九州大学学術情報リポジトリ
Kyushu University Institutional Repository

FORTRANを基礎にしたプログラム言語について(III)

牛島, 和夫
九州大学工学部通信工学科

https://doi.org/10.15017/1467973

出版情報：九州大学大型計算機センター広報. 4 (1), pp.3-14, 1971-02-25. 九州大学大型計算機センター
バージョン：
権利関係：

KYUSHU UNIVERSITY

FORTRANを基礎にしたプログラム言語について(Ⅲ)

牛島　和夫※

9. リストとリストの相互関係

前回から引きつづいてＤＹＳＴＡＬの紹介を行なう。以下で前々回、前回の引用を行なう場合には、Ⅰ－3節、Ⅱ－5節のようにして示すことにする。

ＤＹＳＴＡＬでリストとリストの相互関係を記述するには2通りの方法がある。1つは、Mode＝1（リストの名）を要素とするリストによって、そのリストより下位にあるリストのテーブルをつくる方法。この下位のリストがさらにMode＝1であれば、さらに下位のリストを持ち、このようにして、多層構造（木構造）を表現することができる。もう一つの方法は、Ⅱ－5節で述べた、リストのNode部を利用して次々と結節リストとして結合してゆく方法である。カッコでNode部を表現することにするとLISTA→LISTB→LISTC　の連結は

LISTA(LISTB)—
LISTB(LISTC)—
LISTC(空)

として、リストを鎖状につないでゆくことができる。

ここでは前者の方法の簡単な応用として、行列の処理を考えてみよう。m×nの行列Aの表現はＤＹＳＴＡＬではⅡ－6節に述べたような方法が一般であるが、次のような方法も考えられる。行列Aをm要素をもつn個の列ベクトル$a_1, a_2, \cdots, a_n$に分解し、それぞれのベクトルの要素をMode＝3，Nmax＝Nctr＝mなるn個のリストに格納し、これらのリストの名をMode＝1，Nmax＝nのリストＮＡＭＥに順に格納しておく。このようにすると、A(I, J) 要素は

```
X=FITEM(I,ITEM(J, NAME))
```

によってとり出すことができる。A(I, J)要素へ値Xを格納するには

```
IDUMMY=IPLACE(X, ITEM(J, NAME)+I)
```

となる。また、Ⅰ列とＪ列のいれかえの操作は、記憶装置上のⅠ列のベクトルに対応するリストとＪ列のそれとを動かすことなく、ＮＡＭＥリスト上のⅠ番目の要素とＪ番目の要素をいれかえてやりさえすればよい。すなわち

```
K=ITEM(I, NAME)
IDUMMY=IPLACE(ITEM(J, NAME), NAME+I)
IDUMMY=IPLACE(K, NAME+J)
```

※　九州大学工学部通信工学科

行列AからI列を削除し、新しいベクトルを、新しい第n列として追加するには、次のようにする。

IDUMMY＝IDLETE(I, NAME)

IDUMMY＝LOAD(LSTALL(3, m, NOM), NAME)

10. リストの鎖とその処理

前節に述べた第2の方法によりリストをNode部を利用して鎖状につないだ場合の処理手続きについて述べる。Ⅰ－4節鎖構造をもったデータの処理で解説したように、鎖構造は、入口と出口の異なるqueue, 入口と出口が同じstackが代表的なものである。鎖状になった構造からのデータのとりだし方としては、前者に対しては、**ＦＩＦＯ方式**（先入れ先出し方式）、後者に対しては、**ＬＩＦＯ方式**（後入れ先出し方式）であった。さらに、鎖のはしだけではなく、鎖の内部にあるデータにまで立ち至る場合には、**挿入**、**削除**、**入れ換え**などの処理が、基本的であった。

先ず、鎖構造をstackと見たてた場合についてしらべてみよう。

鎖の端しを示すための特別のリスト（anchor listと呼ぶ）ＬＣＨＡＩＮをあらかじめ用意する。

IDUMMY＝LSTALL(2, 1, LCHAIN)

鎖状につながっているLIST3, LIST2, LIST1（図1(a)）に

LIST4(　　)― 407, 410, 428

をpush down すると 図1(b) のようにＬＣＨＡＩＮとＬＩＳＴ４のNode部が変化する。また図1(b) の状態の鎖から一番上のリストをpop upすると、ＬＩＳＴ４が得られ、図1(a)のような鎖が得られる。push downは結局

