語処理における活用語尾の処理，あるいは，テキスト編集や見易く出力するプリティプリンティングといったワードプロセッシングを行なおうとするとき，強力で豊富な機能を備えたストリング処理は不可欠のものである。こういう立場から，筆者らが開発したリスト処理言語 LIPQ を眺めてみると，LIPQ はストリング処理機能を殆んど備えていないので，ワードプロセッシングには不向きであった。したがって，計算機システムの更新により処理能力が増したのを機会に，LIPQ にストリング処理機能を盛り込んだ新しい言語 LIPX (リペクス) を開発した。

リスト処理という環境の下でストリング処理を実現する場合，その考え方は2つに大別できよう。その1つは，ストリング処理言語 SNOBOL 4 のように行単位で，あるいは，ある記号で括られた文字列をストリングとして捉える方法である。この方法は INTERLISP や LISP 1.9 などが採用している。もう1つは，空白やタブといった区切り記号で区切られた文字列の単位をストリングとして捉え（アトムと同じ），そのようなストリングのリストがテキストを表現していると考える方法である。言い換えれば，区切り記号でテキストを分割し，それをリストにするという前処理を行なった後に，ストリング処理を行なうのである。LIPX のストリング処理はこの考え方を採用している。

本報告では，LIPX の新しい発想に基づくストリング処理について述べ，その応用について報告する。また，LIPX の他の特徴や処理系についても簡単に触れることにする。

本論に入る前に，本報告で使用する言葉の定義を与えておく。同じ文字列を持つものは1つしか作られないので文字列が固定可能なものを「リテラルアトム」と呼び，そのような同定性が保証されていない文字列のことを「ストリング」と呼ぶことにする。リテラルアトムは LISP 1.6 の用語にしたがえば intern されていると言う。

II. リスト処理言語 LIPX の概要

前述したように，LIPX は LIPQ を発展させた上位言語であり，新たに追加した機能としてストリング処理を挙げることができる。LIPQ は PDP 11 ファミリ（主記憶 28 KW）上で実現されているのに対し，LIPX は PDP 11/45（主記憶 64 KW）上で実現されている。PDP 11/45 は他の PDP 11 ファミリと同様に1語が16ビットで構成されているが，メモリマネージメントの機能により 124 KW まで主記憶を拡張することができる。この結果，LIPQ と LIPX の処理系は次の点で異なっている。①システムルーティン（入出力，ガーベッジコレクタ，バルク等）を二次記憶とのオーバレイから主記憶常駐に変更したこと。②ストリング管理領域を二次記憶から主記憶へ移したこと。③自由セル領域が 4 KW 増し 20

KW となったこと．④二進プログラム領域が８KW 増し最大１２KW 利用できるようになったこと．LIPX の処理系のメモリマップを図１に示す。

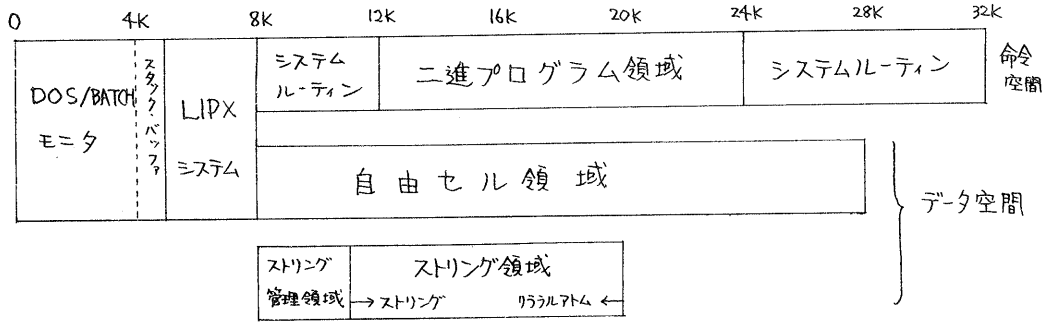

図１．LIPX システムの処理系のメモリマップ

PDP 11/45 のメモリマネージメントは，命令空間とデータ空間の二つを区別する機能がある*ので，LIPX システムは両者を区別して使用している。自由セル領域とストリング管理領域は同じデータ空間に割当てられているので，両者を同時に参照することはできない，両者のいずれを参照するかは，ページアサインメントレジスタを書き換えることによって決定する．

