REDUCE USER'S MANUAL (REDUCE2)

理学部　東 秋夫

はじめに

このマニュアルは金沢大学計算機センターで利用できる数式処理システムREDUCE（REDUCE2）の機能を示し、またTSS端末からの利用方法を示したものである。FORTRAN、PL／Iなどの計算機言語が数値による計算を行なう数値処理言語であるのに対して、REDUCEは文字によって表わされた数式を直接処理する数式処理言語である。

対象とした読者は、

1、2章　TSSが使用できる人（FORTRAN等を少し）

3章　　TSSによるデータセットの処理ができる人

4章　　LISPについて多少知っている人

5章　　TSSでバッチ処理を行った経験のある人

とにかくREDUCEを起動して実行例を試み、うまくいったものから使ってみるのが良い。

なお、本文の図で「READY」、「@」、「>」等の後に続く下線を引いた部分が端末から入力する部分である。式の入力は例えば x^2 ならば X**2 とFORTRANにおける数式と同様の形式で行ない、これに対してREDUCEは

$$X^2$$

あるいは

X**2

でこれを表示してくる。

目　次

第1章　ＲＥＤＵＣＥの起動と終了

1．1　―起動―

ＴＳＳの起動については他のマニュアル（文献〔6〕、〔7〕）に詳しく書かれているのでここでは省略し、「READY」の表示されたＴＳＳコマンドモードからの起動方法を示す。

```
READY                          (ＴＳＳコマンド入力要求状態)
RREDUCE   [+E+]                ([+E+]はＥＮＴＥＲキーを押す)
 *** HLISP SYSTEM (VERSION = FEB. 01, '82)
 *** JOB PARAMETERS ***
       … メッセージ …
GBC L/H/1/2/3 =    49999/  27925/  135/  ･･･
    =  -9447
           =  DUMPFILE
           =  NIL
           =  REDUCE INITIALIZATION HAS COMPLETED !
           =  ( T . T)   (ここまでにしばらくの時間を要する)
@BEGIN( )  [+E+]               (ＨＬＩＳＰの入力要求状態からのＲＥＤＵＣＥの起動)
  REDUCE 2 (84/03/21/ VERSION APR-24-81)  ･･･
>                              (>はＲＥＤＵＣＥ入力要求状態を表わす)
```

ＲＥＤＵＣＥはＨＬＩＳＰと呼ばれるＬＩＳＰの一種で記述されており、その上で動作する。「RREDUCE」のセンタコマンドはＨＬＩＳＰを起動し、ＲＥＤＵＣＥ本体をその上に再生して初期化を行なう。それらの処理が終了すると@（アットマーク）を表示してＨＬＩＳＰの入力要求状態になって止まる。これでＲＥＤＵＣＥが動作するための環境が整ったわけで、「BEGIN( )」と入力してＬＩＳＰ上に定義された関数「BEGIN」を実行すればＲＥＤＵＣＥは起動する。関数ＢＥＧＩＮはＲＥＤＵＣＥへの入口関数である。

1．2　―終了―

```
> END;   [+E+]                  (ＲＥＤＵＣＥの終了)
       = ENTERING LISP  ･･･
```

@ /＊ ［＋E＋］　　（ＨＬＩＳＰの終了）

＊＊＊　・・・

　　… 終了メッセージ …

READY

「END」がＲＥＤＵＣＥからＨＬＩＳＰへの脱出に用いるＲＥＤＵＣＥコマンドである。「；」は入力の区切を表わすための文字で区切り子（ｔｅｒｍｉｎａｔｏｒ）と呼ばれる。ＬＩＳＰの入力状態になったら、「/＊」を入力すればＴＳＳコマンドモードにもどれる。「/＊」はＨＬＩＳＰで入力の終りを表わす為に用いられる文字列で、ＨＬＩＳＰ（ＲＥＤＵＣＥ）はこの文字列が入力されると実行を終了する。

1．3　―補足―

「RREDUCE」コマンドで起動するとＲＥＤＵＣＥ本体はコンパイルされたものが用いられる。コンパイルされていないＲＥＤＵＣＥの起動法についてはデータセットとの入出力の章で述べる。なお以後例の中でどの状態からの入力であるかは、次のようにして区別する。

ＲＥＡＤＹ	ＴＳＳコマンドモード
＞	ＲＥＤＵＣＥ入力モード
@	ＨＬＩＳＰ入力モード

注.　〔＋Ｅ＋〕はＥＮＴＥＲキーを押す事を表わす。

ＡＢ９９９９はそれぞれの利用者の課題番号

ＰＡＳＳＷＯＲＤはパスワード

第２章　　ＲＥＤＵＣＥの各機能の使い方

2．1　―はじめに―（形式的な事柄）

2．1．1　　文字

使用できる文字はＦＯＲＴＲＡＮなどにおけるものと変りがない。特殊な目的に用いられるのは次の５つである。

・；¥$　区切り子（ｔｅｒｍｉｎａｔｏｒ）。入力の区切りを表わすのに用いる。¥及び$が；と異なる点はこれらが区切り子として用いられると、結果の出力が抑制されるという点である。

・%　注釈文（ｃｏｍｍｅｎｔ）。この文字で始まった入力は区切り子が来るまでは注釈とみなされる。

・！　エスケープ（ｅｓｃａｐｅ）文字。そのままでは文字列の中に含める事ができない（特別な意味を持つ）文字の入力を可能にする。（２．２．３、３．５、４．２、４．４．２）

２．１．２　数

整数しか用いる事はできないが、商の形で有理数を用いる事ができるし、数を変数（文字列）で表わすという数式処理の特性から実用上問題がない。桁数はシステム上の制限以内ならばいくらあってもよい。

２．１．３　名前

英字で始まる英数字列（ＦＯＲＴＲＡＮ等と同じ）。ＲＥＤＵＣＥプログラムの中で特別な意味を持つ語（予約語）を名前として用いてはいけない。（ＲＥＤＵＣＥのコマンドに用いられる文字列）

２．１．４　変数

変数名名前と同じ形式の文字列

型	ＩＮＴＥＧＥＲ	整数を値として持つもの	（２．６．１）
	ＳＣＡＬＡＲ	一般の数式を値として持つもの	
	ＡＲＲＡＹ	配列データ	（２．４．１）
	ＭＡＴＲＩＸ	行列データ	（２．１２）

これらの型や宣言方法については後で説明があるのでここでは省略する。宣言なしで変数が用いられるとＳＣＡＬＡＲ変数であるとみなされる。

２．１．５　演算子

・：＝　　代入又は定義演算子

・ＯＲ　ＡＮＤ　ＮＯＴ　ＭＥＭＢＥＲ　　論理演算子

・＝　ＥＱ　ＮＥＱ　＞＝　＞　＜＝　＜　　比較演算子

・＋　－　＊　／　＊＊　　代数演算子

・　ＬＩＳＰのＣＯＮＳ又はベクトルの内積

◎演算の順序

一般には、

・A〔二項演算子〕B〔二項演算子〕Cは

（A［二項演算子］B）［二項演算子］C

・A：＝B：＝C　は　A：＝（B：＝C）

・A．B．C　は　A．（B．C）

◎各演算子の意味や演算順位はＦＯＲＴＲＡＮと同じ。

```
>3<5;  [+E+]                    (真ならばT)
 T
>1 NEQ 1;  [+E+]                (為ならば何も返さない)
>4 DIFFERENCE 1;  [+E+]         (ＬＩＳＰの演算子名で代用する事もできる)
 3                              (4－1の計算結果)
```

図1

2．1．6　E、I、COS、SIN、LOG

・変数名ＥとＩは次の特別な意味を持っている（予約変数）。

Ｅ　　自然対数の底

Ｉ　　虚数単位

・ＣＯＳ、ＳＩＮ、ＬＯＧは、それらの基本的な性質とともに組み込まれている。偏微分公式もＲＥＤＵＣＥは記憶しており、それについては偏微分演算子（ＤＦ）の項で述べる。（2．11）

```
>I**2;  [+E+]
 (-1)
>LOG(E);  [+E+]
 1
>COS(0);  [+E+]
 1
>SIN(-X);  [+E+]
  - SIN(X)
```

図2

2．2　―変数への値の代入、結果の保存、変数からの値の除去―

2．2．1　変数への値の代入

代入には演算子：＝を用いる。

形式

・〔変数名〕：＝〔ＲＥＤＵＣＥで許される式などの表現、又は値〕

・〔変数名〕：＝〔変数名〕：＝…：＝〔値〕

2．2．2　結果の保存

代入文を評価して結果を得た場合を除いて、処理の結果は画面上に出力されるだけで後には残ってくれない。ＲＥＤＵＣＥでは直前の結果は＊ＡＮＳという変数に代入されている。この変数の値は処理の実行とともに変化してしまうのでこれを別の変数に移しておけば結果が保存できる。そのためにはＳＡＶＥＡＳを使う。

形式　　SAVEAS　〔変数名〕

2．2．3　変数からの値の除去

値が代入されていない変数を評価するとその変数名自体が返される。そのような状態に変数をもどすにはＣＬＥＡＲを使う。

形式　　CLEAR　〔変数名〕，…，〔変数名〕

```
>A:=X+Y;   [+E+]          (:=を使って左辺の変数に代入する)
 A  :=  X  +  Y
>B:=Y+Z;   [+E+]
 B  :=  Y  +  Z
>A*B;  [+E+]              (A*Bを求める)
                   2
 X*Y  +  X*Z  +  Y    +  Y*Z
>SAVEAS C;  [+E+]         (SAVEASで変数Cに結果を保存)
>C;  [+E+]                (変数を評価すると内容が値として返ってくる)
                   2
 X*Y  +  X*Z  +  Y    +  Y*Z
>A/B;  [+E+]
  (X  +  Y)/(Y  +  Z)
>!*ANS;     [+E+]         (!は通常文字列の一部として含む事のできない文字を
文字列の一部として入力可能にする。＊ＡＮＳに結果が入っている)
```

```
  (X + Y)/(Y + Z)
>A; [+E+]                          (変数Aを評価してその値を得る)
 X + Y
>B;  [+E+]
 Y + Z
>CLEAR A,B;  [+E+]                 (ＣＬＥＡＲで変数から値を除去する)
>A;B;  [+E+]
  A
 B
```

図3

2．3　―条件による処理の分岐（ＩＦ文）―

条件分岐にＩＦ～ＴＨＥＮ～ＥＬＳＥ～の構文が使える。

形式　　IF〔条件〕　THEN〔実行文〕 ELSE〔実行文〕

ＥＬＳＥ節は省略できる。

```
>S:=-1;  [+E+]
 S := (-1)
>IF S>0 THEN S:=P  [+E+]
                                   (；が入力されるまで複数の行にわたって入力できる)
>ELSE  [+E+]
>          IF S=0 THEN S:=Z  [+E+]
>                    ELSE S:=N;  [+E+]
 N                                 (返された値はＩＦ文内での実行による結果)
>S;  [+E+]
 N                                 (ＳにはＮが代入された)
```