LOT(LIST4－3)＝LOT(LCHAIN－3)

LOT(LCHAIN－3)＝LIST4

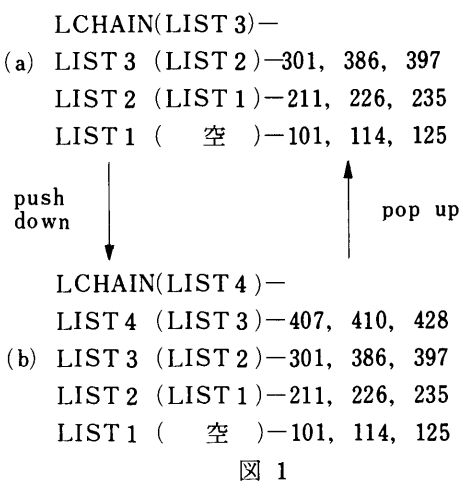


図 1

という操作が行なわれているが、これをこれまでの例にならって関数にする。

```
FUNCTION LPUSH(LISTA, LCHAIN)
```

LCHAINはanchor listである。関数値はLISTA。anchor list にまだ何も連結されていなければLOT(LCHAIN−3)＝0となっている。

pop upの操作としては

```
LPULL=LOT(LCHAIN−3)
IF(LPULL) 20, 20, 10
10 LOT(LCHAIN−3)=LOT(LPULL−3)
20 CONTINUE
```

となるので、これも関数としてまとめておく

```
FUNCTION LPULL(LCHAIN)
```

関数値はpop up されたリスト名である。

ここに2つの鎖、LCHAINとLSTOREがあり、LCHAINからpop upしたリストをLSTOREにpush downする操作は

```
IDUMMY=LPUSH(LPULL(LCHAIN), LSTORE)
```

とすればよい。図1で LIST3とLIST2の間にLISTAを挿入するには

```
IDUMMY=LPUSH(LISTA, LIST3)
```

だけでよい。また 同じ図1で

```
LPOPUP=LPULL(LIST3)
```

とすれば、LIST2が削除されて LIST3(LIST1)−…となる。LPOPUPにはpop upされたリストの名LIST2が得られる。

queueとしての取り扱い　I−4節に述べたようにqueueの場合は、queueの先頭と後尾を示す情報が必要となる。実はLOT(LCHAIN＋1)をqueueの後尾の情報（最新に鎖につながったリストの名）を格納するために使用する。Node部は、queueの先頭のリストの名が格納されることになる。このqueueに新しいリストLISTAが到着すると

```
LAST=LOT(LCHAIN＋1)
LOT(LAST−3)=LISTA
LOT(LCHAIN＋1)=LISTA
LINK=LISTA
```

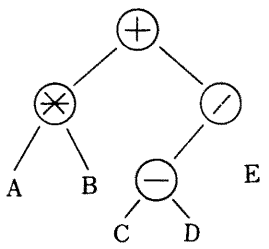
という操作が行われればよい。これも関数としておく。

FUNCTION LINK(LISTA, LCHAIN)

関数値はLISTAである。このqueueの先頭からリストをとり出すのは、stackの場合と同じようにLPULLを用いればよい。

11. 木構造の処理

Ⅰ－4節の図6にあげた算術式A＊B＋(C－D)/EをDYSTALのリストを用いてあらわしてみると、例えば図2のようにすることができる。この例のように必ず枝分れが2つになっている木を2進の木と呼んでいるが、木構造の枝分れは、一般にはいくつであってもよい。木構造に対するもっとも基本的な処理は、木構造に属する、すべての要素をたどることである。DYSTALシステムでは、これを可能にするために2つの関数を用意している。



Ⅰ－4　図6

FUNCTION LOADX(INDEX, MAX, LISTA)

FUNCTION NEXTX(INDEX, NFLAG)

これらの関数の機能を例題によって説明しよう。図3のような木構造が与えられている。

```
LIST1[+]－LIST2, LIST3
    LIST2[＊]－LISTA, LISTB
      LISTA－'A'
      LISTB－'B'
    LIST3[/]－LIST4, LISTE
      LIST4[－]－LISTC, LISTD
            LISTC－'C'
            LISTD－'D'
      LISTE－'E'
```