LIPQ および LIPX 上でのテストプログラムの実行時間を下に示す。

	箱入り娘	TPU（9題の合計，含GC）
LIPQ	111 秒	37.3 秒
LIPX	146 秒	59.0 秒

LIPX は上述した①～④の特徴を持っているので，LIPQ よりも実行速度は速いように思えるが，実際には逆の結果が得られた。その原因としては次のようなものが考えられる．LIPQ の処理系はバイポーラメモリ（アクセス時間 300 ns）しか使用していないのに対して，LIPX の処理系は 32KW のバイポーラメモリの他に 32KW のコアメモリ（アクセス時間 850 ns）を使用している。また，メモリマネージメントを使用すると１メモリアクセスに付き 90 ns 余分にかかる．主にコアメモリの使用が，①～④の効果を帳消しにしてこのような結果を招いたと言えよう。

Ⅲ．LIPX のストリング処理

1．ストリング処理の考え方

リスト処理の下でストリングというデータ型を考えると，INTERLISP ではストリングを示す記号「"」で括られる任意の文字列をストリングとして取扱っており，他の LISP システムの多くも INTERLISP に倣っている。この方法では，空白，タブ，復帰改行といったりテラルアトムの区切り記号をストリング中に含まれることになる。例えば，"(A B C)" は長さが７のストリングである。この方法は次

*）PDP 11/40，60 のメモリマネージメントにはこの機能が備っていない。

の2つの疑問を投げかける。第1の疑問点は、この方法はストリング処理の主な応用であるワードプロセッシングや言語処理とはうまく整合しないのではないかという点である。つまり、テキスト編集やプリティプリンティングといったワードプロセッシングにおいては、入力ストリングを読込んだ後に行なう最初の仕事は空白やタブを探し出して除くということであり、また、自然言語処理においては、入力文を分ち書きすることは前提条件として素直に受け入れられている。一方、処理しようとするストリングが空白やタブといった区切り記号を含んでいるのはほとんどの場合余り能が無いように思われる。第2の疑問点は、第1のものと同じ事であるが、リテラルアトムは区切り記号で区切られるのに、ストリングは区切り記号で区切られないというのでは考え方に首尾一貫性が欠如しているように思える点である。

LIPXでは、これらの点を考慮して、ストリングもリテラルアトムと同様に区切り記号で区切られるというように定めた。言い換えると、LIPXの入力ルーティンはINTERLISPのように文字列を字面通りに読込んだ後に、ストリング用区切り記号にしたがって読込んだ文字列を区切るという前処理を施すのである。このような前処理を施すと、区切り記号を含まずに長さが255を越えるような文字列はほとんど出現しないと予想されるので、ストリングの長さを255に制限するようにした。

2. ストリングの入出力

一般のストリング入力ルーティンは、読込んだ文字列をストリング管理領域に登録し、そのポインタを値として返す。一方、LIPXの入力関数READは、S式の読込みの際、「"」が現われると、その時点からストリング読込みモードとなり、ストリング用区切り記号（空白，タブ，あるいはユーザの指定した記号）で区切られた文字列を一つのストリングとして読込む。つまり、「"」が現われなければリテラルアトムとなるべきものがストリングになるわけである。この状態は次の「"」が現われるまで続く。ただし，数と解釈される文字列が区切られたときには、この状態においてもストリングとはならず、通常の数とする。下に例を示す。

入力文字列	READが返す値
"ABC"	"ABC"
("ABC = 123 * X")	("ABC" "=" 123 "*" "X")
((GIVEN "X = 3") (SOLVE "ABOVE"))	((GIVEN "X" "=" 3) (SOLVE "ABOVE"))
"ABC DE"	"ABC" ；余分なものがついているというエラーメッセージがでる.
"ABC%_DE"	"ABC%_DE" ；%はエスケープシンボル

ストリングは関数READによって入力することができるが、テキストのような一続きのストリングのリストを入力する場合に、入力文字列の先頭に「("」を，末尾に「")」をつけるのは面倒である。関数SREADは入力文字列の先頭に「("」が，末尾に「")」がついているものと見做すREADの変種なので、こういう場合に使用するとよい。また，関数SREADLは入力文字列の先頭に「("」が，改行の次に「")」がついていると見做すREADの変種である。ただし，この2つの関数は，入力文字列中に現われる「"」を入力時のモード変換記号ではなく，エスケープシンボル「%」のついた「%"」と見做す点がREADと異なっている。