図4

2．4　―配列の宣言と繰り返し１（ＦＯＲループ）―

2．4．1　配列の宣言と除去

変数が配列データを持つ事を宣言するときＡＲＲＡＹを使う。配列は０から始まり次のような形式

で宣言する。

形式　　ARRAY〔配列名〕（〔大きさ〕），…，〔配列名〕（〔大きさ〕）

配列データを除去するのにはＲＥＭＡＲＲＡＹを用いる。

形式　　REMARRAY〔配列名〕，…，〔配列名〕

2．4．2　ＦＯＲループ

繰り返し変数による繰り返し処理を行う。

2．4．2．1　ＦＯＲループの標準形

形式　　FOR〔繰り返し変数〕：＝〔開始値〕　STEP〔増分〕　UNTIL〔終値〕
DO〔実行文〕

ＦＯＲ文における繰り返し変数はＩＮＴＥＧＥＲ（整数値を値とする変数）とみなされ、ＦＯＲ文の中では新めて宣言されて用いられるので、同じ変数名でも外部の変数とは独立となる。従ってここに外部で虚数単位を表わす変数名Ｉを使用する事もできる。

2．4．2．2　〔増分〕が１の時のＦＯＲループの省略形

形式　　FOR〔繰り返し変数名〕：＝〔開始値〕：〔終値〕　DO〔実行文〕

2．4．2．3　ＦＯＲループを使ってそれぞれの実行結果の和を求める事

形式　　FOR〔繰り返し変数名〕：＝〔開始値〕：〔終値〕　SUM〔実行文〕

2．4．2．4　ＦＯＲループを使ってそれぞれの実行結果の積を求める事

形式　　FOR〔繰り返し変数名〕：＝〔開始値〕：　〔終値〕　PRODUCT〔実行文〕

```
>ARRAY H(5);  [+E+]
                (宣言ＡＲＲＡＹを用いて配列名Ｈで大きさが０～５の配列を宣言した)
>FOR I:=0 STEP 1 UNTIL 5 DO H(I):=A**I;  [+E+]
>H(3);  [+E+]
  3
 A
>FOR I:=0:4 DO H(I):=H(I)+H(I+1);  [+E+]
                                    (ＦＯＲループの省略形)
>H(3);  [+E+]
```

```
   3
  A *(A + 1)
>FOR I:=0:3 SUM A**I;   [+E+]
```

(和を求めるためのＦＯＲ～ＳＵＭ～の組み合わせ)

```
   3     2
  A   + A   + A + 1
>FOR I:=0:6 PRODUCT A**I;   [+E+]
```

(積を求める。ＦＯＲ～ＰＲＯＤＵＣＴ)

```
   21
  A
```

図5

2．5　―手続きのブロック化（ＢＥＧＩＮ－ＥＮＤブロック、複合文）―

ＦＯＲループやＩＦ文そして後に述べるＷＨＩＬＥループやＰＲＯＣＥＤＵＲＥ文の中に出てくる〔実行文〕の所には通常一つの実行文だけを置く事ができるので、あまり複雑な処理を行なう事はできない。たとえばＩＦ文の例においては単に代入を行なっているだけである　（2．3）。その様な場所にこのＢＥＧＩＮ－ＥＮＤブロックを用いれば、より複雑な処理を行う事ができる。ＢＥＧＩＮ－ＥＮＤブロックはＰＬ／ＩやＰＡＳＣＡＬなどにも登場するが、ＲＥＤＵＣＥではＬＩＳＰという背景によって、関数の引数の部分にＢＥＧＩＮ－ＥＮＤブロックを置く事も可能になっている。

形式　　BEGIN；〔実行文〕；…〔実行文〕；END

ＢＥＧＩＮ－ＥＮＤブロックは一つの値を返すいわば手続き関数で、ブロック内のＲＥＴＵＲＮ文の後に書かれた実行文の結果が値として返される。

形式　　RETURN　〔実行文〕；

またＢＥＧＩＮ－ＥＮＤブロック内ではラベルの書かれた位置へ無条件で制御を移すＧＯＴＯ文が使える。

形式　　（ＢＥＧＩＮ－ＥＮＤブロック内で）

```
〔ラベル名〕：〔実行文〕；
                :
                :
〔実行文〕；
GOTO      〔ラベル名〕（；はＧＯＴＯのラベルの後ろにはない）
```

ラベル名は変数名と同形式のものを用いる。

```
>BEGIN;  [+E+]
>   N:=0;  [+E+]
>   M:=0;  [+E+]
>L: IF N=3 THEN RETURN M; [+E+]          (ラベル名にＬを使った)
>   M:=M+SIN(   [+E+]   (こういう所にもＢＥＧＩＮ-ＥＮＤブロックは置ける)
>              BEGIN;  [+E+]
>               U:=X**N*N;  [+E+]
>               U:=U+N;  [+E+]
>                RETURN U;  [+E+]
>           END          );  [+E+]
                                   (ＥＮＤの後ろにＳＩＮのかっこがきている)
>   N:=N+1;  [+E+]
>   GOTO L   [+E+]          (ＧＯＴＯラベルの後ろに；は付けない)
>END;  [+E+]
                      2
 SIN(X + Y)+ SIN(2*X  + 2)          (ＲＥＴＵＲＮ文で返された値)
>M;   [+E+]                         (変数Ｍに結果が代入された)
                      2
 SIN(X + 1) + SIN(2*X  + 2)
>M:=0;  [+E+]                       (変数Ｍの初期化)
M := 0
>FOR J:=0:3 DO BEGIN;  [+E+]
>                  M:=M-A**J;  [+E+]
>               END;  [+E+]
>M;   [+E+]                         (Ｍを評価して結果を出力)
      3    2
  - (A  + A  + A + 1)
```

図6

2．6 —繰り返し2（ＷＨＩＬＥループ）—

2．6．1　整数を値として持つ変数の宣言（ＩＮＴＥＧＥＲ）

形式　　INTEGER　〔変数名〕，……，〔変数名〕

同じ0でも文字としての0と数値としての0とは計算機内部での表現の仕方が違う。計算機が大小の比較ができるのは数値として表された数字であり、今の場合ＷＨＩＬＥループの〔条件文〕の所で大小の判定を行ないたいのでＩＮＴＥＧＥＲ宣言をする。

2．6．2　ＷＨＩＬＥループ

形式　　WHILE　〔条件文〕　DO　〔実行文〕

〔条件文〕が真（Ｔ）である間〔実行文〕の所を繰り返す。

```
>INTEGER N;  [+E+]
    (ＷＨＩＬＥループの条件文で大小比較を行なうため、変数をＩＮＴＥＧＥＲ宣言しておく)
>N:=0;M:=0;  [+E+]
        (；で区切って１行に複数の実行文を書ける。ＲＥＤＵＣＥの入力に行の概念はない)
 N := 0
 M := 0
>WHILE N<=3 DO  [+E+]                    (ＤＯの後ろに；はない)
> BEGIN;  [+E+]
        (ＤＯの後ろに複数の実行文をおきたいのでＢＥＧＩＮ－ＥＮＤブロックを用いた)
>   M:=M+A**N;  [+E+]
>   N:=N+1;  [+E+]
> END;  [+E+]                  (ＢＥＧＩＮ－ＥＮＤブロックの終り)
>M;  [+E+]                     (実行結果の代入された変数Mを評価する)
   3     2
 A  + A  + A + 1
>CLEAR M;  [+E+]               (結果が出たので変数Mの値をＣＬＥＡＲしておく)
```

図7

2．7　—ＢＥＧＩＮ－ＥＮＤブロックの簡略形（ＧＲＯＵＰ－ＥＸＰＲＥＳＳＩＯＮ）—

ＢＥＧＩＮ－ＥＮＤブロックの様な種々の機能は持っていないが複数の実行文を置くという目的にはＧＲＯＵＰ－ＥＸＰＲＥＳＳＩＯＮが利用できる。

形式　　＜＜〔実行文〕；……〔実行文〕；〔実行文〕＞＞；

注.　宣言を含む事はできない。

ＩＦ文のＥＬＳＥは省略できない。

```
>INTEGER N;  [+E+]
>N:=0;  [+E+]
 N := 0
>M:=0;  [+E+]
 M :=0
>WHILE N<=3 DO << M:=M+A**N; N:=N+1 >> ;  [+E+]
                                    (Mに結果が入っている)
>M;  [+E+]
  3    2
 A  + A  + A + 1
```

図8

2．8 —代入則（ＬＥＴ文）—

2．8．1　ＬＥＴ文

数式の計算には変数なり式の一部を他のもので置き換えたりしたい場合がよくある。
例えば ＳＩＮ（Ｘ）＊＊２＋ＣＯＳ（Ｘ）＊＊２
があった時 $\sin^2 x+\cos^2 x=1$ の関係を使って１になって欲しい。そのような処理を行なうために代入則を定義する。

形式　　ＬＥＴ　〔数式－１〕＝〔数式－２〕，…，〔数式－１'〕＝〔数式－２'〕

この宣言によって〔数式－１〕は〔数式－２〕に置き換えられる。この様な処理を計算機が行なう為にはＬＥＴ文で定義された＝（等号）の左辺の形式を与えられた式中から捜すという操作（パターンマッチング）が必要となる。ＲＥＤＵＣＥは積（＊）の形の表現に対してまでこの操作が行なえるようになっている。（＋、－、／で項が結ばれた式は代入則の左辺には置く事ができない）。

```
>M:=SIN(X)**2+COS(X)**2;  [+E+]
                2          2
 M := COS(X)  + SIN(X)
>M;  [+E+]
       2          2
 COS(X)  + SIN(X)
```

```
>LET SIN(X)**2=1-COS(X)**2;  [+E+]          (代入則の定義)
>M;  [+E+]
 1                                          (結果は1になった)
```

図9

2．8．2　任意変数を含んだ代入則（ＦＯＲＡＬＬ～ＬＥＴ～）

まず例を見て下さい。

```
>M:=COS(Y)**2+SIN(Y)**2;  [+E+]          (変数XをYに換えた)
            2          2
 M:= COS(Y)  + SIN(Y)
>M;  [+E+]
       2          2
 COS(Y)  + SIN(Y)                        (前例で定義した代入則は実行されない)
```

図10

XがYに置き換えられると代入則は実行されていない。実は先のＬＥＴ文による代入則定義の仕方では引数が別の変数の時も代入則を実行するという任意性が定義されていないのである。式の一部に任意変数を含ませた代入則の宣言は次の様にして行なう。

形式　　FORALL　〔任意変数名〕，……，〔任意変数名〕
　　LET　〔代入則－１〕，……，〔代入則－Ｎ〕

2．8．3　代入則の除去

代入則の除去はそれを設定した時と同じ形式を用いて行なう。

形式　　CLEAR　〔代入則の左辺－１〕，……，〔代入則の左辺－Ｎ〕

又は　　FORALL　〔変数名〕，……，〔変数名〕
　　CLEAR　〔代入則の左辺－１〕，……，〔代入則の左辺－Ｎ〕

```
>FORALL X LET SIN(X)**2=1-COS(X)**2;  [+E+]
                                          (任意変数を含んだ代入則の宣言)
>M;  [+E+]
 1                                        (代入則が実行された)
>FORALL X CLEAR SIN(X)**2;  [+E+]
```

(代入則の除去。代入則を定義した時の左辺と全く同一の表現でＣＬＥＡＲする)

```
>SIN(X)**2;  [+E+]
            2
  - COS(X)  + 1                    (前例で定義した代入則はまだ残っている)
>SIN(Y)**2;  [+E+]
          2
 SIN(Y)                            (今回のものは消えている。そのまま返された)
>CLEAR SIN(X)**2;  [+E+]           (前例で定義した代入則の除去)
>SIN(X)**2;  [+E+]
          2
 SIN(X)                            (代入則は消えた)
```

