C 6. 行列の基本演算の数式処理とリスト処理言語

石黒美佐子，中村康弘，稲見泰生，斉藤直之（原子力研究所）

§1. まえがき

リスト処理言語の比較は，〔1〕でもなされている．我々は，行列の演算をFORTRANで書く時に，めんどうくさいDO LOOPを作る手間を省いて，これを自動的に行なえないかということが動機となつて，行列の数式処理を試みた．それと同時に，リスト処理言語の使い易さを比較検討するために，現在7044で使用可能なリスト処理言語，すなわち，SLIP, IPL-V, LISP1.5を使用して，同じ題材に取り組むことにした．§3で処理のあらましをSLIPで行なつたのを例にとつて述べることにするが，これと関連づけて，各々の言語の特徴，リスト構造，使い易さ等について述べてみたい．なおLISPは，原研で作成した（浅井，稲見による）ものを利用した．

§2. 行列の基本演算の数式処理

ここでは我々は，次の4つの行列の基本演算と，その組み合せを許すことにする．

1）スカラー積　　S＊A
2）加法　　A＋B
3）乗法　　A＊B
4）ベキ　　A^3
5）転置　　A^T

インプットとしては，FORTRANのプログラムの中で，たとえば$(A^2+B)(D-S\cdot C)+A^T-B\cdot D$を計算したいときは，次のような呼び出し方をする．

```
CALL MATRIX(25,25H(A**2+B)*(D-S*C)+A**T-B*D)
```

アウトプットとしては，式を整理したものと，行列として，FORTRANとの結合を考えて，答となる行列の(I,J)エレメントを与える．すなわち

```
A*A*D-S*A*A*C-S*B*D+A**T
```

及び

```
SUM(K,A(I,K)*SUM(L,A(K,L)*D(L,J)))-S*SUM(K,A(I,K)*
SUM(L,A(K,L)*D(L,J)))-S*SUM(K,B(I,K)*D(K,J))+A(J,I)
```

後者については，FORTRANで演算しやすいようにカード化することも考えている．

§3. SLIPによる処理過程

3—0 リストの内部表現

内部表現は，A.Lapidus〔Ⅲ〕に準じるものとし，必要に応じて付加を行なう．すなわち演算記号は保存されない．タームの始めの常数部の符号により，そのタームの正負が示されて，ターム内でのかけ算記号は省略される．

変数は1語6文字のうち頭の2文字は変数の種類を表わすものとして利用される．従つて変数は最大4文字しか意味をなさない．

種　類	内部表現	例
scalar	bb××××	S，SCIA（Sで始まる）
matrix	b0××××	A，ABIA（Sで始まらぬ）
転置matrix	0b××××	A^T

（ここでbはブランク，××××はFORTRANの変数の考え方に準じる．）

常数は，次のように定める．

ベキ　　正の整数

その他の常数　ノーマライズされた浮動小数点表示

転置行列は，A^T をA**Tとインプットし，内部では上記のように表現しなおされる．

行列のベキは，A^3 をA**3とインプットされ，内部ではA*A*Aの形で保存される．

タームは，常数と変数がかけ算で結ばれたもので，A^2，$2*B^T$ 等である．

expressionは，タームを加減したもので A^2-2*B^T 等である．これを今迄に述べた内部表現によれば，次の5語のようになる．

```
1.0
b0bbbA
b0bbbA
−2.0
0bbbbB
```

この内部表現は，記憶容量が節約されること，及び，常数と変数の見分けが簡単な点ですぐれている．

3—1 インプットphase

この段階では，FORTRANのコーディングシーケンスをスキャンして，常数と変数を作り出して，リスト1を作ることを目的としている．スキャンには，R.W.Floyd〔Ⅱ〕のフローチャートを修正して利用した．その結果，リスト1の構成要素としては，次の種類のものが残る．