ストリング用区切り記号には，空白，タブ，復帰，改行というそれ自身何の意味も持たない区切り記号の他に，それ自身が1文字のストリングとなる区切り記号がある。前者はシステムによって定まっており，新たに追加したり削除したりすることができないが，後者はユーザが自由に指定することができる。したがって，応用分野の特徴によってうまく区切り記号を定めれば，入力関数がそれらに基づいて前処理を施すので，プログラムを簡潔に書くことができる。

ストリング用区切り記号の追加と削除は次の関数で行なう。

(SDELIM ストリング)　　；ストリング中の各文字をストリング用区切り記号として追加する。

(SREMDELIM ストリング{NIL}*)　　；ストリング中の各文字をストリング用区切り記号から削除する。NILのときは全ての記号を削除する(初期化)。

例で説明しよう。(SREMDELIM) の実行後に入力文字列 ("A=B*30+10") をREADで読込むと ("A=B*30+10") が返される。一方，(SDELIM "=+*") の実行後に同じ入力文字列をREADで読込むと ("A" "=" "B" "*" 30 "+" 10) が返される。

ストリングの出力関数はWRITEとPRINCである。ストリングを出力する際に，WRITEは「"」の対で文字列を括って出力するが，PRINCは「"」の対およびエスケープシンボル「%」を出力しない。また，WRITEは個々のストリングを「"」で括って出力する方法の他に，ストリングおよび数が連続するときに「"␣"」を「␣」で置換えて出力する方法がある。次に出力例を示す。

```
(PRINC "LIPX") --> LIPX
(PRINC '("I" "%" "SEE" "."))  --> I SEE.
(WRITE '("I" "%" "SEE" "."))  --> ("I" "%" "SEE" ".")  または ("I %  SEE .")
(WRITE '("Y=" ("X" "*" 123 "+" "Z"))). --> ("Y=" ("X" "*" 123 "+" "Z"))
                                        または ("Y=" ("X * 123 + Z"))
```

WRITEの2つの出力方法は「"」の考え方―文字列を括ってストリングを表現する記号と入力時のモード変換記号―に対応している。

3. ストリングの内部表現

ストリングの内部表現はストリングヘッダと文字列とで構成されている。文字列はストリング領域にべた詰めに入れられている。ストリング領域に空場所が無くなったときにはストリングガーベッジコレクタが起動されてくずを回収する。ストリングヘッダは二元セルで構成されており，文字列へのポインタと文字数を持っている(図2)。ストリングの番地はストリングヘッダである二元セルの番地

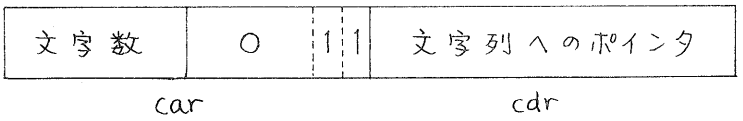

図2. ストリングヘッダ

である。ストリングヘッダの指標はcarの下1バイトが3であることである。なお，carの下1バイトはbignumや浮動小数点を将来実現できるように考慮されている。多くの関数では，実際の文字列のコピーは行なわないで，ストリングヘッダの操作だけを行なう。また，ストリングの値は自分自身であり，ストリングが評価されると(EVAL)，そのストリングが返される。

*) {} で括られた値は引数がdefaultのときの値を示している。

4. ストリング処理関数

関数の引数としてストリングとリテラルアトムの両方が許されるとき，ストリングと書くことにする。この場合，リテラルアトムは自動的にストリングに変換される。ストリング処理関数を次に掲げる。

述語 (STRINGP s式) ；S式がストリングのとき T，それ以外 NIL.
(SNULL s式) ；S式が空ストリングのとき T，それ以外 NIL.
(SEQUAL ストリング ストリング) ；両者の文字列が等しいとき T，それ以外 NIL. e.g. (SEQUAL 'AB "AB")=T
(SLESSP ストリング ストリング) ；前者が辞書式順序で後者より小さいとき T，それ以外NIL. e.g. (SLESSP "AB" "ABC")=T
(SGREATERP ストリング ストリング) ；前者が辞書式順序で後者より大きいとき T，それ以外 NIL.

コピー (SCOPY ストリング) ；ストリングヘッダのコピーを作る.