図11

2．8．4　べきも一致した時だけの代入則の実行（ＭＡＴＣＨ）

Ｐ＋Ｐ＊＊２＋Ｐ＊＊３に代入則Ｐ＊＊２＝Ａを実行することを考える。この場合先のＬＥＴ文によってＬＥＴ　Ｐ＊＊２＝Ａと定義するとＰ＊＊３の項も代入則の適用を受けてＡ＊Ｐの形に変ってしまう。この時Ｐ＊＊２についてだけ代入則を実行したいのならば次のＭＡＴＣＨが使える。

形式　　MATCH　〔代入則〕

```
>M:=P+P**2+P**3;  [+E+]
             2
 M := P*(P  + P + 1)
>MATCH P**2=A;  [+E+]
>M;  [+E+]
       3
 A + P  + P
>CLEAR P**2;  [+E+]      (ＭＡＴＣＨで定義した代入則の除去)
>M;  [+E+]
      2
 P*(P  + P + 1)
```

図12

2．8．5　局所的な代入則の実行（ＳＵＢ）

ある式について一時的に代入則を実行したい場合ＳＵＢを用いる。

形式　　SUB　（〔代入則〕，……，〔代入則〕，〔式〕）

```
>M:=(X+1)**2;   [+E+]
        2
 M:= X  + 2*X + 1
>SUB(X=Y+Z,M);  [+E+]
```

(式Mに対してXをY＋Zに置き換える。この代入則は一時的であとには残らない)

```
  2                          2
 Y  + 2*Y*Z + 2*Y + Z  + 2*Z + 1
```

図13

2．8．6　演算子の定義（OPERATOR、INFIX）

変数名が演算子名である事を宣言するのにOPERATORを用いる。

形式　　OPERATOR　〔演算子名〕，……，〔演算子名〕

この宣言によってPREFIX演算子が宣言される。INFIX演算子を宣言するには次の様にする。

形式　　INFIX　〔演算子名〕，……，〔演算子名〕

```
>OPERATOR A;   [+E+]
>A(B+C);   [+E+]
 A(B + C)
>FORALL X,Y LET A(X+Y)=A(X)+A(Y);   [+E+]
>A(B+C+D);  [+E+]
 A(B) + A(C) + A(D)
>FORALL X,Y LET A(X*Y)=A(X)*A(Y),A(-X)=-A(X),
 A(X/Y)=A(X)/A(Y);   [+E+]
>A((B-C)/(D+E)/(F-G));  [+E+]
 (A(B) - A(C))/(A(D)*A(F) - A(D)*A(G) + A(E)*A(F)
 - A(E)*A(G))
>FORALL X,Y CLEAR A(X+Y),A(X*Y),A(-X),A(X/Y);   [+E+]
>INFIX B;  [+E+]                    (BをINFIX演算子であると宣言する)
>FORALL X,Y LET (X B Y)=X**Y;  [+E+]
>X B Y B Z;   [+E+]
    (Y*Z)
```

X　　（X B Y B Zは(X B Y) B Zになる）

図14

2．8．7　結果の演算子（関数）としての保存

ＳＡＶＥＡＳで結果が変数に保存されるが、ある変数に関する演算子として保存するには次の方法を用いる。なお〔演算子名〕はあらかじめＯＰＥＲＡＴＯＲ宣言しておく必要がある。

形式　　FORALL　〔変数名〕，……，〔変数名〕
　　SAVEAS　〔演算子名〕（〔変数名〕，……，〔変数名〕）

```
>OPERATOR ZZZ;  [+E+]
>(X+Y)**2;  [+E+]
  2          2
 X + 2*X*Y + Y
>FORALL X,Y SAVEAS ZZZ(X,Y);  [+E+]
>ZZZ(U,V);  [+E+]
  2          2
 U  + 2*U*V + V
```

図15

2．9　―演算子（関数）に具体的な性質を持たせる方法―

2．9．1　代入則による方法

代入則（ＬＥＴ文）を演算子に対して定義していく事によって、演算子に具体的な性質を持たせる事ができる。（2．8．2）、（2．8．6）参照。

2．9．2　手続きの関数化（ＰＲＯＣＥＤＵＲＥ）

ＰＲＯＣＥＤＵＲＥ宣言によって具体的に内容を記述して関数を定義することができる。

形式　　〔ＰＲＯＣＥＤＵＲＥ　ＴＹＰＥ〕　PROCEDURE　〔手続き名〕（〔変数名〕，……，〔変数名〕）；〔手続きを表わす実行文〕；

〔ＰＲＯＣＥＤＵＲＥ　ＴＹＰＥ〕　ＩＮＴＥＧＥＲ　整数値を返す関数である事を表わす。
　　ＡＬＧＥＢＲＡＩＣ　一般の数式を返す関数である事を表わす。
　　ＳＹＭＢＯＬＩＣ　ＬＩＳＰ様のデータを返す関数であ

る事を表わす。（4．3．3．2）

省略時の解釈値はＡＬＧＥＢＲＡＩＣ.

2．9．2．1　文字列、値の出力（ＷＲＩＴＥ）

ＢＥＧＩＮ-ＥＮＤブロックやＦＯＲループの計算途中の値を出力させたり、文字列を出力させるのにＷＲＩＴＥを用いる。

形式　　WRITE　〔出力したいもの〕，……，　〔出力したいもの〕

```
>PROCEDURE YON(A,B);  [+E+]
                                    (手続き名はYON、内容は四則計算である)
>BEGIN;  [+E+]
>WRITE "WA=",A+B;  [+E+]                    (ＷＲＩＴＥ文の例)
>WRITE "SA=",A-B;  [+E+]
>WRITE "SEKI=",A*B;  [+E+]
>WRITE "SYOU=",A/B;  [+E+]
>RETURN "OWARI";  [+E+]
>END;  [+E+]
>YON(G,H);  [+E+]                           (手続きの実行)
 WA=G + H
 SA=G - H
 SEKI=G*H
 SYOU=G/H
 OWARI
```

図16

2．9．3　線形演算子の定義（ＬＩＮＥＡＲ、ＤＥＰＥＮＤ）

線形演算子を考える時、数式処理ではこれらが係数と考えているものも変数と考えているものも文字で表わされる為、システムにはその区別がつかない。従ってＲＥＤＵＣＥでは線形演算子の真の引数を第一引数で表わし、その後にその演算子が何を変数と考えた線形演算子であるかを指定するための変数名が続く。

形式　　〔線形演算子名〕（〔引数（式）〕，〔変数名〕，……，〔変数名〕）

この様な形で線形演算子は表わされるが、変数名が多くなると表現が長くなって計算機の記憶域の消費が多くなるので不都合である。変数の他への依存性を定義するＤＥＰＥＮＤを用いれば簡潔になる。

形式　　DEPEND〔変数名－0〕，〔変数名－1〕，……，〔変数名－N〕

この宣言によって〔変数名－0〕がその後に続く変数に依存する事を宣言できる。依存性の解除にはＮＯＤＥＰＥＮＤを用いる。

形式　　NODEPEND　〔変数名－0〕，〔変数名－1〕，……，〔変数名－N〕

```
>OPERATOR A;   [+E+]
>LINEAR A;   [+E+]
>A(U*X+V*Y,X,Y);   [+E+]
 U*A(X,X,Y) + V*A(Y,X,Y)
>DEPEND X,T;   [+E+]
>DEPEND Y,T;   [+E+]
>A(U*X+V*Y,T);   [+E+]
 U*A(X,T) + V*A(Y,T)
```

図17

2．10　－近似計算（ＷＨＩＧＨＴ、ＷＴＬＥＶＥＬ）－

高次項を無視する場合、例えばＬＥＴ　Ｘ＊＊４＝０などと代入則を定義しておいても良いのであるが、ＲＥＤＵＣＥでは各変数に重みを付け、この重みの大きさによって項を無視して近似計算を行なう機能がある。

2．10．1　変数に重みを付ける方法

宣言ＷＥＩＧＨＴを用いる。重みは整数値で与える。

形式　　WEIGHT　〔変数名－1〕＝〔重み－1〕，……，〔変数名－N〕＝〔重み－N〕

2．10．2　重みの制限値の設定

次の方法で設定する制限値を越えた重みを持つ項は無視される。

形式　　WTLEVEL　〔重みの制限値〕

2．10．3　項の重みの計算方法

変数　V1，・・・，VN の重みが　W1，・・・，WNで項の形が次の様な時。

　　(V1＊＊C1)＊・・・＊(VN＊＊CN)

重みは

W1*C1+ ・・・・+WN*CN

で計算される。

```
>WEIGHT X1=2,X2=3;   [+E+]        (変数に重みを付ける)
>WTLEVEL 8;   [+E+]               (切り捨てる重みを8とする)
>(X1+X2)**3;   [+E+]
          2                  2
 X1*(3*X2  + 3*X1*X2 + X1  )
                    (X2**3の項は重みが9になるので切り捨てられた)
```

図18

2．11 —偏微分—

2．11．1 偏微分演算子（DF）

形式　DF（〔式〕，〔微分変数名－1〕，〔変数名－1に対する偏微分回数〕，……，〔変数名－N〕，〔変数名－Nに対する偏微分回数〕）

・偏微分の回数が1の時は偏微分回数を省略できる。

2．11．2 微分則の定義方法

微分則はＦＯＲＡＬＬ～ＬＥＴ～の代入則によって定義する。（2．8．2）

```
>M:=Z*(X+Y)**4;   [+E+]
          4       3         2  2         3    4
 M := Z*(X  + 4*X  *Y + 6*X  *Y  + 4*X*Y  + Y  )
>DF(M,X,Y,2,Z);   [+E+]     (MをXで1回、Yで2回、Zで1回偏微分した)
 24*(X + Y)
>DF(SIN(X),X);   [+E+]  (ＳＩＮなどの関数の微分則はあらかじめ記憶してある)
 COS(X)
>DF(LOG(X),X);   [+E+]
 1/X
>OPERATOR DD;   [+E+]              (ＤＤが演算子である事を宣言する)
>FORALL X LET DF(DD(X),X)=1/DD(X)   [+E+]
                    (演算子ＤＤにＦＯＲＡＬＬ～ＬＥＴ～を用いて微分則を定義する)
```

```
>DF(DD(X),X,2);  [+E+]
                  3
  ( - 1)/DD(X)
```

図19

2．12 —行列計算—

行列データを持つ変数である事を宣言するにはMATRIXを使う。行列の大きさはM（3，4）などのように明示して宣言しても良いし、そうでなくても良い。明示されなかった時には、行列データが与えられた時に決定される。