(
)
+
−
*　{ *
　　**
常数 { 整数
　　浮動小数点表示
変数 { scalar
　　matrix
　　T

§2の例では，リスト1の構成要素は，次のとおりである．

(,A,*,*,2,+,B,),*,(,D,−,S,*,C,),+,A,*,*,T,−,B,*,D

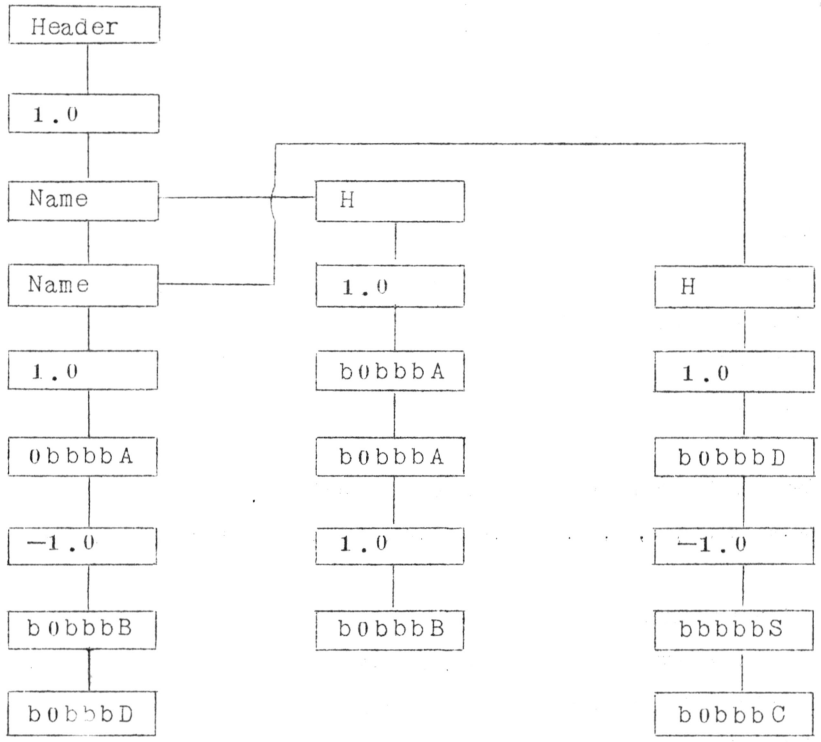
3−2　リスト化shase

この段階では，カッコの処理，すなわち，カッコで包まれたexpression をサブリスト化する．一方行列の転置と，ベキの内部表現を作り，A. Lapidus〔Ⅲ〕に準じた内部表現を完成し，これをリスト2とする．変換には，Table 1を使用した．ここでコントロール語のsignは常数の符号，termはタームの始まりの時は0，始めでなければ1であり，expはカッコの用い方のチェックのためのものである．結果として，リスト2は，次の構成要素を持つ．

Header	(2)
Name	(1)
常数	(0)
変数	(0)

( )内はSLIPのIDコードである．

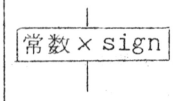
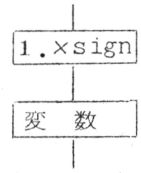
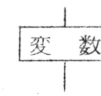
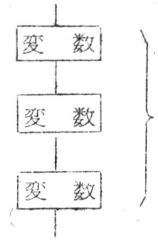
前の例では，リスト2は次のようになる．



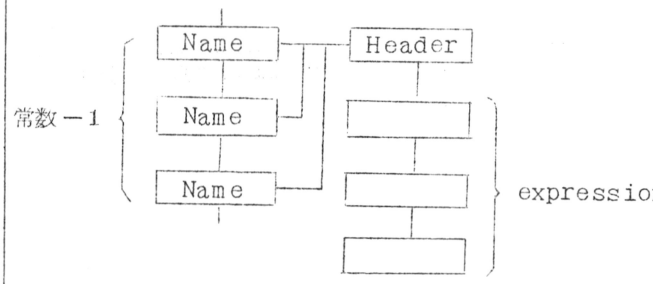
カッコ毎にサブリストを作るこの方法は，行列の転置，ベキの取り扱いに対して，SLIPのリカーシヴな機能を利用出来るので大変好都合である．