連接 (SCONC 不定個のストリング) ；引数をその順で連接したストリングを作る. e.g. (SCONC "A" 'BC "XY")="ABCXY"

切出し (SGET ストリング 数n{1}) ；nが正のときは先頭から，負のときは末尾から |n| 文字のストリングを取出して返す。
(SPOP ストリング 数n{1}) ；返す値はSGETと同じ。副作用として第1引数のストリングヘッダは残りのストリングを指すように変えられる。

接頭辞 (SPRE ストリング$_1$ ストリング$_2$ 書き換えフラッグ{NIL}) ；ストリング$_2$ がストリング$_1$ の接頭辞であれば，残りのストリングを返す。また，書き換えフラッグが T のときは ストリング$_1$ のストリングヘッダを返す値と同じものに書き換える。ストリング$_2$ がストリング$_1$ の接頭辞でないときは NIL を返す。 e.g. (SPRE "UNLUCKY" "UN")="LUCKY"

接尾辞 (SPOST ストリング$_1$ ストリング$_2$ 書き換えフラッグ{NIL}) ；SPREの場合とは接頭辞が接尾辞となる点だけが異なる。
e.g. (SPOST "CONFUSING" "ING")="CONFUS"

アトム化 (MKATOM ストリング) ；ストリングと同じ文字列のリテラルアトムを（LISP 1.6 の intern と同じ）返す。
e.g. (MKATOM "ABC")=ABC，ただし，(MKATOM "")=NIL.

ストリング化 (MKSTRING リテラルアトムか数) ；リテラルアトムと同じ文字列の，あるいは，数の十進表現と同じ文字列のストリングを返す。e.g. (MKSTRING 'AB)="AB"，ただし，(MKSTRING NIL)="NIL".

LIPX では，空ストリング("")とNILが対応しており，それぞれストリングとリストの特別なもの（単位元）と見做している。

5. 強力パターンマッチング関数

SNOBOL4の特徴の1つは強力なパターンマッチング機能であるが，LIPX もSNOBOL4並みの豊富で強力なパターンマッチング機能を備えている。パターンマッチング機能は SMATCH という関数で実現されている。

(SMATCH ストリング パターン 作業情報{NIL})

パターン = (パターン要素 ...)
パターン要素 = &|数|ストリング|(ストリング ...)|(ストリング. WHILE)|(ストリング. UNTIL)
作業情報 = NIL|T|(作業情報要素 ...)
作業情報要素 = NIL|T|数|ストリング

作業情報がNILか作業情報要素が全部NILのときは述語となり，マッチが成功すれば T を，失敗すれば NIL を返す。作業情報要素はパターン要素に対応している。作業情報要素がNILあるいはパターン要素に対応するものがないときはNOPであり，Tはマッチしたストリングを，0はマッチしたストリングの先頭からの位置を返すことを示す。正数はマッチすべきストリングの先頭からの位置[POS]*を示し，負数は末尾からの位置[RPOS]を示す。ストリングはマッチしたストリン

*) 以下，SNOBOL4の対応する術語を角括弧に括って示す。

グと置換すべきストリングを示す。

パターンはパターン要素の連接［concatenation］で構成される。&は任意のストリングにマッチする［ARB］。数は指定された長さのストリングにマッチする［LEN］。（ストリング'ooo）は択一［alternation］を示し，括弧内のいずれか1つのストリングにマッチする。（ストリング'WHILE）はストリング中にある文字が続く最長のストリングにマッチする［SPAN］。（ストリング'UNTIL）はストリング中にない文字が続く最長のストリングにマッチする［BREAK］。マッチングはパターンがストリング全体とマッチしたときに成功となり，部分ストリングとのマッチングは行なわない［anchored mode の scanning］。SMATCH が述語でなく，かつ，マッチングが成功した場合には，SMATCH は各作業情報要素に基づいて返される値をリストにして返す。ただし，置換が行なわれたときには書き換えられたストリングを返す。作業情報が T のときは，すべての作業情報要素が T であることを示す。例をあげよう。

```
(SMATCH "AAABBBCC" '("AA A B BBCC")) = T   ; 連接の例，SMATCH は述語
(SMATCH "ACABC" '(("AC" .WHILE) &) T) = ("ACA BC")   ; SPAN の例
(SMATCH "ABCD" '(& "BC" &) '(NIL "XYZ")) = "AXYZD"   ; 置換の例
(SMATCH "ABCD" '(& "" "BC" &) '(NIL "XYZ")) = "AXYZBCD"   ; 挿入の例
(SMATCH "ABCD" '(& "BC" &) '(NIL 3)) = NIL   ; POS(3)の例．3が2だと Tとなる
(SMATCH "X*(Y+3323)/38=3083" '(& ("( [") & (") ]") &) T)   ; 択一の例
                = ("X* ( Y+3323 ) /38=3083")
(SMATCH "CYBORG007" '(("%01" .UNTIL) & &) '(T 0 T)) = ("CYBORG 7 %007")
(SMATCH "UNFORTUNATELY" '("UN" 3 & "NATE" &) '(NIL T T T -2))
                = ("FOR TU NATE LY")
(SMATCH "MISSISSIPPI" '("1 I 2 I 2 I 2 I") '(T NIL T NIL T NIL T))
                = ("M SS SS PP")
(SMATCH "AA ... A" '(("A AA") ... ("A AA"))) = T   ; 実行時間は39分12秒
          50個              ("A AA")が 25個
```

