

PL/I FORMAC

IBM データセンター 鈴木久子

この記事は、

IBM DATA CENTER User's Guide

SYSTEM/360 PL/I FORMAC

の一部を、講演者の指示に従って、抄録したものである。

PL/I FORMACの言語

PART II では、FORMAC LANGUAGEの機能を詳しく説明します。FORMAC ステートメントを構成するLANGUAGEの要素及びFORMAC Routine についてその機能、使用上の注意、完全なフォーマット (Format) を簡単な例題と共に記述します。

§1 LANGUAGEの要素

FORMAC ステートメントはFORMAC constant, FORMAC variable, FORMAC function等を用いてFORMAC expressionを作成し、そのFORMAC expressionのvalueをFORMAC variableにassignすることであり、PL/Iステートメントとはkeyword "LET" を使用することにより識別されます。

1.1 FORMAC assignment statement

FORMAC assignment statementのFormatは次のようになります。

```
LET (  $v_1 = e_1$  ;  $v_2 = e_2$  ; .....  $v_n = e_n$  ) ;
```

ここで v_n はFORMAC variableであり、 e_n はFORMAC expressionであります。このステートメントが実行されると、右辺のexpression e_n を構成しているFORMAC variableのcurrent value を使用して数式処理が行なわれ、その値が左辺のFORMAC variableのcurrent value となります。PL/Iステートメントとの識別はKeyword "LET" によって行なわれ、1つのLET ステートメントLET () ;で任意の数のassignment statement を実行させることが出来ます。

<例>

```
LET (A=B+C) ;
LET (A=B+3 ; C=A+X ; D=SIN (Y)) ;
```

1.2 FORMAC constants

FORMAC constants には4種類あります。即ち、小数点付実数、整数、有理整数、システム定数であります。

a. Floating point numbers

Floating Point Number は、1~9桁の小数点つき数字である。又 decimal exponent を用いることも出来ます。Floating point number は約 5.4×10^{-79} から 7.2×10^{75} までの数を表わすことが出来ます。Floating point number は長精度 (Double Precision) で記憶されており、又その演算は長精度で行なわれます。

<例>

```
.0032, 6.0, -1.263E23, .034768817E-5, 4E3
```

exponent を使用するときには小数点を省略してもよい。(4E3 ≡ 4000.)

b. Integer constants

1 ~ 2295 桁の整数を表わすことが出来ます。Integer と Floating point number との混合演算ではその結果は Floating point number になる。Integer と Integer の operation (+, -, **, *) は Integer になります。又除算は一般には有理整数となります。

c. Rational numbers

有理整数は Integer 同志の除算によって Integer/Integer の形で表わされる。Rational number 同志の演算は Rational mode arithmetic を使用して行ない、その結果はいつも簡約化される。特に分子が分母で割り切れるときは、その結果は Integer となります。Floating point number との混合演算は Floating point number となります。

<例>

```
LET(A = 1/6 + 1/3; B = 7.3 * A; C = 8 * 6/12; D = 8 * 7/12);
Result: A ← 1/2, B ← 3.65, C ← 4, D ← 14/3
```

d. System constants

前もって FORMAC が備えている 3 個の定数があります。

#E (e ≡ 2.7812 ……) 自然対数の base を表わします。

#P (π ≡ 3.14159 ……) 円周率を示します。

#I (i ≡ √-1) 虚数単位を示します。

これらの System constant は使用方法によって、自動的に変換が行なわれます。

<例>

```
LET(A = #I ** 2; B = #E ** LOG(X); C = SIN(#P/6));
Result: A ← -1, B ← X, C ← 1/2
```

〔注意〕 FORMAC system が Integer を必要とする場合 (例えば, subscripts, FAC 及び COMB の Argument 等) Rational number 又は Floating point number はいつも 4 捨 5 入して Integer になります。

<例>

3.4 → 3, 3.5 → 4, 7/3 → 2, 3/2 → 2, -3.4 → -3, -3.5 → -4

1.3. FORMAC variable

一般に FORMAC ステートメントに現われる variable は FORMAC variable と見られます。FORMAC variable の名前のつけ方は次に記した点を除いて PL/I variable の名前のつけ方と同じであります。

- ① FORMAC variable の名前の長さは最高 8 文字で、underscore (_) を含んではなりません。又 DEN*** の形は FORMAC routine で使用されているので、さげなければなり

ません。

- ② FORMAC がリザーブ (reserve) している記号 (例えば, SIN, LOG, EVAL 等) の使用はさけること。
- ③ FORMAC variable は base, scale, precision を持っていないので DECLARE する必要はありません。