Table 1

list 1 element	control words	list 2
Initial setting	sign＝1 term＝0 exp ＝0	Header
(	exp=exp+1	Sub list を作成する． term＝0 の時 1.×sign Name — Header term＝1 の時 Name — Header

	term=0 sign=1	
)	sign=1 term=1 exp=exp−1 if exp<0, error	Sub listの終りの処理
常　数	. term=1	term=0の時  term=1の時 termのはじめの常数とかけあわせて常数部を変更する.
変　数	term=1	term=0の時  term=1の時 
＋	sign=1 term=0	
−	sign=−1 term=0	
＊		かけ算の場合
＊,＊		ベキの場合 1.　（変数）＊＊（常数） 

2. （expression）＊＊（常数）



3. （変数）＊＊T
変数が行列の時は，転置の内部表現になおす．

4. （expression）＊＊T
expressionの転置処理をする．

3－3 数式処理phase

ここでは，式のカッコがはずされて，式をタームの加減算の形に表わすことを目的としている．タームの構成は，常数が前に出て，scalarがアルファベット順に整理されて次に続き，行列が後にくる．このphaseもSLIPのリカーシヴ機能に助けられるので，たいした困難はない．こうして出来たものをリスト3と名づける．このリストは次のphaseの処理を簡単にするために，メインリストには常数だけ保存され，それ以外のタームの要素はサブリスト化されている．

3－4 編集phase

ここでは，同類項を整理することを目的としている．前のphaseで，常数以外のタームの要素がサブリスト化されているので，SLIPのLSTEQU（リストが等しいかどうかチェックするルーチン）を利用して，同じタームがあれば，その常数部を加える（或いは減じる）ことにより同類項を一つにまとめる．

3－5 アウトプットのphase

§2で示したアウトプットを得るためのphaseである．しかしながらアウトプットフォームは考えていたより見づらいものになり，FORTRANとの結合も有効なやり方がなかなかない．目的に応じて，変更を加えると，満足したアウトプットが得られるのではないかと思う．今後考慮する余地がある．

§4. リスト処理言語の特徴，使い易さ，リスト構造の比較

SLIP，IPL－V，LISPを分けて，それぞれの立場での比較を次にする．リスト構造については，互に比較するためにまとめて載せる．

4－1　SLIP

SLIPは，FORTRANの中で自由に使える点で，非常に使い易い．特に，FORTRANから数式処理ルーチンを呼び込む場合，そのつなぎをFORTRANによつて，実行出来るのは好都合である．FORMACのように，数式処理が数値計算と共存して実行出来る利点もある．しかしFORMACに比べると，数式処理過程を初めから自分で作らなければなら点で初歩のuserには使い難い．しかし，有効な数式処理ルーチンを，たくさん含んだパッケージを作り上げれば，FORMAC以上に多くのプロセスが可能で，ひとえに数式処理ルーチンを作るuserの努力に負うところが大きい．しかしLISPと比べると，リスト処理過程の大部分をuserが知つていなければならない点でめんどうくさい．他のことにもあてはまるが，言語が使い易くて，プログラミングが簡単であればある程その処理は非能率なものとなる．LISPでは，stringsの内部表現は一定で，使うメモリも馬鹿にならないが，SLIPやIPL－Vのように，自分でリストの使い方を決められる場合は，その場その場に応じて，有効な使い方が可能になり，プロセスの効率も悪くない．又LISPとちがつて，他の言語，たとえばassembly言語のサブルーチンを一緒に使用したり，我々FORTRANによる数値計算に慣れたuserには便利な点が多い．その上IPL－Vのように，言語がassembly言語に近くて難解であるという欠点もない．言語の取りつき易さと，流動的に使いこなせるという点で，まさるものといえよう．行列の基本演算の数式処理について，SLIPを使用して，数式処理の初心者が試みて，計算機がそんなに思うように使用出来ない場合で，1人で1ケ月半の労力であつたことをつけ加える．