SMATCH は SNOBOL 4 のパターンマッチング機能のうち LIPX にとって必要なものだけを組込んだために書けないパターンもあるが，そのようなものは他の関数と組合せることによって実現できるので本質的な問題ではない。SMATCH で書けないパターンには再帰的なパターンや同じ文字が2つ続くというパターンなどがある。

Ⅳ. ストリング処理の応用

1. Tiny BASIC / Palo Alto to BASIC/RT11 プログラムコンバータ

このコンバータは，マイクロプロセッサ上で使用されている Palo Alto 版の Tiny BASICで書かれたプログラムを，PDP11 の OS，RT11上の BASIC 用に変換する。この変換はプログラムの意味に立ち入らずに行なうので，変換されたプログラムが正しく実行できるかは保証できない。例えば，Tiny BASIC では変数名を入力するとその値が入力変数に代入されるが，この機能は BASIC/RT11 では許されない。

Tiny BASIC の特徴と BASIC/RT11 の相違について簡単にまとめておく。

①コマンドキーワードと関数名は省略形が使用できる。例えば，「P.」，「PR.」，「PRI.」，「PRIN.」はすべて「PRINT」と同じである。最も省略した形を次に掲げる。

```
A.=ABS    F.=FOR    G.=GOTO    GOS.=GOSUB    IF=IF    IN.=INPUT
L.=LIST   N.=NEW    P.=PRINT   REM=REMARK    R.=RETUR  R.=RND
R.=RUN    S.=SIZE   S.=STEP    S.=STOP       TO=TO     IMPLIED=LET
```

これは Tiny BASICでだけ有効。また，文の区切り記号は各々「;」と「\」である。

② Tiny BASIC では論理式は0か1を返す式であり，また，IF文中の述部では0以外の値を真とし，「THEN」を書く必要はない。例えば，V=(A>B)*X+(A<B)*Y や IF X Y=8, Z=9 … などが許される。BASIC/RT11 ではIF文中の述部は論理式でなければならず，かつ，論理式はそこでしか使用できない。また，述部の次には GOTO 文だけが許される。③ Tiny BASIC では INPUT文中にストリング（「"」が「'」で括られた文字列）があると，それが入力待ち記号として出力される。④ Tiny BASIC では変数は1文字であり，配列名は「@」である。また，RNDは両者で異なる。

コンバータは1パスで Tiny BASIC 用プログラムを BASIC/RT11 用に変換する。Tiny BASIC 用のプログラムを附図1に，変換されたプログラムを附図2に掲げる。変換処理は4つの部分—入力処理，行番号の処理，両BASIC間の変換，出力—に分かれている。入力は1行毎に行なわれ，下に示す出力単位に変換されて出力へ

出力単位 ＝ NIL | (行番号 . 出力単位)

【(ストリング…)　出力フラグ　ストリング長 . 出力単位】

渡される。出力フラグはTのとき出力できることを示し，NILのとき出力できないことを示す。NILのときは後参照の行番号が未だ同定されていないことを示している。行番号の処理は可能な限り入力された行番号を採用するようにしているので，直接管理すべき行番号は入力行番号と異なるものと後参照されていて未だ同定されていないものだけである。後参照されている行番号は，出力単位中のストリングが入るべきセルを使用してポインタで継ぐことによって管理している。出力は出力単位中のストリング長を参考にして，1行70字で改行を行なう。

両BASIC間の変換処理では，Tiny BASIC 用プログラムを構文解析して得られる構文木を基にして BASIC/RT11用に変換している。構文解析ルーティンは，Tiny BASIC の文法をBNF記法で表現したときの超変数に対応するルーティンから構成されている。構文解析は入力行がうまく前処理されていると極めて都合がよい。入力処理とは大げさであるが，コンバータは次のようにストリング用区切り記号