図2．算術式A＊B＋(C－D)/Eの木構造のDYSTALによる表現の一方法
ここで[　]はIden部のラベルを示す。

```
LISTA
    LIST1
      L－1－COW, RAT
      L－2－TIGR, LION, FOX
      L－3－DOG, CAT, MICE
      L－4－HEN
    LIST2
      L－5－101, 102, 103
      L－6－ 4.6, 81.31, 53.41, 16.34, 76.54
```

図3．木構造の例

ここでLISTA, LIST1、LIST2はMode＝1, L－1, L－2, L－3, L－4はMode＝4, L－5 L－6はそれぞれMode＝2およびMode＝3である。この木構造に属するリストをとにかく図4に示すように①, ②, ……, ⑪の順でたどろうというのである。ここで9つのリストのうちLISTA, LIST1,

LIST 2 は、木の節を示すリスト(Mode=1)でこれをname listと呼ぶことにする。またL－1，…，L－6の木の端（葉といってもよい）にあたるリストをdata listと呼ぶことにする。また、木の根にあたるリストの名をこの木構造の名とする。木の根からの近さによって、木の根の層から、level 0，level 1 ……のように名付ける。さて、図4のリストのたどり方をみると

下降（levelが深くなる）……図4で①，②，⑧

横行（同じlevelを進む）……図4で③，④，⑤，⑦，⑨

上昇（levelが浅くなる）……図4で⑥，⑩，⑪

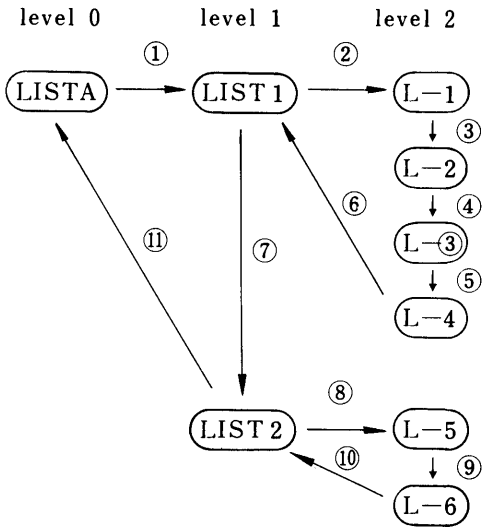


図4．

の動きがあることがわかる。さらに、動いた結果行きあたったリストがname listの場合とdata listの場合にわけられる。そこで、これらをまとめて、次のようなflagで示すことにする。

flag＝1　得られたリストはname listで、次の動きは下降である（①，⑦の結果）。

flag＝2　得られたリストはdata listである（②，③，④，⑤，⑧，⑨）。

flag＝3　上昇によって得られたname list、これは以前に出会ったリストである（⑥，⑩）。

flag＝4　木の根のリストにもどったことを示す。これによって、木構造をたどる仕事の終りを示す（⑪）。

木構造上でtraceの結果得られたリストからtraceをして次のリストを得る関数がNEXTXである。第2引数のNFLAGが上述したflagの値を返し、また関数値は、そのtraceの結果得られたリストの名である。たとえば、①の結果現在LIST 1 にあって

IGET=NEXTX(INDEX, NFLAG)

とすると②によって、NFLAG＝2、IGET＝L－1 が返される。また第1引数のINDEXは、現在どのlevelのどのリストの何番目の要素をtraceしているかを示す指標の役割をするリストでLOADX によって全てのtraceに先立って用意される。

FUNCTION LOADX(INDEX, MAX, LISTA)

LISTAは木の根、MAXは、木構造の深さより1以上大きい値（図4では2＋1以上）。LOADX を実行すると temporary 領域（Ⅱ－4節参照）に指標用のリスト INDEXとその結節リストNAME ができあがる。関数値としてINDEXが返される。NAMEはLOT(INDEX－3)によって得られる。

Iden	Node	Mode	Nmax	Nctr	1	2	3	4	5
INDX	NAME	2	5	0	0	0	0	0	0
NAME	1	1	5	1	LISTA				

図5　LOADXによって作られる指標用の作業リスト

例えば

IDUMMY＝LOADX(INDEX, 5, LISTA)