4－2　IPL－V

IPL－Vによる数式処理の方法は，基本的にはSLIPによる場合と同じである．すなわち，数式処理過程は5 phasesからなり，各phaseの処理目的はSLIPによる場合と同様である．しかし，IPL－VとSLIPのリスト処理言語上の相違から，各phaseの処理手続は細かい点で異なる．特にリスト構造の内部表現は大部違う．IPL－Vだけで数式処理プログラム全体を書くことは面倒で，不可能に近い．ことに，インプットされた数式をR. W. Floydの方法によつて処理するとき，IPL－Vだけではstring manipulationがやりにくい．そこで最初のphaseは，ほとんどassembler言語で書いた．その他のphasesでも，IPL－Vプログラムのほかに，assembly言語によるprimitive routinesをいくつか付加することによつて，IPL－Vでの処理を容易にした．その中

には，論理演算等がある．IPL—Vのリスト処理ルーチンはいろいろあつてそれで大部分処理できた．ただ，リスト構造のリストセルを1つづつ取り出す時，前向きにはJ60ルーチンによつて可能だが，後向きの場合は，それに相当するものがないので不便である．リカージョンの方法はSLIPに似ている．プログラムの書き易さ，見易さ，Debuggingの容易さ等の面からみるとIPL—Vはassembler的で劣つていると思う．

4—3　LISP

LISP1.5は，カッコでくくられたstringsでなければ取り扱いが出来ないので，SLIP，IPL—Vのインプット形式では，SLIP言語では処理出来ない．あえて処理しようとすると，LISP言語の本質からかけ離れてしまうので，インプット形式を今までのものを例にとれば，次の形のものとなり，LISPで述べたはじめの2つのphases(3—1，3—2)を省略した．

(((((A,**,2),+,B),*,(D,—,(S,*,C))),+,(A,**,T)),—,(B,*,D))

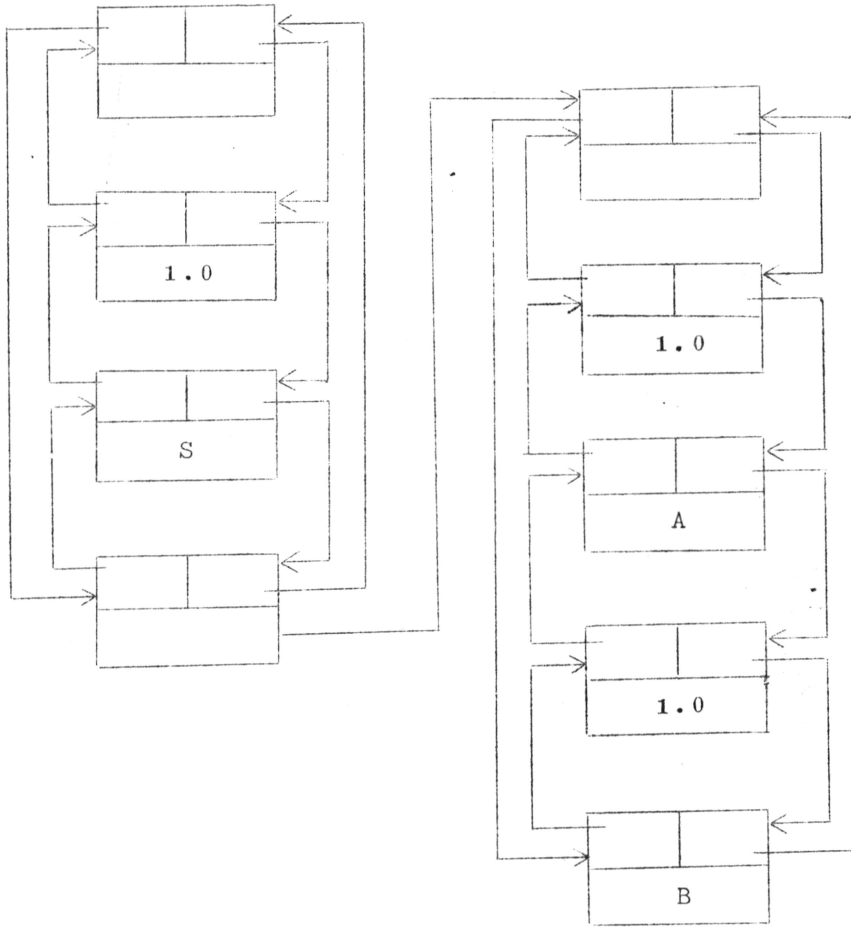
内部表現は3—0で述べた考え方に従つた．プログラミングは簡単で，phase 3—3と3—5はLISPの処理ルーチンにまかせてよい．3—4はLISP言語によるプログラミングの時に考慮すればよい．問題なのは，LISPの最小単位がATOMであり，カッコのついたstrings(Sで始まればscalar，‥‥)でなければ取り扱えないので特別なサブルーチンを作成し，チェック出来るようにした．SLIP，IPL—Vと比較してプログラミングの簡単さはあるが，その為に柔軟性がない上に，メモリをたくさん使う．この点については，後に例を示す．一般的に言われていることであるが，LISPでもリカージョンが簡単に出来るので，プログラミングが簡単になる．将来の問題としては，数式処理と数値計算が同時に行なえるようになるために，LISP言語を拡張して数値計算が簡単に出来るようにしたい．