(SDELIM "( ) , ; %" @ <=> ＃ * / ＋ − '")

を宣言することによって，SREADL に前処理を依頼する。

SDELIM と宣言してある区切り記号は SEQUAL で調べることができる。この他のキーワードの切分けは，例えば，変数Lがストリングのリストを指しており，FORについて行なおうとすると，(SMATCH (CAR L) '(("FOR FO. F.") &) T) で行なわれることになる。L＝("FO.I") とすると ("FO." "I") が返される。

BASIC/RT11 への変換は，「;」を「\」へ，配列名「@」を「A1」(使用されていない変数名ならば何でもよい)へ，「RND」を「FNR」へ，という字面だけの置換えで済むものばかりではない。少し意味まで立ち入って変換しなければならない文のうち，主なものについて変換の方針を示しておく。

① 論理式が算術式中に現われたとき，作業用の配列W1を用いて論理式の値を予め配列に代入しておき，算術式の計算を行なう（附図1の20行と附図2の20~26行）。

② IF文中の述部に算術式が現われたときは，0と比較する論理式で置換える（附図1の25行と附図2の27~29行）。THEN部がGOTO文以外のときは，述部の真偽判定を逆にして，偽のときに次の行番号へGOTOするように変更する。このとき，真のときに実行すべき文がその行に入り切らない場合の手当もしておく（附図2の50~51行）。

コンバータのプログラムは500行であり，その一部を附図3に示す。Tiny BASIC 用の130行のプログラムを LIPX インタプリータで1分48秒，コンパイルドコードで12秒の時間で，BASIC/RT11用の180行のプログラムに変換する。

コンバータで一番厄介であったのは、INPUT文とPRINT文中のストリング*であった。通常の場合は、空白やタブといった区切り記号は区切り記号以上の意味を持たないので、その出現個数は全く無視される。それに対して、ストリング中では空白やタブは意味を持っているので、その出現個数も無視できないが、LIPXではこの取扱いが不可能であり(現在の処)、すべて1つの空白で置換えるように処理した。また、ストリング中にストリング用区切り記号が現われると、1文字の文字列として区切られるので、"A␣(␣B", "A␣(B", "A(␣B", "A(B", の区別がつかなくなるという問題がある。これはすべて "A(B" という一続きのものであると解釈した。これらの点以外は説明してきたように、入力時にストリング用区切り記号に基づいて入力文字列の切分けを行なうので、構文解析ルーティンは極めて書き易いものとなった。

2. LIPX(LIPQ)のプリティプリンタ

LIPXのプリティプリンタはWRITEに組込まれている機能であり、アセンブリ語で書かれている。ここではLIPXのストリング処理の応用として、LIPXのプリティプリンティング機能を取り上げる。LIPXのプリティプリンタは、1行分のバッファを介したコンシューマジェネレータというコルーティンの形で実現されており、括弧にだけ注目するアルゴリズムを採用している。このアルゴリズムは、構文木を構成するときに、1行で出力できるような部分木が見つかるとその部分木の構文解析は棚上げにするという方法である。それに対して、ここでは、フォーマッティング情報を持った構文木からプリティプリンティングを行なう汎用プリティプリンタを考える。(ただし、構文規則、フォーマッティング規則および入力データを与えると、構文解析を行なって得られる構文木からプリティプリンティングを行なうという汎用プリティプリンタは、実行速度と作業領域の点で実用的でない。)

構文解析ルーティンは、前節と同様に構文をBNF記法で表現したときの各超変数に対応するルーティンから構成されている。ストリング用区切り記号として、LIPXのリテラルアトム用の区切り記号そのものを使用する。構文木は四元セルで構成する(下図)。フォーマッティング情報は対応するルーティン名であり、

構文木 ═ NIL | [(区切り記号 | (ストリング…) | 構文木) 文字数 フォーマッティング情報 . 構文木]