によって、図5のようにできあがる。なお、NAMEのNode部の値は、次にNEXTXが呼ばれたときのtraceの方向を示す指示子で

1……次は下降

2……次は横行

3……次は上昇

を示している。図4の例で、例えばLIST1から動く方向は②と⑦の2通りあるわけで、LOT(NAME－3)の内容が1ならば②、2ならば⑦となるわけである。図5は初期状態であるから1がセットされている。図3(図4)の木構造をNEXTXによってたどった結果の指標リストINDEX、NAMEの内容およびそのとき返されたNEXTXの関数値、NFLAGの値を図6にあげておく。

	Iden	Node	Nctr	1	2	3	NEXTXの関数値	NFLAGの値
初期状態	INDX	NAME	0	0	0	0		
	NAME	1	1	LISTA				
①	INDX	NAME	1	1	0	0	LIST1	1
	NAME	1	2	LISTA	LIST1			
②	INDX	NAME	2	1	1	0	L－1	2
	NAME	2	3	LISTA	LIST1	L－1		
③	INDX	NAME	2	1	2	0	L－2	2
	NAME	2	3	LISTA	LIST1	L－2		
④	INDX	NAME	2	1	3	0	L－3	2
	NAME	2	3	LISTA	LIST1	L－3		
⑤	INDX	NAME	2	1	4	0	L－4	2
	NAME	3	3	LISTA	LIST1	L－4		
⑥	INDX	NAME	1	1	0	0	LIST1	3
	NAME	2	2	LISTA	LIST1	0		
⑦	INDX	NAME	1	2	0	0	LIST2	1
	NAME	1	2	LISTA	LIST2			
⑧	INDX	NAME	2	2	1	0	L－5	2
	NAME	2	3	LISTA	LIST2	L－5		
⑨	INDX	NAME	2	2		0	L－6	2
	NAME	3	3	LISTA	LIST2	L－6		
⑩	INDX	NAME	1	2	0	0	LIST2	3
	NAME	3	2	LISTA	LIST2			
⑪	INDX	NAME	0	0	0	0	LISTA	4
	NAME	3	1	LISTA				

図6　指標用リストの状態変化

```
      COMMON LOT(5000)
      CALL INLOT(10,5000)
      LIST = LSREAD(NOM)
      CALL POLISH(LIST)
      CALL RPOLSH(LIST)
      STOP
      END


      SUBROUTINE POLISH(LIST)
      COMMON LOT(5000)
      IDUMMY = LOAD(LOT(LIST-4),LSTALL(4,100,LOUT))
      IDUMMY = LOADX(INDEX,5,LIST)
    3 IGET = NEXTX(INDEX,NFLAG)
      GO TO (1,2,3,4),NFLAG
    1 IDUMMY = LOAD(LOT(IGET-4),LOUT)
      GO TO 3
    2 IDUMMY = LOAD(LOT(IGET+1),LOUT)
      GO TO 3
    4 CALL KDUMP
      IDUMMY=LERASE(INDEX)
      RETURN
      END


      SUBROUTINE RPOLSH(LIST)
      COMMON LOT(5000)
      IDUMMY = LSTALL(4,100,LOUT)
      IDUMMY = LOADX(INDEX,5,LIST)
    1 IGET = NEXTX(INDEX,NFLAG)
      GO TO (1,2,3,4), NFLAG
    2 IDUMMY = LOAD(LOT(IGET+1),LOUT)
      GO TO 1
    3 IDUMMY = LOAD(LOT(IGET-4),LOUT)
      GO TO 1
    4 IDUMMY = LOAD(LOT(IGET-4),LOUT)
      CALL KDUMP
      IDUMMY=LERASE(INDEX)
      RETURN
      END
```

図7

なお、NEXTXによって木構造をtraceして得られるリストに対する処理の方法はいろいろありうる。図2の算術式の木構造に対して2通りの処理を行なったプログラムを図7に示す。ここでLSREADは図8に示す算術式の木構造のデータの読みこみ。LISTは木の根を示すリストの名である。サブルチンPOLISHによって得られる出力は、