次に FORMAC variable の便利な事柄を記述します。

- ① たとえ FORMAC variable と PL/I variable で同じ名前を使用しても相異なるものと見なされます。

<例>

A=5; LET (A=6); B=A; Result: B←5

- ② FORMAC variable は 4 元まで使用可能であります。サブスクリプトは FORMAC Integer か又は計算結果が Integer になる FORMAC expression であり、そのサブスクリプトの範囲は -31622 ~ +31622 まで許されます。又サブスクリプトのついた FORMAC variable は DECLARE する必要もなければ DIMENSION の定義もする必要がありません。機能上 FORMAC variable は、Atomic variable 及び Assigned variable の 2 種類に分けることができます。

Atomic variable は value (数値又は数式) を持たない variable で変数自身を表わしています。従って FORMAC ステートメントの左辺には現われません。Assigned variable は value を持つ variable で少なくとも一度は FORMAC ステートメントの左辺に現われます。

1.4. FORMAC expression

FORMAC expression は本質的には PL/I expression と同じ様に書かれます。FORMAC variables と constant (+, -, *, /, **) で結ばれ、その計算順序は PL/I expression の時と同様、普通の計算法則が適用されます。FORMAC expression が実行されると、右辺の expression に現われる assigned variable はその時点での値 (current value) を使用して計算します。然しながら、次の例のような場合は注意を要します。

<例>

① LET (A=B+C; B=X*Y; E=A+B);

② LET (B=X*Y; A=B+C; E=A+B);

① のステートメントが実行されると $A \leftarrow B+C$, $B \leftarrow XY$, $E \leftarrow B+C+XY$ となり、 $A=B+C$ の実行時には B は atomic variable と考えられます。B=X*Y; の実行で B は Assigned variable と見なされ XY の値をもち、E=A+B; の実行において $A=B+C$, $B=XY$ の Current value を代入します。A=B+C の B の変数に B=XY の値を代入してから E の計算を行なうというようなことはしません。即ち "Back-Substitution" は行なわれません。一方②のステートメントの実行では $B \leftarrow XY$, $A \leftarrow XY+C$, $E \leftarrow 2XY+C$ となります。

1.5. FORMAC function

FORMAC functionとしてはFORMAC expressionをArgumentとしてとる4種類のfunctionがあり、これらのfunctionはPL/I-Like function, Integer-valued function, user-defined function, function variableがあります。

a. PL/I-like function

PL/Iにおいてnumeric argumentをとる次の様なfunctionはFORMAC においてはsymbolic argumentと共にFORMAC expression の中で使用することが出来ます。

SIN COS SINH COSH ATAN ATANH ATAN(x, y)
LOG LOG10 LOG2 ERF

〔注意〕

これらのfunctionはOPTSET (TRANS) ;が指定されているときは、これらのfunctionのargumentが数値 (floating point, integer, rational) であるときは自動的にfunctionの計算が行なわれます。又OPTSET (NOTRANS) ;が指定されているときには、argumentはsymbolicのまま残っていて計算は行なわれません。

<例>

LET (A=LOG (X**2) + SIN (3/4)) ;

Result : A ← LOG (X²) + .68163876 (OPTSET (TRANS) の時)

A ← LOG (X²) + SIN (3/4) (OPTSET (NOTRANS) の時)

又次の様なfunctionがFORMAC expression の中に使用されると変換されます。

<u>function</u>	<u>translation</u>
EXP (X)	#E**X
SQRT (X)	X** (1/2)
SIND (X)	SIN (1/180*#P*X)
COSD (X)	COS (1/180*#P*X)
TAN (X)	SIN (X)/COS (X)
TAND (X)	SIN (1/180*#P*X)/COS (1/180*#P*X)
TANH (X)	SINH (X)/COSH (X)
ATAND (X)	180*ATAN (X)/#P
ATAND (X, Y)	180*ATAN (X, Y)/#P
ERFC (X)	1-ERF (X)

<例>

LET (A=EXP (SIND (X**2))) ; Result : A ← #E SIN (1/180#PX²)

b. FORMAC integer-valued function

FORMAC expression の中で使用することの出来るものでIntegerの値をもつ

functionとしてFAC, COMB, STEPの3種類があります。

FAC(n) = $n!$

$$\text{COMB}(n, n_1, n_2, \dots, n_k) = \frac{n(n-1)\dots(n-n_1-\dots-n_k+1)}{n_1!n_2!\dots n_k!} \left(\sum_{i=1}^k n_i \leq n \right)$$

〔注意〕

OPTSET controlとしてOPTSET(INT)が指定されるとIntegerに計算される。

OPTSET(NOINT)の時はsymbolicとしてそのまま残っています。

<例>

LET(X=FAC(5); Y=COMB(5;2));

Result: X ← 120, Y ← 10 (INT指定のとき); X ← 5!, Y ← (5;2) (NOINTのとき)