NILのときはそれ以上分解できないことを示す。例を下に示す。

```
A --> [("A") 1 NIL]

'?ABC --> [("'" "?" "ABC") 5 NIL]

(A B)
 --> [[("(") 1 OPENPR !("A") 1 NIL !("B") 1 NIL !(")") 1 CLOSEPR] 5 DIVIDABLE]

(CAR . CDR)
 --> [[("(") 1 OPENPR !("CAR") 3 NIL !(".") 1 DOT !("CDR") 3 NIL
       !(")") 1 CLOSEPR ]
       10 DIVIDABLE ]

((#1:A) B 123 (CD #1))
 --> [[("(") 1 OPENPR
       ![("(") 1 OPENPR !("#1:" "A") 4 NIL !(")") 1 CLOSEPR] 6 DIVIDABLE
       !("B") 1 NIL !("%123") 3 NIL
       ![("(") 1 OPENPR !("CD") 2 NIL !("#1") 2 NIL !(")") 1 CLOSEPR] 6 DIVIDABLE
       !(")") 1 CLOSEPR ]
       21 DIVIDABLE ]
```

*) 「"」か「'」で括られる文字列のことをこの節ではストリングと呼ぶ。

この応用においても前節と同様に「〃」で括られた文字列中にストリング用区切り記号が現われる場合を除いて，構文解析ルーティンは極めて書き易いものであった。フォーマッティングルーティンはリスト操作であり，ストリング処理は構文解析時にのみ必要である。

V. おわりに

ストリング処理をリスト処理の下で行おうときには，空白，タブといったストリング用区切り記号でストリングを区切り，テキストをストリングのリストとして取扱うべきであるという考え方を提案し，その発想に基づいて実現されたLIPXのストリング処理について説明をしてきた。この発想は，言い換えると，ユーザレベルでのテキストの区切り記号による切分けを，入力段階で行なおうというものである。この入力段階での区切り記号による切分けという前処理の機能によって，構文解析やテキスト編集といったワードプロセッシングは極めて容易にプログラム化ができるようになったと考えられる。また，本報告では触れなかったが，LIPX上で実現されつつある日本語処理システムにおいても，SMATCHによってプログラムの構造がすっきりしたという事を聞いている。

本報告で説明した2つの応用におけるLIPXのストリング処理の使い勝手は完璧ではないもののまずまず満足のできるものであった。ただ，関数SREADとSREADLにおいて，空白やタブが意味を持つ箇所，例えば、「〃」で括られた文字列中では，文字列がそのまま読込める機能を追加する必要性を痛感した。この機能は，構文解析を行なう上では不可欠であるが，他の応用においても必要であるほど汎用な機能であるという点には疑問が残る。

筆者らが本報告で提案したストリング処理が世に普及することを願って本報告を終える。

謝辞

日頃御指導をいただく池野室長，若菜調査役，御討論いただいた池野特別研究室の方々に感謝いたします。また，本報告をまとめる機会を与えていただいた高平第一研究室室長に感謝いたします。

参考文献

(1) 奥乃・竹内：リスト処理言語LIPQとその処理系，通研実報，vol.26, no.10 (1977).
(2) 奥乃・竹内：LIPQからの話題，情処学会，第18回大会予稿集，211，1977.
(3) W. Teitelman：INTERLISP Reference Manual，1974，Xerox，Bolt Beranek and Newman.
(4) Griswold et al：The SNOBOL4 Programming Language，1971，Prentice-Hall.
(5) LISP User's Manual（LISP1.9），1977，電総研.
(6) Quam and Diffie：Stanford LISP 1.6 Manual，SAILON，No.26.6，1972.
(7) L.C. Wang：Palo Alto Tiny BASIC，Dr. Dobb's J. of Computer Calisthenics & Orthodontia，May，1976.
(8) RT11 System Reference Card，1976，DEC.
(9) BASIC programming manual — Single-User，Paper Tape Software，1971，DEC.
(10) 鶴田・武市・和田：Syntax-Directed Pretty-Printer，第17回プログラミングシンポジウム報告集，1976.

```
  5 Y=2999;IN.,"DO YOU WANT A DIFFICULT GAME?  ",,"(Y OR N)"A
 10 P.,"STARDATE 3200;  YOUR MISSION IS ",;IF A=YY=999
 15 K=0,B=0,D=30;F.,I=0 TO 63;J=R.(99)<5,B=B+J
 20 M=R.(Y),M=(M<209)+(M<99)+(M<49)+(M<24)+(M<9)+(M<2),K=K+M
 25 @(I)=-100*M+10*J-R.(8);N.,I;IF (B<2)+(K<4)G.15
 30 P.,"TO DESTROY ",#1,K,," KLINGONS IN 30 STARDATES."
 35 P.,"THERE ARE ",#1,B,," STARBASES.";GOS.160;C=0,H=K
 40 U=R.(8),V=R.(8),X=R.(8),Y=R.(8)
 45 F.,I=71 TO 152;@(I)=0;N.,I;@(B*X+Y+62)=4,M=ABS(@(B*J+V-9)),N=M/100
 50 I=1;IF NF.,J=1 TO N;GOS.165;@(J+134)=300,@(J+140)=S,@(J+146)=T;N.,J
 55 GOS.175;M=M-100*N,I=2;IF M/10GOS.165
 60 M=M-M/10*10,I=3;IF MF.,J=1 TO M;GOS.165;N.,J
 65 GOS.145;GOS.325;IF KG.95
```