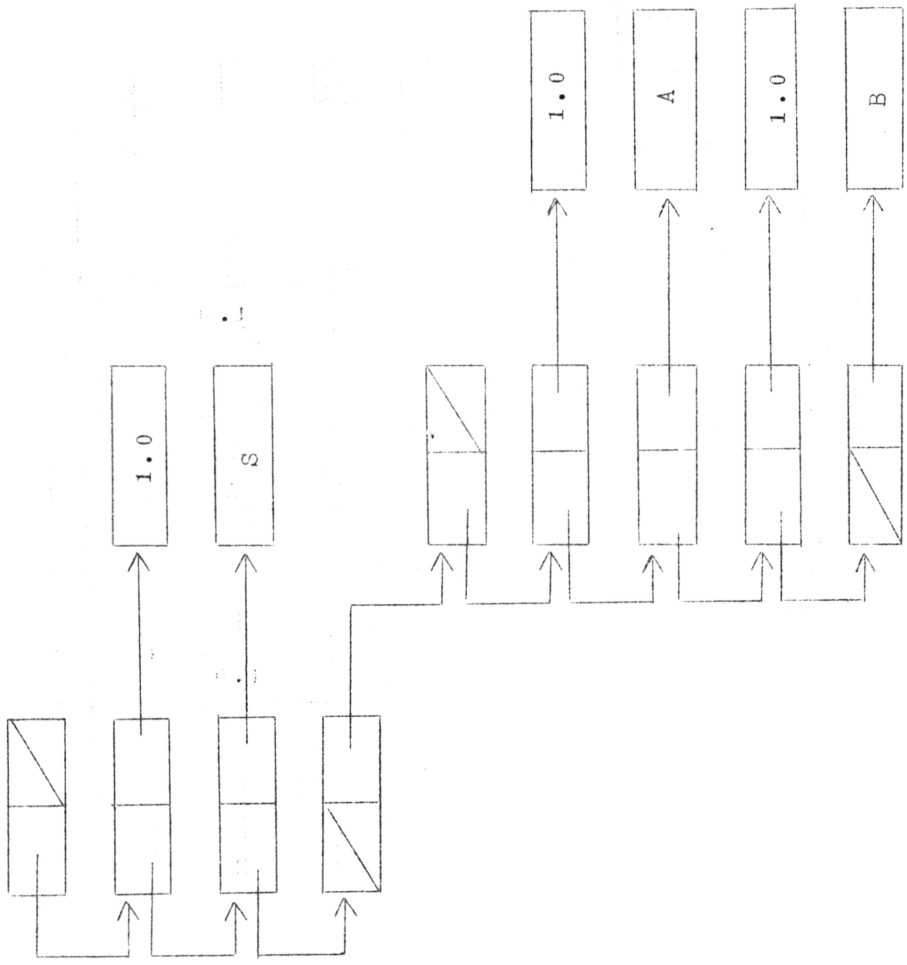
4—4　リスト構造の比較

リスト構造の違いを比較するために，式S*(A+B)を例にとつて，前述の3つの言語に対するリスト構造をそれぞれ載せる．必要とする語数は

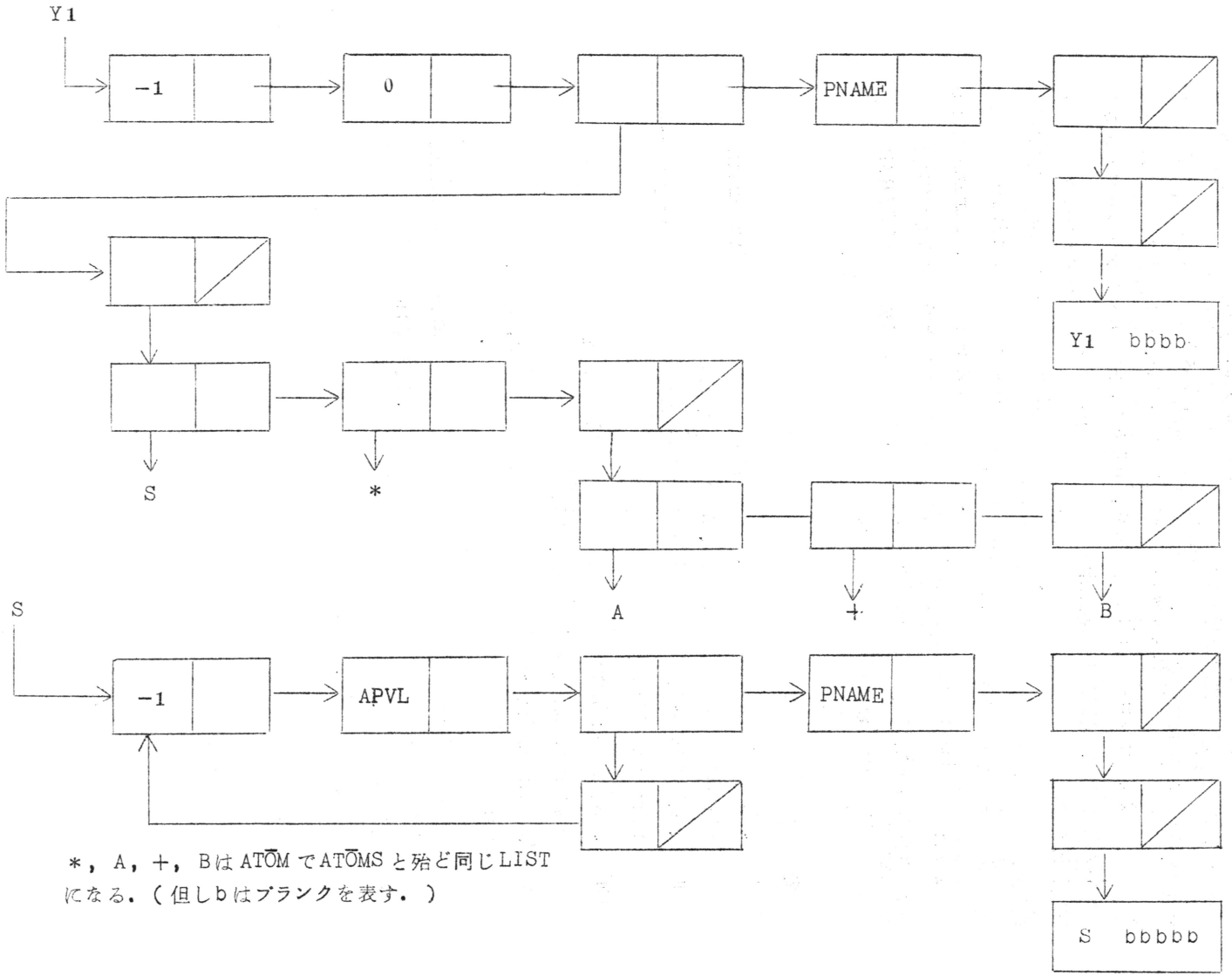
SLIP	18語
IPL—V	15語
LISP1.5	54語

SLIP





IPL-V



*，A，＋，BはATOMでATOMSと殆ど同じLISTになる．（但しbはブランクを表す．）

§5. まとめ

ここで，これから数式処理を行なつていくために，どんなリスト処理言語が望ましいかを考えてみたい．柔軟性の点では，FORTRANのサブルーチン形式（実際は，function形式のものが多いが）になつているSLIPが勝るが，リスト処理の詳細まで自分で考えるのは，手間がかかるので誰もが使いこなせるわけではない．従つて基本的なものを，たとえば，代数式のカッコをはずすこと，微分，積分とか，式の展開等サブルーチン化して，組み入れておく．又LISPの基本関数もSLIPで実行出来るように組み込み，LISPの働きを持たせることが考えられる．一方，リスト処理機能を多様化するために，今SLIPでは，2語で1 itemが構成されるが，それをmultiple-way multiple words itemの形式に拡張することも考えられる．結局最小限必要な機能は，string manipulationが可能なこと，リスト処理が可能なこと及び数値計算と共存して実行出来ることの3点が挙げられる．さらにもつと拡張されて，数値計算に於けるFORTRAN に対応する数式処理言語の規格品が作られみんなに利用されるのが望ましい．