次にSTEP functionはSTEP(e_1, e_2, e_3)の形をとり, expression e_1, e_2, e_3 が数として評価されるなら次の様に定義されます。

$$\begin{aligned} \text{STEP}(e_1, e_2, e_3) &= 1 && e_1 < e_2 < e_3 \text{ 又は } e_1 = e_2 = e_3 \text{ の時} \\ &= 0 && \text{Otherwise} \end{aligned}$$

但し, e_1, e_2, e_3 が数値でないとき, あるいはOPTSET(NOINT)の指定があるときは, Symbolicとして残ります。

<例>

LET(Y=STEP(-1, X, 0)+SEP(0, X, 1) * COS(X));

Result: Y ← 0 X < -1, X > 1の時

Y ← 1 -1 < X < 0の時

Y ← COS(X) 0 < X < 1の時

c. Function Variables

FORMACでは任意の関数を数式を定義することなくfunction Variableとして使用することが出来ます。

一般の形 function name.(e_1, e_2, \dots, e_n)

<例>

LET(Y=F.(A+B, A-B)+SIN(G.(X+3)));

ここでF.(A+B, A-B)はA+B及びA-Bを変数とする関数を表わし, G.(X+3)はX+3を変数とする関数を示しています。又function variableはsubscriptをつけることも出来ます。Y(I, J).(A, B, C)ここでI, Jはsubscriptであります。更にfunction variableは他のfunction variableにvalueとしてassignすることも出来ます。

<例>

LET(F=G; X=F.(A, B));

Results: X ← G.(A, B) となります。

d. User-defined functions

FORMAC では数式を明確に定義する function も使用することが出来ます。

$\text{FNC}(\text{function name}) = \text{expression}(\$(1), \$(2), \dots, \$(n))$

ここで $\$(1), \$(2), \dots, \$(n)$ は expression に従って, function の変数を示しています。

<例>

```
LET(FNC(F) = 3 * COS( $(1) ** 2 + $(2) ) + 1);
```

```
LET(X = B + C ; Y = F(X + 4, 3) + 2);
```

この例で function $F \equiv F(\$(1), \$(2)) = 3 * \text{COS}(\$(1) ** 2 + \$(2)) + 1$ として定義されており, $Y \leftarrow 3 * \text{COS}((B + C + 4)^2 + 3) + 2$ となります。

[注意]

function は使用される前に定義されなければならない。又 function の定義の中で使用された FORMAC variable は, function の定義された時点における FORMAC variable の current value が使用されます。更に前似って定義した function を FNC ステートメントの中で使用することも出来ます。

<例>

```
LET(B = 5 ;
```

```
  FNC(R) = B * COS( $(1) + 2 * $(2) );
```

```
  FNC(S) = R( $(1), B) ** 3 ;
```

```
  :
```

```
  B = 6 ;
```

```
  Y = R(A + B, B * X);
```

```
  Z = S(ALPHA));
```

この結果は FNC(R) では $R \equiv R(\$(1), \$(2)) = 5 * \text{COS}(\$(1) + 2 * \$(2))$ と定義され FNC(S) では $S \equiv S(\$(1)) = \{5 * \text{COS}(\$(1) + 2 * 5)\} ** 3$ と定義され, Y の計算では $B = 6$ が使用されます。従って $Y \leftarrow 5 * \text{COS}(A + 12X + 6)$, $Z \leftarrow 125 * \text{COS}^3(\text{ALPHA} + 10)$ となります。

1.6. CHAIN

CHAIN operation によって, FORMAC expression の list (又は Chain) を作り FORMAC variable に assign することが出来る。これは FORMAC function や routine の argument を作成するのに使います。

<例>

```
LET(X = A + 3 * SIN(B); Y = CHAIN(A, X + A, B + 7, 4));
```

```
Results: Y ← (A, 2A + 3 SIN(B), B + 7, 4)
```

```
LET(X = (A, B + 2, C); Y = F.(X, D));
```

Results : $Y \leftarrow F(A, B+2, C, D)$

なお, chain を定義するときには, "CHAIN" を省略してもよい。 $X = (A, B+2, C);$ は $X = \text{CHAIN}(A, B+2, C);$ と同じことを示します。

1.7 FORMAC output

FORMAC output として PRINT_OUT ステートメントがあります。これは FORMAC variable の value をプリントアウトします。

<例>

```
LET(X=A**5+SIN(3*A)**2+2/3); PRINT_OUT(X);
```

Results : $X = A^5 + \sin^2(3A) + 2/3$ とプリントアウトされます。

[注意]

- ① 1個の PRINT_OUT ステートメントで複数の FORMAC variable をプリントすることが出来ます。即ち PRINT_OUT(X;Y;Z); と書くとき、各ラインにそれぞれ変数とその value がプリントアウトされます。
- ② atomic variable X を PRINT_OUT(X) とすると、X= とプリントされます。
- ③ Floating Point number は9桁の文字と小数点でプリントされます。 $10^{-2} \sim 10^6$ 外の数は E format でプリントされます。
- ④ Keyword "LET" は PRINT_OUT で置きかえることが出来ます。