附図1.　Tiny BASIC で書かれたプログラム

```
  1 DIM W1(20),A1(200)
  2 DEF FNR(X)=X*RND(X)
  5 Y=2999\PRINT "DO YOU WANT A DIFFICULT GAME?","(Y OR N)";\INPUT A
 10 PRINT "STARDATE 3200; YOUR MISSION IS",\IF A<>Y GOTO 15\Y=999
 15 K=0\B=0\D=30\FOR I=0 TO 63\W1(1)=1\IF FNR(99)<5 GOTO 16\W1(1)=0
 16 J=W1(1)\B=B+J
 20 M=FNR(Y)\W1(1)=1\IF M<209 GOTO 21\W1(1)=0
 21 W1(2)=1\IF M<99 GOTO 22\W1(2)=0
 22 W1(3)=1\IF M<49 GOTO 23\W1(3)=0
 23 W1(4)=1\IF M<24 GOTO 24\W1(4)=0
 24 W1(5)=1\IF M<9 GOTO 25\W1(5)=0
 25 W1(6)=1\IF M<2 GOTO 26\W1(6)=0
 26 M=W1(1)+W1(2)+W1(3)+W1(4)+W1(5)+W1(6)\K=K+M
 27 A1(I)=-100*M+10*J-FNR(8)\NEXT I\W1(1)=1\IF B<2 GOTO 28\W1(1)=0
 28 W1(2)=1\IF K<4 GOTO 29\W1(2)=0
 29 W1(3)=W1(1)+W1(2)\IF W1(3)>0 GOTO 15
 30 PRINT "TO DESTROY",K;"KLINGONS IN 30 STARDATES."
 35 PRINT "THERE ARE",B;"STARBASES."\GOSUB 160\C=0\H=K
 40 U=FNR(8)\V=FNR(8)\X=FNR(8)\Y=FNR(8)
 45 FOR I=71 TO 152\A1(I)=0\NEXT I\A1(8*X+Y+62)=4\M=ABS(A1(8*U+V-9))\N=M/100
 50 I=1\IF N=0 GOTO 55\FOR J=1 TO N\GOSUB 165\A1(J+134)=300\A1(J+140)=S
 51 A1(J+146)=T\NEXT J
 55 GOSUB 175\M=M-100*N\I=2\W1(1)=M/10\IF W1(1)=0 GOTO 60\GOSUB 165
 60 M=M-M/10*10\I=3\IF M=0 GOTO 65\FOR J=1 TO M\GOSUB 165\NEXT J
 65 GOSUB 145\GOSUB 325\IF K>0 GOTO 95
```

附図2.　BASIC/RT11 用に変換されたプログラム


```
(DE FACTOR (L)
 (PROG (X Y Z)
       (COND ((UNSIGNED=INTEGER L))
             ((SMATCH (CAR L) '(("R, RN, RND")))
              (COND
               ((SEQUAL (CADR L) "(")
                (SETQ X (EXPRESSION (CDDR L)))
                (SETQ L (CDR X))
                (CONS (NCONC (LIST "FNR(") (FLATTEN X) (LIST (POP L))) L) )))
             ((SETQ Z (FUNCTION=IDENTIFIER L))
              (SETQ L Z)
              (COND
               ((SEQUAL (CADR L) "(")
                (SETQ X (EXPRESSION (CDDR L)))
                (SETQ L (CDR X))
                (CONS (NCONC (LIST (CAR Z) "(") (FLATTEN X) (LIST (POP L))) L)
               )))
             ((VARIABLE L))
             ((SEQUAL (CAR L) "(")
              (SETQ X (EXPRESSION (CDR L)))
              (SETQ L (CDR X))
              (CONS (COND
                     ((BCELL= X) (NCONC (LIST "(") (FLATTEN X) (LIST (POP L))))
                     (T (POP L) (FLATTEN X)) )
               L )))))
```

附図3.　Tiny BASIC to BASIC/RT11 プログラムコンバータの1部分