形式　MATRIX 〔行列名〕（〔大きさ〕，〔大きさ〕），……，
〔行列名〕（〔大きさ〕，〔大きさ〕）

行列データはMAT関数を用いて表わす。

形式　MAT（（〔要素－1，1〕……，〔要素－1，N〕），……，
（〔要素－M，1〕，……，〔要素－M，N〕））

は ┌〔要素－1，1〕…〔要素－1，N〕┐ の形の行列データを表わす。
　 │　　：　　：　　│
　 └〔要素－M，1〕…〔要素－M，N〕┘

行列については種々の演算関数が用意されている。

```
>MATRIX MM,NN;  [+E+]                    (行列変数である事の宣言)
>MM:=MAT((M11,M12),(M21,M22)); [+E+]
                    (MAT関数を使って行列データを行列変数に代入する)
 MM(1,1) := M11
 MM(1,2) := M12                  | M11 M12 |
 MM(2,1) := M21                  | M21 M22 |
 MM(2,2) := M22
>NN:=MAT((N11,N12),(N21,N22));  [+E+]
                    (REDUCEからの出力は長くなるので省略する)
>MM+NN;   [+E+]     (行列の和)                    ↑
>MM-NN;   [+E+]     (差)                          |
>MM*NN;   [+E+]     (積)                          |
>1/NN;    [+E+]     (逆行列)      (出力結果は省略した)
```

```
>MM/NN;   [+E+]          (商は逆行列との積を意味する) |
>MM**2;   [+E+]          (行列のべき乗)               |
>TP(MM);  [+E+]          (転置行列)                   |
>DET(MM); [+E+]          (行列式)                     |
>TRACE(MM); [+E+]        (トレース)                   |
>CONST*MM; [+E+]         (行列でない通常の変数との積) ↓
```

図20

2．13　—出力式の制御—

2．13．0　フラグについて（ON、OFF）

REDUCEでは処理を制御するのにフラグを使う。種々の処理を指定するフラグをONにしたり、OFFにしたりする事によって、利用者は処理を制御することができる。

形式　　ON　〔フラグ名〕，……，〔フラグ名〕
　　OFF　〔フラグ名〕，……，〔フラグ名〕

2．13．1　展開（EXPフラグ）

展開はEXPフラグで制御する。

```
>OFF EXP;  [+E+]                    (展開を抑制する)
>(X+Y)**2;  [+E+]
          2
  (X + Y)                           (展開しないで返された)
>ON EXP;   [+E+]
>(X+Y)**2;  [+E+]
  2               2
 X  + 2*X*Y + Y                     (展開して返された)
```

図21

2．13．2　変数の並びの制御（ORDER宣言）

通常の状態では変数はアルファベット順で出力される。その出力順を利用者が指定するときORDER宣言を使う。宣言は変更のなかった部分については前のORDER宣言によるものが優先され、後の宣言は先の宣言による指定の後ろに付け加えるという形で行われる。

```
>CLEAR X,Y,Z;   [+E+]                    (変数の値を除去)
>X+Y+Z;   [+E+]
 X + Y + Z                               (通常出力はアルファベットの順になる)
>ORDER Y,Z,X;   [+E+]                   (変数の並びの指定)
>X+Y+Z;   [+E+]
 Y + Z + X                               (Y、Z、Xの順になった)
>ORDER Z;  [+E+]
 Y + X + Z              (Y、Xに対する古い宣言がZに対するものより優先される)
>ORDER X,Y,Z;   [+E+]
 X + Y + Z                               (X、Y、Zの順になった)
```

図22

2．13．3　変数のくくり出し（ＡＬＬＦＡＣフラグ）

このフラグがＯＮの時、式全体を通しての単純な乗数項はくくり出された形で出力される。

2．13．4　指定した変数でのべき級数の形への式の変形（ＦＡＣＴＯＲ宣言）

ＦＡＣＴＯＲはＡＬＬＦＡＣフラグをＯＦＦにして使うとうまくゆく。ＦＡＣＴＯＲで宣言した変数に対して係数をくくり出してべき級数の形に直す。指定の解除はＲＥＭＦＡＣ宣言を用いて行なう。

```
>OFF ALLFAC;   [+E+]          (ＡＬＬＦＡＣフラグは通常ＯＮになっている)
>FACTOR X;  [+E+]              (変数Ｘに対するべき級数形にする事を指定)
>X*Y**2+X**2*Y+2*Z*X*Y;   [+E+]
  2          2
 X *Y + X*(Y  + 2*Y*Z)
>REMFAC X;   [+E+]             (変数Ｘに対するくくり出し指定を解除する)
>ON ALLFAC;  [+E+]             (全変数に対するくくり出し指定)
>X*Y**2+X**2*Y+2*Z*Y*X;   [+E+]
 X*Y*(X + Y + 2*Z)        (結果は式全体に対するくくり出しになっている)
```

図23

2．13．5　割り算を指数形で出力する（ＤＩＶフラグ）

／を使って割り算で表わしていたものを負のべきを使って指数形で出力する。

```
>(X+Y)/X;  [+E+]
  (X + Y)/X
>ON DIV;   [+E+]                    (ＤＩＶフラグは通常ＯＦＦ)
>(X+Y)/X;  [+E+]
    (-1)
  X     *Y + 1                      (割り算が指数形で出力された)
>OFF DIV;  [+E+]                    (ＤＩＶフラグを通常の状態にもどしておく)
```

図24

2．13．6　ＲＡＴフラグ

ＦＡＣＴＯＲ宣言で式を指定した変数についてのべき級数の形に直せる事を述べたが、式全体が別の変数で割られている場合、ＦＡＣＴＯＲ宣言をしても割り算による式全体のくくり出しはまだ残っている。これを項ごとの割り算の形に直すのに用いる。

```
>(X+Y)/(X*Z);   [+E+]
  (X + Y)/(X*Z)
>OFF ALLFAC;   [+E+]
>FACTOR X;   [+E+]
>(X+Y)/(X*Z);   [+E+]
  (X + Y)/(X*Z)
>ON RAT;   [+E+]
>(X+Y)/(X*Z);   [+E+]
 X/(X*Z) + Y/(X*Z)
```

図25

2．13．7　式を各項ごとに一行として出力する　(ＬＩＳＴフラグ)

通常このフラグはＯＦＦでＲＥＤＵＣＥは式をベタ打ちしてくるので項の対応がつかみにくい。それを考慮して出力させたいときＬＩＳＴフラグをＯＮにする。

```
>X*(Y+Z*(W+1));  [+E+]
 X*(W*Z + Y + Z)
>ON LIST;  [+E+]                    (ＬＩＳＴフラグは通常ＯＦＦになっている)
>X*(Y+Z*(W+1));  [+E+]
```

```
 X*(W*Z                                    (ＬＩＳＴフラグがＯＮの時の出力)
      + Y
      + Z)
>OFF LIST;  [+E+]
```

図26

2．13．8　有理式の和を求める時に共通分母にするかどうかの制御（ＭＣＤフラグ）

```
>1/X+1/Y;  [+E+]
  (X + Y)/(X*Y)                          (ＭＣＤフラグは通常ＯＮ)
>OFF MCD;  [+E+]
>1/X+1/Y;  [+E+]
   (-1)     (-1)
  X      + Y
>ON MCD;   [+E+]                         (フラグを元にもどす)
```

図27

2．14　―係数の取り出し（ＣＯＥＦＦ）―

式をべき項の和とみなした時のべき項の係数を取り出すのに関数ＣＯＥＦＦを用いる。式をべき項の和の形に直す方法はＦＡＣＴＯＲ（2．13．4）とＲＡＴ（2．13．6）を参照して欲しい。

形式　　COEFF（〔式〕，〔べき項とみなす変数名〕，〔係数を代入する変数名〕）

係数を代入する変数名が通常の変数名の時

〔係数を代入する変数名〕〔べき項の次数〕

の形を変数名として持つ変数が新しく作られ、そこに係数は代入される。

```
>OFF ALLFAC;  [+E+]                                    ↑
>M:=(Y+1)*X**2+(Y**3+3)*X+Y;  [+E+]                    |
        2      2        3                              |
 M := X *Y + X  + X*Y  +3*X + Y  (式をＸについてべき項の和に変形)
>FACTOR X;  [+E+]                                      |
>M;  [+E+]                                             |
  2                  3                                 |
 X *(Y + 1) + X*(Y  + 3) + Y                           ↓
```

```
>COEFF(M,X,COE);  [+E+]
```

(式MをXについてのべき項の和とみなし係数を頭文字ＣＯＥの変数を作ってそれに代入させる)

```
 *** COE2 COE1 COE0 ARE NON ZERO
```

(変数がシステムによって作られた事によるメッセージ)

```
 2
```

(ＣＯＥＦＦの返す値は指定した変数の最大次数)

```
>COE2;COE1;COE0;  [+E+]
```

(それぞれの変数に代入された値の表示)

```
 Y + 1
  3
 Y  + 3
 Y
```

```
>REMFAC X;  [+E+]
```

(Xについてのべき項指定を解除)

図28

配列名を指定しておけばそこに代入される。

```
>FACTOR Y;  [+E+]                                    ↑
>M;  [+E+]                                           |
  3          2           2   (式Mを今度はYについてのべき項の形に直した)
 Y *X + Y*(X  + 1) + X  + 3*X                        ↓
>ARRAY COE(3);  [+E+]
```

(配列名をＣＯＥとし大きさ０～３で宣言する)

```
>COEFF(M,Y,COE);  [+E+]
```

(配列ＣＯＥにＹの係数を代入)

```
 3
```

(Ｙの最大次数は３であった)

```
>COE(3);COE(2);COE(1);COE(0);  [+E+]
```

(配列の各値を出力してみる)

```
 X
 0
  2
 X  + 1
  2
 X  + 3*X
```

図29

配列の断面を＊で指定してそこに代入させる事もできる。
C（3，2）の配列でC（2，＊）として断面を指定するとC（2，0）～C（2，2）に代入される。

```
>ARRAY COEF(3,3);   [+E+]
                        (配列名ＣＯＥＦで大きさを（0～3）×（0～3）で指定)
>COEFF(M,Y,COEF(2,*));   [+E+]
                    (変数Ｙの係数をＣＯＥＦ（2、＊）で配列ＣＯＥＦの断面に代入させる)
 3                                    (Ｙの最大次数は3)
>COEF(2,0);COEF(1,0);   [+E+]
                                       (配列要素の一部を出力してみた)
  2
 X  + 3*X
 0
>REMFAC Y;   [+E+]                    (式の変数Ｙに対するべき項指定を解除)
>ON ALLFAC;   [+E+]   (ＡＬＬＦＡＣフラグも通常のＯＮの状態に回復しておく)
```

図30

2．15 —有理式の分子、分母の取り出し（NUM、DEN）—

ＮＵＭは有理式の分子をＤＥＮは分母を取り出す関数である。

```
>OFF EXP;  [+E+]
                (入力した形式ほまま式を取り扱いたいのでＥＸＰフラグをＯＦＦにした)
>NUM((U+V)/(W+X));   [+E+]      (有理式の分子を取り出す)
 U + V                          (分子が値として返された)
>DEN((U+V)/(W+X));   [+E+]      (分母の取り出し)
 W + X                          (分母が値として返された)
>ON EXP;   [+E+]                (ＥＸＰフラグを元へもどした)
```

図31

2．16 —アテンションキー（［ＰＡ1］）を押した時の操作—

```
> !                                    (「ALT」キーを押しながら「PA1」キーを押した)
  RETRY/CREAR=(R/C)=?
R  [+E+]                               (RETRYを選択)
  SYSTEM IN ILLEGAL STATE. EXIT BY 'XQ RN=1' AT QUIT MODE
     A  NIL
     W  NIL
     R  NIL
     R  NIL
>REDUCE( ) [+E+]
  (REDUCEを再び初期化する。「>」が出力されているがLISPの入力状態になっている)
  REDUCE INITIALIZATION HAS COMPLETED !
>BEGIN( ) [+E+]                        (REDUCEの再起動)
  REDUCE 2 (84/03/22 /VERSION APR-24-81) ...
>
```