<例>

```
LET(A=B+C); PRINT_OUT(A);
```

```
LET(D=SIN(X)); PRINT_OUT(D);
```

これは次の様にも書くことも出来ます。

```
PRINT_OUT(A=B+C; D=SIN(X));
```

§2 FORMAC routine

FORMAC においては FORMAC expression の数式処理を行なうのに各種の routine が備えられています。然しながら FORMAC routine は、幾つかの例外を除いて FORMAC expression を value として送ってくれる function であります。又 FORMAC routine の argument はその routine が call される前に計算されます。

2.1 User²Controlled Simplification

この routine には expression を展開する為の routine と、 a/b 形の expression にする為の routine とがあります。

a. Expansion Routines : MULT, DIST, EXPAND

Format : (i) MULT(*e*)
 (ii) DIST(*e*)
 (iii) EXPAND(*e*)

Result :

(i) MULTは expression e の中に現われる“べき乗”を展開する。NOINT option が set されていると展開されて出来る係数を combinatorial を使用して表わします。

<例>

```
LET(X=(A+B)*(A-B)**2; Y=MULT(X)); Y←(A+B)(-2AB+A2+B2)
OPTSET(NOINT); LET(X=5*(A+B)**3; Y=MULT(X)+A**5);
Y←5((3;1)B2A+(3;1)BA2+A3+B3)+A5
```

ここで (3; 1) は COMB(3, 1) を意味しています。

(ii) DISTは expression e に分配法則 (distributive Law) を適用します。

即ち $(a+b)(c+d) \rightarrow ac+ad+bc+bd$, $a(b+c) \rightarrow ab+ac$, $(a+b)/c \rightarrow a/c+b/c$

DISTはMULTの働きを含んでいません。

<例>

```
LET(P=DIST(C*(A+B)); Q=DIST((A+B)/C)); P←AC+BC, Q←A/C+B/C
LET(Y=(X-A)**2+2*(X-A)**4+3*(X-A)**6; Z=DIST(2*(X-A)**3*Y));
Z←2(X-A)5+4(X-A)7+6(X-A)9
```

(iii) EXPANDはMULTとDISTの機能を両方とも備えていて, expression e を展開します。

<例>

```
LET(X=A*(COS(B(C+D))+A); Y=3*(A-2)**2; Z=EXPAND(X-Y));
Result: Z←12A+COS(BC+BD)A-12A2-12
LET(X=(A**2+A*B*C)/(A*B); Y=EXPAND(X)+4*C);
Result: Y←A/B+5C
LET(Z1=-3+4*#I; Z2=-2+7*#I; PRODUCT=EXPAND(Z1*Z2));
Result: PRODUCT←-34+13#I
```

[注意]

MULT, DIST及びEXPANDは factored denominators(因数化された分母) は完全に展開はしない。

<例>

```
LET(X=3*(Q+7)/(P*(Q+R)**3); Y=EXPAND(X));
Result: Y←3Q/(P(Q+R)3)+21/(P(Q+R)3)
```

更にOPTSET controlとしてOPTSET(EXPND)が指定されるとすべての expression の計算で展開が自動的に行なわれます。

b. Common Denominator Functions : CODEM, FRACTN

Format (i) CODEM(e)
(ii) FRACTN(e)

Result : 両方の function 共 expression e を a/b の形に変換する。

- (i) CODEMはすべてのレベルにおける expression を a/b の形にする。
 (ii) FRACTN(e) は外側のレベル (outermost level) だけにおいて a/b の形にする。

<例>

```
LET(X=A/(B+C/D)+E/F; Y=CODEM(X); Z=FRACTN(X));
Result: Y←(FDA+(C+DB)E)/((C+DB)F)
        Z←(FA+(B+C/D)E)/((B+C/D)F)
```

2.2. Substitution routine

FORMACにおいて、substitution routineにはEVALとREPLACEがあります。これらのroutineはargumentのexpressionに任意のexpressionを代入して計算する働きを持ってあります。

a. EVAL

Format: EVAL($e, a_1, b_1, a_2, b_2, \dots, a_n, b_n$)

Result:

EVALはargument expression e の中のatomic variable, system constant, 又はfunction a_i に任意のexpression b_i を代入します。

- ① a_i がatomic variable又はSystem constant(#E, #I, #P)の時はexpression e の中に a_i が現われる度に b_i を代入します。
- ② a_i がfunction則ち $f(\$ (1), \$ (2), \dots, \$ (m))$, b_i が $\epsilon(\$ (1), \$ (2), \dots, \$ (m))$ のときexpression e の中の $f(arg_1, arg_2, \dots, arg_m)$ は $\epsilon(arg_1, arg_2, \dots, arg_m)$ で代入されます。
- ③ 代入方法は並行代入 (parallel substitution) であります。