＋　＊　A　B　/　－　C　D　E

サブルチンRPOLSHによって得られる出力は

A　B　＊　C　D　－　E　/　＋

である。結果の一部を図9に示す。

	Iden	Node	Mode	Nmax	Nctr
8	11	28	42	56	70
	NAME		1	10	0
1	+		1	2	2
	2	5			
2	*		1	2	2
	3	4			
3	LSTA		4	1	1
	A				
4	LSTB		4	1	1
	B				
5	/		1	2	2
	6	9			
6	-		1	2	2
	7	8			
7	LSTC		4	1	1
	C				
8	LSTD		4	1	1
	D				
9	LSTE		4	1	1
	E				
	STOP				

```
DUMP OF LIST   4 NAME  79 IDEN =   + NODE =    0 MODE =    1 NMAX =    2 NCTR =    2
                86          105
DUMP OF LIST   5 NAME  86 IDEN =   * NODE =    0 MODE =    1 NMAX =    2 NCTR =    2
                93          99
DUMP OF LIST   6 NAME  93 IDEN = LSTA NODE =    0 MODE =    4 NMAX =    1 NCTR =    1
                A
DUMP OF LIST   7 NAME  99 IDEN = LSTB NODE =    0 MODE =    4 NMAX =    1 NCTR =    1
                B
DUMP OF LIST   8 NAME 105 IDEN =   / NODE =    0 MODE =    1 NMAX =    2 NCTR =    2
                112         131
DUMP OF LIST   9 NAME 112 IDEN =   - NODE =    0 MODE =    1 NMAX =    2 NCTR =    2
                119         125
DUMP OF LIST  10 NAME 119 IDEN = LSTC NODE =    0 MODE =    4 NMAX =    1 NCTR =    1
                C
DUMP OF LIST  11 NAME 125 IDEN = LSTD NODE =    0 MODE =    4 NMAX =    1 NCTR =    1
                D
DUMP OF LIST  12 NAME 131 IDEN = LSTE NODE =    0 MODE =    4 NMAX =    1 NCTR =    1
                E
DUMP OF LIST  13 NAME 137 IDEN =      NODE =    0 MODE =    4 NMAX =  100 NCTR =    9
                +         *         A         B         /
                -         C         D         E
DUMP OF LIST  14 NAME 242 IDEN =      NODE =    0 MODE =    4 NMAX =  100 NCTR =    9
                A         B         *         C         D
                -         E         /         +
```

算術式の木表現

POLISHの結果

RPOLSHの結果

図9

12. 文字処理

FACOM230−60は1語に4文字を格納することができる。ある変数に文字データを与えるには

```
IWD= 4HLION
```

とすればよい。ところが、

```
IWD= 3HDOG
```

のような用法は、FORTRANでは許されていない。さらに4文字を超える文字列を単精度の変数に与えることも不可能である。4文字以下の文字列の場合は、前述した関数ISTOREを用いて

```
IDUMMY=ISTORE(3HDOG, IWD)
```

とすればよい。ところが、文字列は、任意の長さをとりうるので、新しい関数を用意する。

```
FUNCTION ISHIFT(N, IWORDS, LISTA)
```

IWORDSは配列名、LISTAはリストの名。配列IWORDS(1)〜IWORDS(N)に格納されている文字列をLISTAに移しかえる。すなわち

LOT(LISTA+1)=IWORDS(1), LOT(LISTA+2)=IWORDS(2), ····

関数値はLISTA。ここで

```
DIMENSION IWDS(10)
```

とすると、次のようにしてISHIFTを用いることができる。

```
IDUMMY=ISHIFT(10, IWDS, LISTA)
IDUMMY=ISHIFT(5, IWDS(6), LISTA)
```

ここでは、LOT(LISTA＋1)=IWDS(6), LOT(LISTA＋2)=IWDS(7), ….となる。

```
IDUMMY=ISHIFT(7, 26HTHERE LIVED A PRETTY GIRL., LISTA)
```

ここでは、LOT(LISTA＋1)＝4 HTHER, LOT(LISTA＋2)＝4 HE　LI, ….となる。

さてこのようにして、リストに格納された文字列を1文字毎に弁別するためには、それを1文字毎に分解する関数が必要である。

```
FUNCTION NPACK(N, IWORDS, LISTA)
```

がそのために用意されている。配列IWORDSに入っているN文字の文字列をLISTAの次に使える場所(LOT(LISTA＋Nctr＋1))から1文字ずつに分解して格納する。この結果LOT(LISTA)=Nctr＋Nとなる。関数値はLISTAである。いま、LOT(LISTA)＝0とすると

```
IDUMMY=NPACK(5, 5HAIUEO, LISTA)
```

によって、LOT(LISTA＋1)＝4 HA　　, LOT(LISTA＋2)＝4 HI　　, …そしてLOT(LISTA)＝5となる。リストLTEXTに入ったMode 5のtextを1文字毎に分解してLISTBに格納するには、

```
IDUMMY=NPACK(LOT(LTEXT)＊4,LOT(LTEXT＋1), LISTB)
```