図32

第3章データセットとの入出力操作

REDUCEはDATA属性を持つデータセットに対してデータの入出力が可能である。入力データセットにはメンバーを持った区分データセットもメンバーを持たない順データセットも使用できるが出力に対しては順データセットを指定する必要がある。REDUCEはデータセットの形式に対して標準値を持っており、出力に対応するデータセットをその形式に変えてしまう。REDUCEの持つ標準値を次に掲げておく。

		標準値	値の代入されているREDUCEの変数
レコードフォーマット	RECFM	VB	RECFM*
レコード長	LRECL	136	RECL*
ブロック長	BLKSIZE	2724	BLKSIZE*

例えばRECFM*に対してはREDUCE入力モードからRECFM!*;と入力する事によって値が出力される。
次に入出力の手順を図で示す。

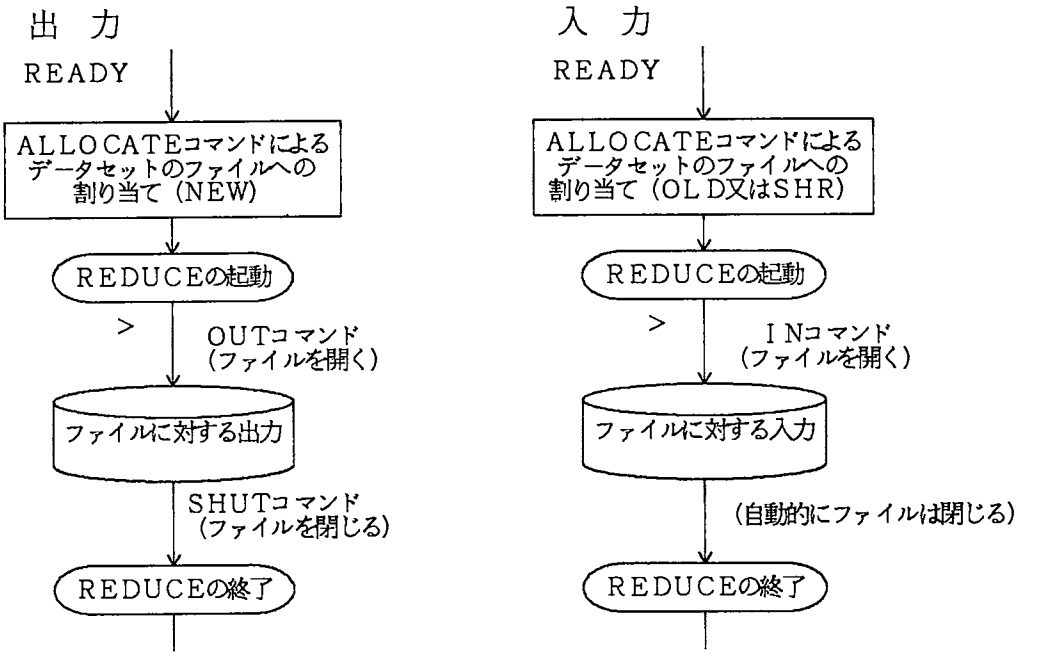

図33

3．1　—ファイルへのデータセットの割り当て—

ＲＥＤＵＣＥでも他の計算機言語と同様にプログラム内では具体的なデータセットをファイル名という形で取り扱う。従ってファイル名と実際のデータセットをあらかじめ対応付けておく必要がある。それにはＴＳＳコマンドの一つであるＡＬＯＣＡＴＥコマンドを用いる。データセット名をＦ．ＤＡＴＡファイル名をＦとした時のデータセットの割り当ての例を示す。パラメタをＮＥＷとしてデータセットＦ．ＤＡＴＡを新規に作成して割り当てた。

```
 READY
ALLOC F(F) DA(F.DATA) NEW  [+E+]
        (ファイル名を「F」として、それを「F. DATA」のデータセットに割り当てた)
LISTC  [+E+]
 IN CATALOG:UCAT01              (新しいデータセットF. DATAができた)
 AB9999.F.DATA
 READY
```

図34

3．2 —ＨＬＩＳＰからのＲＥＤＵＣＥの起動—

センタコマンドＲＲＥＤＵＣＥで起動した時に起動するコンパイルされたＲＥＤＵＣＥでは今の所ファイルからの出し入れはできない。そこでコンパイルされていないＲＥＤＵＣＥの起動方法を示す。

```
READY
HLISP  [+E+]                          (HLISPの起動)
FILE DUMPFILE NOT FREED, IS NOT ALLOCATED
FILE LISP NOT FREED, IS NOT ALLOCATED
FILE LISPFILE NOT FREED, IS NOT ALLOCATED
FILE LISP NOT FREED,IS NOT ALLOCATED
FILE DUMPF NOT FREED, IS NOT ALLOCATED
@REDUCE( )  [+E+]                     (ＲＥＤＵＣＥの初期化)
 GBC L/H/1/2/3 =   65135/  35754/   212/    0/
    =  REDUCE INITIALIZATION HAS COMPLETED !
@BEGIN( )  [+E+]                      (ＲＥＤＵＣＥの起動)
 REDUCE 2 (84/06/22/VERSION APR-24-81) ...
>                                     (ＲＥＤＵＣＥの入力要求状態)
      :
      :
>END;  [+E+]                          (ＲＥＤＵＣＥの終了)
       =  ENTERING LISP ...
@/*  [+E+]                           (ＨＬＩＳＰの終了)
READY
```

図35

センタコマンドＨＬＩＳＰでＨＬＩＳＰを起動し、次にＲＥＤＵＣＥ（ ）でＲＵＤＵＣＥ本体の初期化を行なう。そしてＢＥＧＩＮ（ ）でＲＥＤＵＣＥが起動する。終了方法はＲＲＥＤＵＣＥコマンドの時と同一である。

3．3 —ＦＯＲＴＲＡＮ形式での出力（ＦＯＲＴフラグ）—

ＲＥＤＵＣＥの結果をデータセットに出力したい理由の一つには、それをＦＯＲＴＲＡＮ等の数値計算言語のプログラムとして直接使いたいという事がある。そのためにＲＥＤＵＣＥは数式をＦＯＲ

ＴＲＡＮプログラム形式で出力することを宣言するＦＯＲＴフラグを用意している。また使用するＦＯＲＴＲＡＮによっては継続行数に制限があるので、それに対応するためにその設定が変更できるようになっている。その値は＊ＣＡＲＤＮＯという変数に入っている。

3．4　—ファイル出力のためのＲＥＤＵＣＥコマンド（ＯＵＴ）—

形式　　ＯＵＴ　〔ファイル名〕

このコマンドの入力によってファイル名に対応したデータセットに対して結果の出力が開始される。

3．5　—ＯＵＴコマンドのファイルオープンに対してファイルを閉じるＲＥＤＵＣＥコマンド（ＳＨＵＴ）—

形式　　ＳＨＵＴ　〔ファイル名〕

ＯＵＴコマンドでファイル出力が開始され、このＳＨＵＴコマンドによってファイルが閉じて出力は終了する。ＳＨＵＴコマンドが入力されると空文が一行出力されてファイルは閉じる。

```
>ON FORT; [+E+]                    (FORTRAN形式の出力を指定)
>(A+1)**2; [+E+]
      ANS=A**2+2*A+1
                    (FORTRAN形式では常に代入された形で出力される)
>!*CARDNO; [+E+]
                    (使用するFORTRANによっては継続行の長さに制限がある)
      ANS=20        (制限値は「＊CARDNO」という変数に入っている)
>OUT F; [+E+]                      (ファイル「F」に出力を指定する)
>WRITE "C FORTRAN PROGRAM" [+E+]
   (REDUCEで出力できないFORTRAN特有の部分はWRITE文で出力しておく)
>WRITE "      READ(5,*) A"; [+E+]
>Y:=(A+1)**10; [+E+]
>WRITE "      WRITE(6,*)"; [+E+]
>WRITE "      STOP"; [+E+]
>WRITE "      END"; [+E+]
>SHUT F; [+E+]                     (ファイル「F」を閉じる)
```

図36

次に示したのが出力されたデータセットをＬＩＳＴした様子である。

```
 READY
LIST F.DATA NONUM   [+E+]
  AB9999.F.DATA
       ANS=11      (ファイルが開かれた事に対する出力である。あとで除けばよい)
C FORTRAN PROGRAM
       READ(5,*) A
       Y=A**10+10*A**9+45*A**8+120*A**7+210*A**6
      X+252*A**5+210*A**4+120*A
      X**3+45*A**2+10*A+1
                              (先頭の「X」は継続行を表わすための「X」である)
       WRITE(6,*) Y
       STOP
       END
                              (ＳＨＵＴによってスペースが１行出力されている)
  END OF DATA
 READY
```

図37

3．6　—ＲＥＤＵＣＥが再入力可能な形での出力（ＮＡＴフラグ）—

結果のデータセット出力の理由は計算結果の保存と再利用である。ＮＡＴフラグは通常ＯＮになっていて、これまで見て来たような形でＲＥＤＵＣＥは結果を出力する。ＮＡＴフラグをＯＦＦにする事で指数をＦＯＲＴＲＡＮ形にしたり、出力に区切子が付けられるなどの処理が行なわれて、結果は再入力可能な形で出力される。この機能を利用してファイル出力を行えば再入力可能な形でデータセットに結果が保存できる。

次に示したのが再入力可能な形でのファイル出力の例である。ファイルのデータセットへの割当てやＲＥＤＵＣＥの起動方法は、ＦＯＲＴＲＡＮ形式での出力の場合と同一である。

```
>OFF NAT;  [+E+]                    (再入力可能な形で出力する事を指定する)
>(A+1)**2;  [+E+]
 A**2 + 2*A + 1¥ (最後に「¥」(実行文の区切り子 ) の付いた形で出力される)
>OUT F;  [+E+]                      (ファイルＦを開く)
```

```
>X:=(A+1)**2;    [+E+]
>Y:=SUB(A=A+1,X);    [+E+]
>Z:=Y-X;  [+E+]
>WRITE"END";    [+E+]
                          (「END」を入力したいファイルの最後には必ず入れておく)
>SHUT F;    [+E+]                    (ファイルFを閉じる)
```

図38

次に出力されたデータセットをＬＩＳＴしてみた様子を示す。

```
 READY
LIST F.DATA NONUM    [+E+]
 AB9999.F.DATA
11¥
X := A**2 + 2*A + 1¥
Y := A**2 + 4*A + 4¥
Z := 2*A + 3¥
END¥                              (最後にはＥＮＤコマンドが入れてある)
 END OF DATA                      (入力時の為に空行を一行入れておく)
READY
```

図39

3．7 ―出力結果の再入力（ＩＮ）―

3．6の方法で出力されたデータセットからは再入力が可能である。データセットへの出力の時と同様な方法でファイルをデータセットに割り当て、そしてＲＥＤＵＣＥをＨＬＩＳＰ上から起動する。ファイルからの入力はＩＮコマンドによってする。

形式　　ＩＮ　〔ファイル名〕

入力が終るとファイルは自動的に閉じられる。3．6の例の中に示した様に入力ファイルの最後にはＥＮＤコマンドが入れてあって、さらに最後に空行が一行入れてある必要がある。

```
>IN F;    [+E+]                   (ファイル「Ｆ」からの入力指定)
 11¥                               (入力に対してエコーバックされる)
```

```
X:=A**2+2*A+1¥
Y:=A**2+4*A+4¥
Z:=2*A+3¥
END¥                                    (ファイルは自動的に開じられる)
>X;Y;Z;   [+E+]                         (X, Y, Zには値が代入された)
 2
A  + 2*A + 1
 2
A  + 4*A + 4
2*A + 3
```