<例>

```
LET(X=#P*(R-RHO)**2; Y=EVAL(X, #P, 3.141592, R, A+B));
Y←3.141592(A+B-RHO)2
LET(X=CHAIN(A, B, B, E, C, A); Y=A*(A+B)+C*(A+D);
Z=EVAL(Y, X));
```

XはCHAINによってリスト $X \leftarrow (A, B, B, E, C, A)$ になっていますから $Z = EVAL(Y, X)$ は $Z = EVAL(Y, A, B, B, E, C, A)$ と同じことを示します。又(A, B), (B, E), (C, A)をpairにして代入されますがAにBを, BにEを代入, 従ってAにEを代入するとはなりません。それぞれ独立して代入されます。(parallel substitution)
 従って, 結果は $Z \leftarrow B(B+E) + A(B+D)$ となります。

<例>

```
LET(X=F.(A-B, A+B)+G.(C**2)**2;
Y=EVAL(X, F.($), $(1)**2+$(2), G.($), COS($(1))));
```

$$Y \leftarrow (A-B)^2 + (A+B)^2 + \text{COS}^2(C^2)$$

〔注〕 F.(\$ (1), \$ (2)), G(\$ (1)) の代りに F.(\$), G(\$) を使用することが出来ます。

b. REPLACE

Format : REPLACE(e, a₁, b₁, a₂, b₂, ..., a_n, b_n)

Result :

a_i, b_i は任意の FORMAC expression であり, expression e 中の subexpression a_i に subexpression b_i を代入します。代入方法は連続代入 (serial substitution) で次の様に行なわれます。即ち expression e 中の a₁ に b₁ を代入し, その結果得られる expression e₁ とすると e₁ 中の subexpression a₂ に subexpression b₂ を代入します。同様に (a_n, b_n) まで行なわれます。

<例>

```
LET (Y=A*X**2+B*X+C*D+E; Z=REPLACE(Y, X**2, T(2), X, T(1),
C*D*E, CONST));
```

```
Z←AT(2)+BT(1)+CONST
```

<例>

```
LET(X=CHAIN(A, B, B, E, C, A); Y=A*(A+B)+C*(A+D);
```

```
Z=REPLACE(Y, X);
```

```
Z←2E2+A(E+D)
```

4.3. Analytic Differentiation Routines: DERIV, DIFF, DRV

FORMAC では 3 種類の微分関係の routine があります。

DERIV は任意の変数で, expression を偏微分します。DIFF は function variable の偏関数を取り出します。又 DRV は function variable の argument に関して微分します。

a. DERIV

Format : DERIV(e, V₁, n₁, V₂, n₂, ..., V_r, n_r)

Result :

e は expression で, V_i は atomic variable, n_i は非負の整数であるとき求められる結果は次の様になります。

$$\frac{\partial^n}{\partial V_1^{n_1} \partial V_2^{n_2} \dots \partial V_r^{n_r}} \cdot e \quad (\text{ここで } n = \sum_{i=1}^r n_i)$$

なお, n_i が 1 のときは省略してもよい。

<例>

```
LET(P=3*SIN(A*X)+A*X**2;
```

```
Q=DERIV(P, X); R=DERIV(P, X, 2, A, 1));
```

```
Q←3A COS(AX)+2AX, R←-3A2X COS(AX)-6ASIN(AX)+2
```

<例>

```
LET(Q=DERIV(F.(A, B, C), A, 2, C, 1); R=DERIV(G.(A), A, 3));
Q←F(123).(A, B, C), R←G(13).(A)
LET(X=DERIV(A**2+F.(A, B), A, 2);
Z=DERIV(F.(A+B, 2*A*B), A, 2));
Y←F(12).(A, B)+2, Z←F(12).(A+B, 2AB)+4BF(12).(A+B, 2AB)
+4B2F(23).(A+B, 2AB));
```

<例>

```
LET(S=EVAL(Q, F.($), $(1)**5*$(2)*$(3));
T=EVAL(R, G.($), SIN(2*$(1))));
```

今Q, Rとして前の例の結果 $Q = F^{(123)}.(A, B, C)$, $R = G^{(13)}.(A)$ を使用するとすると
 $S \leftarrow 20A^5B$, $T \leftarrow -8 \cos(2A)$ となります。

b. DIFF Pseudo variable

Format: DIFF(f)=CHAIN(e_1, e_2, \dots, e_n)

Result :

n 個のvariableからなるfunction variable f の1階偏導関数をexpression e_1, e_2, \dots, e_n にassignする。然しながらDIFFはPseudo variable (見かけ上の変数)であるから e_1, e_2, \dots, e_n を取り出すことは出来ません。又expression e_1, e_2, \dots, e_n はfunction variableの i 番目のargumentを $\$(i)$ として使用することが出来ます。もし、任意の偏導関数を定義する必要のないときは e_i に $\$$ を指定します。