文　献

〔I〕 D. G. Bobrow, B. Raphael ; A Comparison of List-Processing Computer Languages, Comm. ACM (Apr. 1964)

〔II〕 R. W. Floyd ; An Algorithm for Coding Efficient Arithmetic Operation, Comm. ACM (Jan. 1961)

〔III〕 A. Lapidus; Some Experiments in Algebraic Manipulations by Computer, Comm. ACM (Aug. 1965)

〔IV〕 J. Weizenbaum ; Symmetric List Processor, Comm. ACM (Sept. 1963)

〔V〕 A. Newell etc ; Information Processing Langurge-V Manual, Prentice-Hall. Inc. Englewood Cliffs, N.1 (1965)

本 PDF ファイルは 1968 年発行の「第 9 回プログラミング一シンポジウム報告集」をスキャンし、項目ごとに整理して、情報処理学会電子図書館「情報学広場」に掲載するものです。

この出版物は情報処理学会への著作権譲渡がなされていませんが、情報処理学会公式 Web サイトの https://www.ipsj.or.jp/topics/Past_reports.html に下記「過去のプログラミング・シンポジウム報告集の利用許諾について」を掲載して、権利者の捜索をおこないました。そのうえで同意をいただいたもの、お申し出のなかったものを掲載しています。

過去のプログラミング・シンポジウム報告集の利用許諾について

情報処理学会発行の出版物著作権は平成 12 年から情報処理学会著作権規程に従い、学会に帰属することになっています。

プログラミング・シンポジウムの報告集は、情報処理学会と設立の事情が異なるため、この改訂がシンポジウム内部で徹底しておらず、情報処理学会の他の出版物が情報学広場 (=情報処理学会電子図書館) で公開されているにも拘らず、古い報告集には公開されていないものが少からずありました。

プログラミング・シンポジウムは昭和 59 年に情報処理学会の一部門になりましたが、それ以前の報告集も含め、この度学会の他の出版物と同様の扱いにしたいと考えます。過去のすべての報告集の論文について、著作権者（論文を執筆された故人の相続人）を探し出して利用許諾に関する同意を頂くことは困難ですので、一定期間の権利者捜索の努力をしたうえで、著作権者が見つからない場合も論文を情報学広場に掲載させていただきたいと思います。その後、著作権者が発見され、情報学広場への掲載の継続に同意が得られなかった場合には、当該論文については、掲載を停止致します。

この措置にご意見のある方は、プログラミング・シンポジウムの辻尚史運営委員長 (tsuji@math.s.chiba-u.ac.jp) までお申し出ください。

加えて、著作権者について情報をお持ちの方は事務局まで情報をお寄せくださいますようお願い申し上げます。

期間：2020 年 12 月 18 日 〜 2021 年 3 月 19 日
掲載日：2020 年 12 月 18 日

プログラミング・シンポジウム委員会

情報処理学会著作権規程
https://www.ipsj.or.jp/copyright/ronbun/copyright.html