図40

第4章　記号式モードと代数式モード

第3章までに述べて来たものはＲＥＤＵＣＥの本来の機能である。代数式はＲＥＤＵＣＥ内部ではＬＩＳＰのデータ構造であるリストに直されて処理される。この章ではＲＥＤＵＣＥを特殊な目的で利用しようとする人達、そして数式処理やＬＩＳＰに関心を持つ人達にＲＥＤＵＣＥのデータ構造を示す事を目的とする。これだけ巨大なＬＩＳＰプログラムはなかなかお目に掛かれないので、ＬＩＳＰを学ぶ上でも格好の研究材料となるだろう。

4．1　—ＲＥＤＵＣＥ入力式のＬＩＳＰへの翻訳（ＤＥＦＮフラグ）—

ＲＥＤＵＣＥは入力式をＬＩＳＰプログラムの形や後に述べるＲＥＤＵＣＥ特有のデータ構造であるカノニカル表現に直してから処理をする。
ＤＥＦＮフラグをＯＮにすると、入力された数式や手続きがＲＥＤＵＣＥが評価するＬＩＳＰプログラムの形で出力される。この時入力された式自体の評価は行われない。

```
>ON DEFN;   [+E+]
>(X+Y)**2;   [+E+]
 EVAL(( AEVAL ( LIST ( QUOTE EXPT ) ( LIST( QUOTE PLUS ) X Y ) 2 ) ))
>OFF DEFN;   [+E+]
```

図41

4．2　—記号式モードと代数式モード—

ＲＥＤＵＣＥには代数式モード（ＡＬＧＥＢＲＡＩＣ　ＭＯＤＥ）と記号式モード（ＳＹＭＢＯＬＩＣ　ＭＯＤＥ、ＬＩＳＰ　ＭＯＤＥ）の二つの処理モードがある。どのモードで処理しているかは

　　！＊ＭＯＤＥ；

と入力する事によって表示される。この入力に対して

　　代数式モードならば　　ＡＬＧＥＢＲＡＩＣ

　　記号式モードならば　　ＳＹＭＢＯＬＩＣ

と表示される。起動時には代数式モードになっている。

4．2．1　　代数式モードから記号式モードへ

代数式モードから記号式モードへは次の様に入力する事によって移動できる。

ＬＩＳＰ；　　又は　ＳＹＭＢＯＬＩＣ；

4．2．2　　記号式モードから代数式モードへ

記号式モードから代数式モードへの移動は次の様にして行なう。

　　ＡＬＧＥＢＲＡＩＣ；

4．3　—記号式モード—

4．3．1　　記号式モードでの演算子

記号式モードでは通常のＬＩＳＰ関数はＰＲＥＦＩＸ　ＯＰＥＲＡＴＯＲの形をとるが次の二つが二項演算子になっている。

　　ＥＱ

　　．（ＬＩＳＰのＣＯＮＳ）

記号式はＬＩＳＰのメタ言語（Ｍ式）に従う。（文献［4］、［5］等を参照）

4．3．2　　引用式（記号式モード）

ＬＩＳＰのＱＵＯＴＥ関数に対応するものとして′を使う。文字列は自動的に引用されるので′は必要がない。

4．3．3　　記号式の設定

記号式モードでは、もし設定文　（：＝）の右辺が変数であれば右辺の左辺に対するＳＥＴＱとみ

なされ右辺がＬＡＭＢＤＡ式であれば関数の定義とみなされる。

4．3．3．1　ＬＡＭＢＤＡ式（記号式モード）

ＬＡＭＢＤＡ式を使用する事でＬＩＳＰのＬＡＭＢＤＡ式を構成できる。

形式　　LAMBDA　（〔変数名〕，……，〔変数名〕）；

〔M式によるＬＩＳＰプログラム〕；

```
>LISP;  [+E+]                                    (記号式モードに入る)
 NIL
>B:='(LAMBDA (X) (COND((EQ X 0) 1) (T (TIMES X (B (SUB1 X))))));
[+E+]                                            (階乗を計算する関数)
(LAMBDA (X) (COND ( (EQ X 0) 1) (T ( TIMES X ( B (SUB1 X) )
 ) ) ) )      (右辺がＬＡＭＢＤＡ式であれば記号式モードでは関数の定義になる)
>B(10); [+E+]
3628800                                          (１０の階乗の計算結果)
>OPERATOR B;  [+E+]
 (記号式モードでＯＰＥＲＡＴＯＲ宣言をして定義した関数を代数式モードでも使えるようにする)
 NIL
>ALGEBRAIC;  [+E+]                              (代数式モードに戻る)
 >B(20);  [+E+]
 2432902008176640000
>SYMBOLIC; [+E+]
 NIL
>!*MODE;  [+E+]
 SYMBLIC
```

図42

4．3．3．2　ＳＹＭＢＯＬＩＣ　ＰＲＯＣＥＤＵＲＥ（記号式手続き）

２．９．２で述べた手続き関数の手法も記号式の手続きを定義するのに用いる事ができる。関数属性はＳＹＭＢＯＬＩＣである。

```
>LISP;  [+E+]
```

```
NIL                    (ＬＩＳＰの関数ＡＳＳＯＣをＰＲＯＣＥＤＵＲＥ文で定義してみる)
>SYMBOLIC PROCEDURE ASS(U,V);  [+E+]
                                        (手続きのタイプはＳＹＭＢＯＬＩＣ)
>IF NULL(V) THEN NIL ELSE IF U EQ CAAR(V) THEN CAR(V)ELSE
 ASS(U,CDR(V));  [+E+]
 ASS
>ASS;  [+E+]
 (LAMBDA (U V) (COND ((NULL V) NIL) ((EQ U (CAAR V))(CAR V))
 (T (ASSU (CDR V)))))
>ASS('B,'((A.AA)(B.BB)(C.CC)));  [+E+]
 ( B . BB )
```

図43

4．4　―記号式モードと代数式モードとの間の値のやりとり―

記号式モードで定義した関数を代数式モードでも使用できるようにするためには、ＳＹＭＢＯＬＩＣモードでＯＰＥＲＡＴＯＲ宣言すればよい。

形式　　OPERATOR　〔関数名〕，……，〔関数名〕

4．4．1　二つのモード間での変数の共有（ＳＨＡＲＥ宣言）

代数式モードの変数と記号式モードの変数とは、別のものとして扱われるので、二つのモード間で値の受け渡しを行うためには共有される変数を宣言する必要がある。

形式　　SHARE〔変数名〕，……，〔変数名〕

4．4．2　ＲＥＤＵＣＥの内部表現

変数をＳＹＭＢＯＬＩＣモードで評価すると、その値として、その内部表現のリストが返される。ＳＣＡＬＡＲ属性を持った変数では一般に次の形を持っている。

(＊ＳＱ〔ＳＴＡＮＤＡＲＤ　ＱＵＯＴＩＥＮＴ〕Ｔ)

〔ＳＴＡＮＤＡＲＤ　ＱＵＯＴＩＥＮＴ〕の部分がカノニカル形式（ＣＡＮＯＮＩＣＡＬ　ＦＯＲＭ）と呼ばれる部分でＲＥＤＵＣＥでの有理式がここに置かれている。

ＣＡＮＯＮＩＣＡＬ　ＦＯＲＭ

〔ＳＴＡＮＤＡＲＤ　ＱＵＯＴＩＥＮＴ〕（有理式）

（〔分子〕〔分母項〕……〔分母項〕）

〔STANDARD FORM〕(分子)

(〔項〕……〔項〕)

〔STANDARD TERM〕(項)

(〔べき〕.〔係数〕)

〔STANDARD POWER〕(べき)

(〔主変数〕. 〔次数〕)

例として(A+B)／(C+D)は

(*SQ(((A. 1). 1) ((B. 1). 1)) ((C. 1). 1) ((D. 1). 1)) T)

となる。代数式モードでの式をこの様なカノニカル形式にしたり、ＬＩＳＴ形式に変換する関数がＲＥＤＵＣＥ内部に存在するので、リスト形式になった数式をＬＩＳＰ関数で処理したりＲＥＤＵＣＥ本体の演算関数で演算させたりして代数式モードに返せば、単に代数式モードで行なったのよりも複雑な変換が可能になる。

```
>SHARE X;  [+E+]                    (変数Xを記号式モードで共有する事を宣言する)
>X:=(A**2+A+C)/(B**2+B+D);  [+E+]
        2             2
 X := (A  + A + C)/(B  + B + D)
>LISP;  [+E+]
 NIL
>X;  [+E+]
 (*SQ(((A.2).1)((A.1).1)((C.1).1))
((B.2).1 ((B.1).1)(D.1).1))T)
>X:=SIMP!*(X);  [+E+]
                    (有理式の本体をカノニカル形式で取り出す。評価もこの時受ける)
  (((A.2).1)((A.1).1)((C.1).1))
  ((B.2).1)((B.1).1)((D.1).1))
>X:=PREPSQ(X);  [+E+]                              (ＬＩＳＰ形式に直す)
  (QUOTIENT (PLUS (EXPT A 2) A C) (PLUS (EXPT B 2) B D))
>X:=CONS(CAR(X),LIST(CADDR(X),CADR(X)),  [+E+]
                    (ＬＩＳＰ形式になった式をＬＩＳＰ関数を使って加工する)
  (QUOTIENT (PLUS (EXPT B 2) B D) (PLUS (EXPT A 2) A C))
>X:=SIMP!*(X);  [+E+]                              (カノニカル形式にもどす)
  (((B.2).1)((B.1).1)((D.1).1))
```

```
 ( ( A . 2 ) . 1 ) ( ( A . 1 ) . 1 ) ( ( C . 1 ) . 1 ) )
>X:=MK!*SQ(X);  [+E+]                    (有理式を表わす　*SQ　などを付ける)
 ( *SQ ( ( ( ( B . 2 ) . 1 ) ( ( B . 1 ) . 1 ) ( ( D . 1 ) . 1 ) )
 ( ( A . 2 ) . 1 ) ( ( A . 1 ) . 1 ) ( ( C . 1 ) . 1 ) ) T )
>ALGEBRAIC;  [+E+]                                  (代数式モードへ)
 >X;  [+E+]
   2             2
 (B  + B + D)/(A  + A + C)
```

図44

G. C. D. とL. C. M. を求める内部関数　ＧＣＤＦとＬＣＭを用いてG. C. D. , L. C. M. を求める為の関数ＧＧＧとＬＬＬを構成した例を示す。

```
 >!*MODE;  [+E+]
 ALGEBRAIC
>SYMBOLIC PROCEDURE GGG(X,Y);  [+E+]
>BEGIN;  [+E+]
>SCALAR U,V;  [+E+]
>U:=CAR(SIMP!*(X));V:=CAR(SIMP!*(Y));  [+E+]
                                        (U、VにX、Yの分子を取り出す)
 >U:=GCDF(U,V);  [+E+]                  (UとVのＧＣＤをUへ代入)
 >U:=MK!*SQ(CONS(U,1));  [+E+]                  (有理式の形にする)
 >RETURN U; END;  [+E+]
 GGG
>SYMBOLIC PROCEDURE LLL(X,Y);  [+E+]
>BEGIN;  [+E+]
>SCALAR U,V;  [+E+]
>U:=CAR(SIMP!*(X));V:=CAR(SIMP!*(Y));  [+E+]
>U:=LCM(U,V);  [+E+]
>U:=MK!*SQ(CONS(U,1));  [+E+]
>RETURN U;  [+E+]
>END;  [+E+]
 LLL
```

```
>LISP;  [+E+]
 NIL
>OPERATOR GGG,LLL;  [+E+]
 NIL
>ALGEBRAIC;  [+E+]
>X:=(A+B)*(C+D);  [+E+]
 X :=A*C + A*D + B*C + B*D
>Y :=(C+D)*(E+F);  [+E+]
 Y := C*E + C*F + D*E + D*F
>GGG(X,Y);  [+E+]
C + D
>GGG(6,15);  [+E+]
 3
>LLL(X,Y);  [+E+]
 A*C*E + A*C*F + A*D*E + A*D*F + B*C*E + B*C*F + B*D*E
 + B*D*F
```