<例>

```
LET(Y=F.(3*X); DIFF(F)=CHAIN($(1)**2+A**2); R=DERIV(Y, X));
R←3(9X2+A2)
DIFF(F)より$(1)≡3Xとなり偏導関数 $e_1 \equiv 9X^2 + A^2$ を作り $R \equiv 3F^{(1)}.(3X)$ の $F^{(1)}.(3X)$ に $e_1$ を代入するとRの結果となります。
```

<例>

```
LET(Y=A**2+G.(A+B, 2*A*B); DIFF(G)=CHAIN($(1)+$(2), 1-$(2));
Z=DERIV(Y, A));
Z←3A+B+2AB+2B(1-2AB)
DIFF(G)よりfunction variable G.(A+B, 2*A*B)の偏導関数 $e_1 \equiv \$(1) + \$(2) \equiv A+B+2AB$ ,  $e_2 \equiv 1-\$(2) \equiv 1-2AB$ を作り $Z \equiv 2A+G^{(1)}.(A+B, 2AB) + 2BG^{(2)}.(A+B, 2AB)$ の $G^{(1)}.(A+B, 2AB)$ ,  $G^{(2)}.(A+B, 2AB)$ にそれぞれ $e_1, e_2$ を代入します。
```

<例>

```
LET(Y=F.(B-2*A, B+2*A); DIFF(F)=CHAIN($, SIN($ (1)+$ (2)));
```

```
Z=DERIV(Y, A);
```

```
Z ← -2F(1).(B-2A, B+2A) + 2 SIN(2B)
```

\$は偏導関数 e_1 を定義しないことを示しています。 $e_2 \equiv \text{SIN}(2B)$,

$Z \equiv -2F^{(1)}.(B-2A, B+2A) + 2F^{(2)}.(B-2A, B+2A)$ に e_2 のみを代入します。

c. DRV

Format: $\text{DRV}(f(\text{arg}_1, \text{arg}_2, \dots, \text{arg}_r), \$ (1), n_1, \$ (2), n_2, \dots, \$ (r), n_r)$

Result:

n_i は非負の整数で、 $\$(i)$ は arg_i を示しています。この時次の様な結果となります。

$$f^{(1^{n_1} 2^{n_2} \dots r^{n_r})}(\text{arg}_1, \text{arg}_2, \dots, \text{arg}_r)$$

なお n_i が省略されると1と見なします。

<例>

```
LET(Y=DERIV(G.(X**2), X); Z=DRV(G.(X**2), $ (1)));
```

```
Y ← 2XG(1).(X2), Z ← G(1).(X2)
```

これはDERIVとDRVとの相異を示しています。即ちDERIVはatomic variable で微分するのに比して、DRVはargument (この例では $\$(1) \equiv X^2$)で微分することを示しています。

<例>

```
LET(A=DRV(F.(X+Y, Y, X*Y), $ (1), 2, $ (3)));
```

```
A ← F(12 3).(X+Y, Y, XY)
```

```
LET(DIFF(G)=CHAIN($, $ (1)*COS($ (2)));
```

```
D=DRV(G.(X*Y, X-W), $ (2));
```

```
D ← XYCOS(X-W)
```

この例ではDIFF(G)により e_1 は偏導関数を求めないけど e_2 の偏導関数は

$e_2 \equiv \text{XYCOS}(X-W)$ となり、これらを $D \equiv G^{(2)}.(XY, X-W)$ に代入します。

2.4. Comparison: IDENT

Format: $\text{IDENT}(e_1; e_2)$

Result:

2つのFORMAC expression e_1 と e_2 を比較し数式として等しいとき IDENT は1となる。等しくないときは0となる。代数的な比較は行ないません。

<例>

```
LET(X=A+C-D*E+A*SIN(-A)+A;
```

```
Y=-D*E+C+2*A-SIN(A)*A);
```

IF IDENT(X;Y) THEN.....