とすればよい。次にこのようにして分解された文字列を再びもとの文字列に縮める関数として

```
FUNCTION IPACK(N, LISTA, LISTB)
```

がある。関数値はLISTBである。

LISTA－H, Y, P, O, T, H, E, S, I, S

のとき

```
IDUMMY=IPACK(10, LISTA, LISTB)
```

により

LISTB－HYPO, THES, IS

となる。

次に、与えられた文字列が一つの文章であると考えよう。この中からある単語を探すという問題はどう処理すればよいであろうか。Ⅱ－7節に述べた関数LOCATEは、リスト中の1要素を探すのには有効であるが、単語といえば必ずしも1要素とはかぎらない一つながりの情報である。

```
FUNCTION MASK(LISTM, LISTA, I)
```

が用意される。LISTMにある一連の情報をLISTAの第I番目の要素からはじめて、合致するものを

探す。関数値は、合致したものがあれば、LISTA中の合致した最後の要素の番地である。

例えばLOT(LISTM)＝4のとき、LOT(LISTA＋3)～LOT(LISTA＋6)の情報と合致すれば、関数値は6となる。合致するものがなければ0を返す。

```
IDUMMY=NPACK(17, 17HTHE IDES OF MARCH, LISTA)
IDUMMY=NPACK(2, 2HES, LISTM)
```

のとき

```
I=MASK(LISTM, LISTA, 1)
```

によってIに8が与えられる。また、LISTAの中にLISTMを全て探して、その場所をLISTBに得るプログラムは次のように書けばよい。

```
    KSTART=0
10  KSTART=MASK(LISTM, LISTA, KSTART+1)
    IF(KSTART) 30, 30, 20
20  IDUMMY=LOAD(KSTART, LISTB)
    GO TO 10
30  CONTINUE
```

探した単語を別の単語でおきかえる操作、たとえば

LISTA—T, H, E, , B, O, Y, , R, U, N, S

LISTR—G, I, R, L

として、LISTA中にB, O, Y(LISTMに入っている)を見付けたらそれをG, I, R, Lでおきかえたいといった目的のために

```
FUNCTION ISWAP(LISTR, N, LISTA, I)
```

LISTRの要素をLISTAのI番目からI＋N－1番目までの要素とおきかえる。関数値はLISTA。

上の例では

```
IDUMMY=ISWAP(LISTR, 3, LISTA, 5)
```

ここでLOT(LISTA)の内容は12から13にかわることに注意されたい。LISTA中にLISTMの内容が入っていたら、それをそっくりLISTRと入れかえるプログラムは

```
    LOC=MASK(LISTM, LISTA, 1)
    IF(LOC)30, 30, 20
20  IDUMMY=ISWAP(LISTR, LOT(LISTM), LISTA, LOC-LOT(LISTM)+1)
30  CONTINUE
```

と書けばよいであろう。

以上でDYSTALの紹介を終る。以下に2、3の注意をあげる。

Ⅰ－7節に注意したようにIDUMMYは、関数を、サブルチンのように使いたいときに用いた。するとこのIDUMMYという変数は、決して算術式の右辺や、WRITE文の出力リストとして用いられることがない。FACOM230－60FORTRANでは、このような使い方をしてもコンパイラから警告を受けることはないようであるが、NEAC2200－500のFORTRAN－LやHITAC5020E HARPでは、右辺または、WRITE文で使われていない変数に値を代入しているというような警告がコンパイラから出される。しかし、これはもっとも軽傷のエラーとみなして実行に入ってくれる。

次にⅠ－6節の表1にあげたように、JIS FORTRANでは、副プログラムのパラメタとして、関数の場合には文字定数を用いないようにしている。ところが、ここで紹介した関数、NPACK、ISHIFTなどみな関数を引用する際に、文字定数を実パラメタとしている。FACOM230－60 FORTRANはじめ現存するFORTRANプロセッサは、関数の実パラメタに文字定数を使ってもさしつかえないが、JISの規定にのっとって、場合によっては、文字定数の使えないプロセッサもありうるので注意を要する。次回はSLIPの紹介を行う予定である。

参考文献

DYSTAL manual(広報vol. 3, No. 6 参照)