↑

(属性がＳＹＭＢＯＬＩＣの手続き関数は、記号式モードでＯＰＥＲＡＴＯＲ宣言↓ しておく事が必要である)

図45

ＳＩＭＰ＊，ＭＫ＊ＳＱ，ＰＲＥＰＳＱ，ＧＣＤＦ，ＬＣＭ等の関数はＲＥＤＵＣＥ本体の内部関数である。Ｇ．Ｃ．Ｄ．やＬ．Ｃ．Ｍ．を求める機能はＲＥＤＵＣＥ２の代数式モードではコマンドとして用意されていないので直接利用する事ができない。しかし今述べた方法を用いれば利用できるようになる。

４．５ —ＲＥＤＵＣＥの内部関数の表示方法—

ＲＥＤＵＣＥの内部関数は「RREDUCE」コマンドで起動した場合にはコンパイルされてしまうので、参照するのは不可能だが、３章で述べた方法で起動した場合にはそれが可能である。

内部関数名は次の様にすれば表示される。

```
@OBLIST ( )  [+E+]
        = ( LER3 ....
```

関数の内容は次の様にすれば表示される。

@EVAL([関数名] NIL) [+E+]

あるいはＲＥＤＵＣＥの記号式モードから

```
>LISP;  [+E+]
 NIL
>GCDF;  [+E+]
 (LAMBDA (U V) (ABSF (GCDF1 U V)))
>LCM;  [+E+]
 (LAMDA (U V) (COND ((OR (NULL U) (NULL V)) NIL) (ONEP U V)
 ((ONEP V) U) (T (MULTF U (QUOTF V (GCDF U V))))))
```

図46

ここでＧＣＤＦ１は４．４．２で述べたカノニカル表現のＳＴＡＮＤＡＲＤ　ＦＯＲＭで書かれた数式に対して、ユークリッドの互除法を用いてＧ．Ｃ．Ｍを求める関数であり、ＭＵＬＴＦ、ＱＵＯＴＦは積及び商をＯＮＥＰは１かどうかを判定する関数である。ＲＥＤＵＣＥ内部にはこの他にもカノニカル表現の各レベルに対応した処理関数が存在する。

第５章　バッチによる利用法

ＢＡＴＣＨ処理でもＲＥＤＵＣＥを利用できる。ＣＮＴＬ属性のデータセットに次の様な形式でジョブストリームを作成し、サブミット（ＳＵＢＭＩＴ）する。

```
//    ジョブ文
//EXEC REDUCE
//SYSIN DD *
        :
         REDUCE  プログラム
        :
/*
//
```

ＲＥＤＵＣＥプログラムはデータセットからの入力に用いたものと同形式のものを使う。例えば、課

題番号がＡＢ９９９９でパスワードがＰＡＳＳＷＯＲＤの時は次の様になる。

```
//AB9999   JOB REDUCE,PASS=PASSWORD,CLASS=B,REGION=1024K,NOTIFY=AB9999
//  EXEC REDUCE,TYPE=C
//SYSIN  DD *
(X+Y)**2;
END;
/*
//
```

図47

データセット　ＲＥＤＵＣＥ．ＤＡＴＡ（ＰＲＯＧＲＡＭ）にＲＥＤＵＣＥプログラムが入っている時には次の様になる。

```
//AB9999   JOB REDUCE,PASS=PASSWORD,CLASS=B,REGION=1024K,NOTIFY=AB9999
//  EXEC REDUCE,TYPE=C
//SYSIN  DD DSN=AB9999.REDUCE.DATA(PROGRAM),DISP=SHR
//
```

－　ＢＡＴＣＨ処理の実行例　－

```
LISTC  [+E+]
IN CATALOG:UCAT01
AB9999.BATCHR.CNTL          (次に続いてリストしてある２つのデータセットを用意する)
AB9999.REDUCE.DATA
READY
LIST REDUCE.DATA(PROGRAM)  [+E+]
 AB9999.REDUCE.DATA(PROGRAM)
10 (X+Y)**2;                              (ＲＥＤＵＣＥのプログラムが入っている)
20 END;
END OF DATA
LIST BATCHR.CNTL  [+E+]
00010 //AB9999 JOB REDUCE,PASSWORD,CLASS=A,REGION=1024K,NOTIFY=AB9999
00020 //  EXEC REDUCE,TYPE=C
```

```
00030 //SYSIN DD DSN=AB9999.REDUCE.DATA(PROGRAM),DISP=SHR
00040 //
 END OF DATA                                          (ジョブストリームが入っている)
READY
SUB BATCHR.CNTL  [+E+]
ENTER JOBNAME CHARACTER(S) -
B  [+E+]                                              (任意の一文字を入力する)
  ENQC00*AB9999B  ACCEPTED CLASS=A,IN/JOB=AB9999
 JOB AB9999B(JOB00690) SUBMITTED
READY
ST  [+E+]                                             (ジョブの状況の確認)
 JOB AB9999B(JOB00690) EXECUTING                      (実行中)
READY
 18.47.38 JOB  690 JEM1651  AB9999B  ENDED AT N1CN(01)
                                                      (ジョブ終了メッセージ)
OUTPUT AB9999B PRINT(K)     (ジョブ結果をデータセットK. OUTLISTに出力する)
READY
LISTC  [+E+]
IN CATALOG:UCAT01
AB9999.BATCHR.CNTL
AB9999.K.OUTLIST
AB9999.REDUCE.DATA
READY
```

図48

— ジョブ結果の出力例 —

```
READY
LIST K.OUTLIST NONUM  [+E+]          (データセットK. OUTLISTをリストする)
AB9999.K.OUTLIST
1                    JES  JOB LOG         --
 SYSTEM 170F -- NODE N1
-
 18.47.14 JOB 690 JEM373I AB9999B STARTED - INIT  4 - CLASS A - SYS 170F
 18.47.14 JOB 690 INTJ01I AB9999B STARTED  CLASS=A,AIFNO=50995
```

```
5
 18.47.14 JOB 690 JDJ403I AB9999B - STARTED - TIME=18.47.14
 18.47.37 JOB 690 JDJ404I AB9999B - ENDED - TIME=18.47.37
 18.47.38 JOB 690 ACTR001 AB9999B ENDED   COND=0000,AIFNO=50995
 18.47.38 JOB 690 JEM395I AB9999B ENDED

              E40   V12L02                        <<<   JCL STATEMENT

    1     //AB9999B JOB REDUCE,ADDRSPC=VIRT,CLASS=A,REGION=1024K,NOTIFY=
    2     //  EXEC REDUCE,TYPE=C
   13     //SYSIN    DD DSN=AB9999,REDUCE.DATA(PROGRAM),DISP=SHR

                                                 <<<   SYSTEM MESSAGES

 JDJ142I AB9999B RUN - STEP WAS EXECTED - COND CODE 0000
 JDJ373I STEP /RUN     / START 84097.1847
 JDJ374I STEP /RUN     / STOP  84097.1847 CPU   0MIN 03.82SEC SRB
 JDJ375I  JOB /AB9999B / START 84097.1847
 JDJ376I  JOB /AB9999B / STOP  84097.1847 CPU   0MIN 03.82SEC SRB
1AB9999B        HLISP INTERPRETER-COMPILER M-200/M-190 (VERSION DEC/01/
0
     GBC L/H/1/2/3 =   49999/  27925/  137/    0/    0
        =  DUMPFILE
     EVAL  T/E/I/O/G =     581/    306/ 138/   0/  137
        =  NIL
     EVAL  T/E/I/O/G =    3302/   2719/   2/   0/    0
        =  REDUCE INITIALIZATION HAS COMPLETED!
     EVAL  T/E/I/O/G =    3397/     93/   1/   1/    0

1AB9999B        HLISP INTERPRETER-COMPILER M-200/M-190 (VERSION DEC/01
0
            REDUCE 2 (84/04/06 / VERSION APR-24-81) ...
           ON TIME;
         P (STEP/TOTAL) TIME = (77/78) MS
           (X+Y)**2;
             2           2
            X  + 2*X*Y + Y                          (実行結果)
         P (STEP/TOTAL) TIME = (61/139) MS

 END OF DATA
READY
```

図49

付 録

東京大学大型計算機センターニュースのものをそのまま転載する。

1. 使用上制限のある名前

REDUCEにおける予約語の一覧表を以下に掲げる。ここにあるのは予約語とコマンド名、演算子であり、高エネルギー物理学（説明略）の計算に使用する予約語も含めてある。

予約語	BEGIN DO ELSE END FOR FUNCTION GO
	GOTO LAMBDA NIL PRODUCT RETURN STEP
	SUM TO WHILE
	予約スカラー変数
	E I
infix演算子	:= = >= > <= < + − * / **
	SETQ AND NOT OR MEMBER EQUAL UNEQ
	EQ GEQ GREATERP LEQ LESSP PLUS MINUS
	TIMES QUOTIENT EXPT CONS
prefix演算子	ARB COEFF CONS DEN DET DF EPS G LOG
	MAT NUM SIN SUBTRACE
コマンド	ALGEBRAIC ARRAY CLEAR COMMENT END
	FACTOR FOR FORALL GO GOTO IF IN
	INTEGER LET LISP MASS MATCH MATRIX
	HELL NOSPUR OFF ON OPERATOR ORDER
	OUT PROCEDURE PUNCH REAL REMARRAY
	RETURN SAVEAS SCALAR SHUT SPUR
	SYMBOLIC VECTOR WEIGHT WRITE WTLEVEL

2. REDUCEで使用可能なコマンド

表記法について。E、E1、……Enは式を、V、V1、……Vnは変数を意味している。

ALGEBRAIC E;	もしEがなければシステムモードを代数式モードにする。もしEがあれば代数式モードの下でEを評価するが、モードは変化しない。
ARRAY V1〈size〉, … …, Vn〈size〉;	V1からVnまでの変数を配列名として宣言する。〈size〉によって各配列の最大次数を指定する。
CLEAR E1, …, En	E1からEnまでの式に対して宣言された値、代入則等をシス

テムから消去する。

COMMENT〈any〉; プログラム中に注釈を含める為に使用する。〈any〉はterminatorを含まないならばどんな文字列から成っていてもよい。

CONT; 対話用のコマンドであり、入力ファイルの中の一番最近に出会ったPAUSEが現われた所から計算を再開させる。

END〈any〉; REDUCEへのファイルからの入力を終える。〈any〉はterminatorあるいは、予約語END、ELSE、UNTILを含まないどのような文字列であってもよい。