この例では, AUTSIM transformation (Automatic Simplification) と並べかえ (lexicographical recording) によって $X \leftarrow 2A - ASIN(A) + C - DE$, $Y \leftarrow 2A - ASIN(A) + C - DE$ となり, X と Y は数式的に等しいと見なされ, 従って IF ステートメントの THEN 以後が実行されます。

<例>

```
LET(X=3*(A+B); Y=3*A+3*B); BIT1=IDENT(X;Y);
    BIT2=IDENT(EXPAND(X);Y);
```

PL/I variable BIT1=0, BIT2=1 となります。BIT1=IDENT(X;Y) は代数的には等しいけれど, 同一数式ではありませんので BIT1=0 となります。

2.5. Expression Analysis Routine:

これらの Routine は FORMAC expression を分析する時に使用します。expression の係数関係の処理をする routine, a/b 形の expression を処理する routine, 更に expression の operator の処理をする routine があります。

a. COEFF, HIGHPOW, LOWPOW

Format: (i) COEFF(e_1, e_2)
 (ii) HIGHPOW(e_1, e_2)
 (iii) LOWPOW(e_1, e_2)

Result:

(i) COEFF は expression e_1 の中にある subexpression e_2 の係数を取り出します。

<例>

```
LET(Y=A*X**3+2*A*X+(A-B)*(A+B)**2+X**Z+7;
    C1=COEFF(Y,X**3); C2=COEFF(Y,X**Z); C3=COEFF(Y,X);
    C4=COEFF(EXPAND(X*Y),X); C5=COEFF(Y,A);
    C6=COEFF(Y,(A+B)**2);
```

Result: C1 ← A, C2 ← 1, C3 ← 2A, C4 ← $-B^2A + A^2B + A^3 - B^3 + 7$
 C5 ← $2X + X^3$, C6 ← A - B

(ii) HIGHPOW は expression e_1 の中から subexpression e_2 に関する最も高い power (べき乗) をとり出して来ます。

(iii) LOWPOW は expression e_1 の中から subexpression e_2 に関する最も低い power をとり出します。

<例>

```
LET(Y=X**5+(A+B)**4*X**2+7*LOG(X+1);
    P=HIGHPOW(Y,X); Q=LOWPOW(Y,X); R=HIGHPOW(Y,A+B);
    S=LOWPOW(Y,A);
```

$P \leftarrow 5, Q \leftarrow 2, R \leftarrow 4, S \leftarrow 0$

b. NUM, DENOM

Format: (i) NUM(e)
(ii) DENOM(e)

Result:

- (i) NUMは expression e が a/b 形である時, numerator a を取り出します。
(ii) DENOMは expression e が a/b である時, denominator b を取り出します。
なお, 負のべき乗をもつ expression 等は denominator として取り出されます。更に expression e が a/b 形でないときは, NUMは e を DENOMは 1 の値となります。

<例>

LET(P=A/B+C/D; Q=CODEM(P); R=NUM(P); S=DENOM(P);
T=NUM(Q); U=DENOM(Q));

Result: $R \leftarrow A/B+C/D, S \leftarrow 1, T \leftarrow AD+BC, U \leftarrow BD$

<例>

LET(X=(A+B)/3; Y=NUM(X); Z=DENOM(X));

$Y \leftarrow 1/3(A+B), Z \leftarrow 1$ この例に見られる様に rational number の numerator, denominator は取り出すことは出来ません。

c. Non-algebraic Manipulation Function. LOP, NARGS, ARG

LOP, NARGS, ARG は任意の expression の構造に関する情報をうるための function である。これらの function の目的は, 任意の expression が “Lead operator” で関連づけられた argument の chain であると考えるところにある。即ち

X^2 は Lead operator $**$ によって支配された Chain($X, 2$) である。

$3AX^2$ は $\text{Chain}(3, A, X^2)$ である。

AX^2+BX+C は $\text{Chain}(AX^2, BX, C)$ である。

このように expression を考えることにより, LOP は Lead operator を, NARGS は Argument の数を, ARG は i 番目の argument をそれぞれとり出すことにある。

(i) LOP

Format: LOP(var). ここで var は FORMAC variable name である。

Result:

LOP は FORMAC variable “var” が持っている “Lead operator” によって次の様な PL/I integer(FIXED, (31, 0)) をとり出します。

LOP	Lead operator	LOP	Lead operator
0	ERF	3	LOG2
1	LN	4	SIN
2	LOG10	5	COS

LOP	Lead operator	LOP	Lead operator
6	ATAN	32	negative rational constant
7	ATAN(X,Y)	33	negative long integer
8	SINH	34	negative floating constant
9	COSH	35	negative short Integer
10	ATANH	36	zero
11	CHAIN	37	positive rational constant
12	DERIV	38	positive long integer
15	COMB	39	positive floating constant
16	FAC	40	positive short integer
19	STEP	41	#P
21	FNC	42	#E
24	PLUS(+)	43	#I
25	MINUS(-)	44	VAR(atomic variable)
26	TIMES(*)	45	BVAR(S-variable)
31	EXPON(**)	46	atom(new atomic)
		99	undefined Lead operator

<例>

```
LET(P=3*X**2+4*X+5;Q=A*G.(A+B,C));LOPP=LOP(P);
  LOPQ=LOP(Q);
Result:LOPP=24(Code'+'),LOPQ=26(Code '**')
```

<例>

```
LET(X=A/B;Y=A-B);L1=LOP(X),L2=LOP(Y);
Result:L1=26(Code '**'),L2=24(Code '+')
```