FACTOR E1, …, En; 出力の際に共通因子はくくり出すように宣言する。

種々のプログラムループを定義する為のコマンド。

FORALL V1, …, Vn 〈command〉 V1からVnまでの変数を、〈command〉で示される代入則の中では、任意であることを宣言する。

GOTO V; Vというラベルへ無条件に制御を移す。これは複合文の中でのみ使用することができる。

IF 条件文を定義するのに使用される。

IN V1, …, Vn; V1からVnまでのファイルから順にプログラム、又はデータを読み込む。

INTEGER V1, …, Vn; V1からVnまでの変数を整数型変数として宣言する。

LET E1, …, En; E1からEnまでの式の左辺の式に対する代入則を指定する。

その他に、微分公式を定義することも可能である。

LISP E; もしEがなければ、処理モードを記号式モードにする。Eがあれば記号式モードの下でEを評価し、処理モードは変えないでおく。

MATCH E1, …, En; E1からEnまでの式の左辺の式と、べき指数が完全に合った時だけ有効な代入則を指定する。

システムに行列変数を宣言する。各Eiは行列変数名であっても、又行列の大きさを陽に含んでいてもよい。

OFF V1, …, Vn; V1からVnまでのフラグをoffにする。

ONV1, …, Vn; V1からVnまでのフラグをonにする。

OPERATOR V1, …, Vn; V1からVnまでを代数演算子として宣言する。

ORDER V1, …, Vn; V1からVnまでの変数に対する、出力における並びの順序を設定する。

OUT V;	Vを今後出力用ファイルとして使用する旨宣言する。
PAUSE;	入力用ファイルに入れる会話用のコマンド。このコマンドを評し終わると制御は端末に渡される。
PROCEDURE	計算をくり返して行なう為に使用する文に名前をつける為に使用する。手続きの型宣言はコマンド名の前になければならない。
REAL V1, …, Vn;	V1からVnまでの変数を実数型変数として使用されるものとして宣言する。
RETURN E;	複合文の実行を終了し、次のプログラムの実行を始める。Eの値は複合文の値として返されるが、Eはなくてもよい。
SAVEAS E;	Eに、保存用領域中にとってある、現在の式を設定する。
SCALAR V1, …, Vn;	V1からVnまでの変数をスカラー型変数として使用されるものとして宣言する。
SHUT V;	出力ファイルVを閉じる。
SYMBOLIC E;	LISP E;と同じ。
WEIGHT E1, …, En;	E1からEnまでの式の左辺の式に重みを設定する。
WRITE E1, …, En;	E1からEnまでの値を現在の出力ファイルに出力する。
WTLEVEL V;	システムで使用する重みの制限値をVに設定する。

3. REDUCEにおけるモードフラグ

この節ではON又はOFFコマンドの引数として書けるフラグについて説明してある。特に断わらない限りフラグがonの時の働きが説明してある。

ALLFAC	式を出力する際に共通の積をくくり出してしまうようにする。
DEFN	REDUCE入力と等価なLISP式を出力させる。その時、LISP式の評価は行なわない。
DIV	出力時に、負のべき乗や有理分数式が出力されるように、分母の各項をを分子で割った形で出力する。
ECHO	入力式を確認する為のおうむ返しを行なう。
EXP	式を評価する時に展開することを示す。
FLOAT	式の評価の途中で浮動小数点を二つの整数の比に変換することを禁止する。
FORT	FORTRANスタイルの出力とすることを宣言する。
GCD	有理式の分子分母のGCDを消去する。
INT	対話的に処理を行なうようにする。

LIST　　一つの項を一行に出力するようにする。

MCD　　二つの式が足された時に分母を共通にする。

MSG　　診断メッセージを出力する。

NAT　　出力を自然な形で出力する。

NERO　　0を設定する式の出力を仰止する。

PRI　　見やすい形の出力にする。

RAT　　FACTOR宣言と一緒に使用する出力制御用フラグ。式の分母が各項に現われるようにする。

RESUBS　　RESUBSがoffの場合、一度代入則が使用されたならば、代入を行なわれた式の中に代入則が使用できる式があったとしてももう代入を行なわない。

TIME　　計算時間を出力する。

4.　診断メッセージ及びエラーメッセージ

REDUCEシステムの出力する診断メッセージは、エラーメッセージと警告メッセージの2つの種類がある。エラーメッセージが出力された時は現在行なっている計算を終了してしまうが、警告メッセージが出力された場合は使用者に、不明確な所があることを教えるか又はエラーであることを示すように動作する。意味が不明確な場合にも、もし会話的に使用していたならば、正しい入力を要求してくる。それ以外の場合には、もっともありそうな場合を仮定しその旨メッセージを出力して、処理を続行する。

式あるいはプログラムの入力中に、構文解析上のエラーが生じたならば、どこでエラーが見つかったかを示す¥¥¥を挿入して、入力式をもう一度くり返す。

この付録には、高エネルギー物理学上の計算を実行する場合に生ずるエラーについての説明も含まれている。

4．1　エラーメッセージ

A REPRESENTS ONLY GAMMA 5 IN VECTOR EXPRESSIONS

Aはベクトル式の中では γ_5 を意味するものとしてのみ使用可能である。

ARRAY TOO SMALL

COEFFコマンド中の第三引数にきている配列の添字は、第一引数の中に第二引数に関する最高次の次数よりも小さい。

CATASTROPHIC ERROR

このエラーが発生したら、入出力リストを、東大大型センター金田までお送り下さい。

DIFFERENTIATION WRT〈expressin〉NOT ALLOWED

〈expressin〉に関する微分は許されない。

DOT CONTEXT ERROR

記号式モード中で、引用された式はＬＩＳＰの文法に合わない。

ELEMENTS OUT OF BOUNDS

配列を引用している添字の大きさが、おかしい。

INCORRECT ARRAY ARGUMENTS FOR〈name〉

〈name〉を持つ配列又は行列を、非数値の添字で引用しようとした。

LARGER SYSTEM NEEDED

現在のシステムで、与えられた問題を解くことはできない。

LOCAL VARIABLE USED AS OPERATOR

局所変数が演算子としての位置で使用されている。

MATRIX MISMATCH

与えられた二つの式の次数の大きさが合わない為に、和又は積の演算を行なうことができない。

MATRIX〈name〉NOT SET

行列の要素に値が設定されていないのに使用しようとした。

MISMATCH OF ARGUMENTS

評価手順が定義された演算子の仮引数と実引数の数が合わない。

MISSING ARGUMENTS FOR G OPERATOR

γ－マトリックスの中で、入出力線を区別する記号がない。

MISSING OPERATOR

入力エラー。

MISSING VECTOR

ベクトルがくるべき所にベクトルがきていない。

NON－NUMERICAL ARGUMENT IN〈expression〉

配列要素又は行列要素が非数値的な式の中から参照されている。

NON SQUARE MATRIX

非正方行列に対しての不当な演算である。（例　対角和）

NOT YET IMPLEMENTED

入出力データを同封して、東大大型センター金田までお送り下さい。

OPERATOR〈name〉CANNOT BE ARBITRARY

代入則の定義において、演算子を任意なものとして指定することができない。

REDUNDANT OPERATOR

入力エラー。

REDUNDANT VECTOR

ベクトル式の中に冗長なベクトルがある。

SINGULAR MATRIX

行列式が0である行列の逆行列を求めようとした。

SUBSTITUTION FOR 〈expression〉 NOT ALLOWED

代入則の定義中に不当な式がある。

SYNTAX ERROR

入力式の中に文法上のエラーがある。エラーの原因が不明な場合に出力される。

TOO FEW RIGHT PARENTHESES

入力エラー。右カッコが足りない。

TOO MANY RIGHT PARENTHSES

入力エラー。左カッコが多すぎる。

TYPE CONFLICT FOR 〈expression〉

〈expression〉は不当な型である。

UNMATCHED INDEX ERROR 〈list〉

ベクトル式の評価中に、添字が合わなかった。

ZERO DENOMINATOR

分母が0となっている。

0／0 FORMED

0／0の式は取り扱えない。

〈expression〉 CANNOT BE AN OPERATOR

演算子の所にきている名前がおかしい。

〈expression〉 INVALID PROCEDURE NAME

手続き文に不当な名前が付けてある。

〈expression〉 NOT FOUND

システムの中にあるべき〈expression〉（例えば、CLEAR文中で）が宣言、又は登録されていない。

〈file name〉 NOT OPEN

OPENされていないファイル、又はSHUTされたファイルに対してSHUTコマンドを実行しようとした。

〈identifier〉 UNDEFINED

〈identifier〉はコマンドとして登録されていない。

〈number〉 TOO LONG FOR FORTRAN

FORTRAN用出力を行なっている時に、長い整数を出力しようとした。

〈type〉 〈variable〉 USED AS SCALAR

〈type〉の型をした変数がスカラーとして使用される所で使用されている。

〈variable〉 ALREADY DEFINED AS 〈type〉

すでに別の型宣言がなされている。

〈variable〉 INVALID又は〈variable〉 INVALID IN 〈statement〉 STATEMENT

文の構文がおかしい。

〈variable〉 HAS NO MASS

MSHELL中の変数〈variable〉には質量が設定されていない。

4. 2 診断メッセージ

ASSIGNMENT FOR 〈expression〉 REDEFINED

〈expression〉は新しい値に再設定された。

COEFF GIVEN EXPRESSION WITH DENOMINATOR 〈exp〉

COEFFの引数として、核の関数である分母を持った式がきたが、そのべきも考慮にいれた。

MISSING END IN FILE 〈name〉

〈name〉ファイルの最後にEND文がなかった。

〈expression〉 REPRESENTED BY 〈expression〉

第二の〈expression〉で最初の〈expression〉を表わした。

〈name〉 REDEFINED

演算子又は手続名〈name〉が再定義された。

〈variable〉は〈type〉型宣言を受けている。

参考文献

○マニュアルを書くに当って参考としたもの

[1] REDUCE USER'S MANUAL VERSION 3.0 ANTHONY C.HEAN
THE RAND CORPORATION SANTA MONICA,CA90406
(REDUCE 3に関するマニュアル. 東京大学大型計算機センターから入手できる. 英文)

[2] 東京大学大型計算機センターニュース VOL.14 NO.3 ～ NO.7 1982

［3］ 数理科学 8 1983 数式処理 サイエンス社（具体的な応用例が記載されている。本文中では全く記載しなかった高エネルギー物理学への応用例も書かれている。）

○ＬＩＳＰに関するもの

［4］ ・コンピュータサイエンス大学講座ＬＩＳＰ入門―システムとプログラミング―
中西正和 近代科学社
・情報処理シリーズ4 ＬＩＳＰ
P.H. ウィンストン,B.K.P. ホーン著／白井良明・安部憲広訳 培風館

［5］ ＦＡＣＯＭ ＯＳ Ⅳ ＬＩＳＰ 手引書 富士通（金大計算機センターでＬＩＳＰコマンドで起動するＬＩＳＰのマニュアル．付録から読むのがこつ）

○ＴＳＳ及びバッチ処理について（金沢大学計算機センター発行のマニュアル）

［6］ 利用の手引き ＴＳＳ実習書（初心者編）ＦＯＲＴＲＡＮユーザのためのＴＳＳ入門
中島 恵美

［7］ 利用の手引き ＴＳＳ端末によるバッチ処理―フルスクリーン機能と出力検索―
関崎 正夫