内部的にはA/B, A-BはそれぞれA*B*(-1), A+(-B)と記憶されています。

(ii) NARGS

Format: NARGS(var)

Result:

NARGSはvarにassignedされたexpressionのLead operatorによって結ばれているargumentの数を取り出して来ます。しかもPL/I integer (FIXED, (31, 1))であります。

<例>

```
LET(P=3*X**2+4*X+5;Q=A*G.(A+B,C);NP=NARGS(P);
  NQ=NARGS(Q));NP=3(3X2と4Xと5),NQ=2(AとG.(A+B,C))となります。
```


(iii) ARG

Format: ARG(n, e)

Result:

ARGは expression e の n 番目の argument をとり出して来ます。 n は非負の整数。
 $n=0$ の時は e となります。

<例>

```
LET(P = 3 * 4 ** 2 + 4 * X + 5);
DO I = 1 TO NARGS(P);
LET(I = "I"; OP(I) = ARG(I, P));
END;
```

Result: $P \leftarrow 4X + 3X^2 + 5$, NARGS = 3 従って OP(1) $\leftarrow 4X$, OP(2) $\leftarrow 3X^2$,
 OP(3) $\leftarrow 5$ となり argument をとり出して来ます。

2.6. Economization of Storage

長い FORMAC expression は大きな space を必要とするので、使用出来る記憶場所がなくなってしまうことがあります。これを防ぐためには2つの方法があります。

a. SAVE

Format: SAVE($var_1; var_2; \dots; var_n$)

Result:

VAR _{i} に assign されている FORMAC expression は二次記憶装置に移され、それまで VAR _{i} によって使用されていた記憶場所は解放される。続いて Var _{i} を使用すると自動的に core の中へとり出して来ます。SAVE されていた FORMAC variable を一度使用すると、二次記憶装置のその場所は解放されるので、更に resave するためには今一度 SAVE する必要があります。

<例>

```
(i) LET(A = X + Y);
(ii) SAVE(A);
    :
(iii) LET(B = A + C);
(iv) SAVE(A; B);
    :
(v) LET(A = X * Y);
```

結果としては(i)では $A \leftarrow X + Y$ となり、その結果を(ii)により secondary storage に copy され、A に使用されていた space は解放される。(iii)により A を使用することにより再び secondary storage から core の中に取り出して来て、その結果 $B \leftarrow X + Y + C$ となると共に A の expression を記憶している Secondary storage は scratch されます。従っ

て④によりA, Bにassignされている expression が secondary Storageに store されます。⑤によりA←XYとなりsecondary storageのAの expression は scratchされます。

b. ATOMIZE

Format: ATOMIZE (var₁; var₂; ...; var_n)

Result:

FORMAC variable varはatomic variableとなり, varにassignされていた expressionは消えてしまうと同時にそのspaceは解放されます。

2.7. FORMAC option: OPTSET

Format: OPTSET(option₁; option₂; ...; option_n)

Result:

optionによって指定された下記の optionによりセットされる。一度setされると次の OPTSETによりリセットされるまでその効力は残っています。

optionとしては次の様なものがあり, 指定しないとunderlineのついたものが使用されます。

OPTSET option:

- | | |
|--------------------------|-----------------|
| 1. <u>TRANS</u> | NOTTRANS |
| 2. <u>INT</u> | NOINT |
| 3. <u>EXPND</u> | <u>NOEXPND</u> |
| 4. <u>EDIT</u> | NOEDIT |
| 5. <u>PROPER</u> | <u>IMPROPER</u> |
| 6. <u>LINELENGTH=xxx</u> | (<u>120</u>) |

Effect:

- TRANSはPL/I Like-function (SIN, LOG etc) の argumentが numeric constant であるときは, 常にその関数値を計算します。
NOTTRANSはsymbolic entityとして残しておきます。
- INTはFORMAC integer function FAC, COMB, STEPの argumentが numericであるときその関数を自動的に計算します。
- EXPNDはexpressionの計算をするとき, multinomial及びdistributiveの法則を適用します。NOEXPNDはこの機能を中止させる働きがあります。
- EDITは結果をprint outするとき, 編集して見易い様にします。
NOEDITはCoreに記憶されている形のままプリントアウトします。
- PROPERはrational numberを $a+b/c$ ($b < c$) の形に直します。
IMPROPERは a/b の形のままでおきます。
- LINELENGTH=xxx は結果をプリントするとき1行に印刷される文字の数を示し

ています。指定しないと 120 文字となります。

〔注意〕

この他に option として PRINT, NOPRINT があります。OPTSET(PRINT) はこのステートメントに続く LET ステートメントをすべて PRINT OUT ステートメントと同じ働きをもたせます。これはプログラム・デバッグに役立ちます。OPTSET (NOPRINT) は PRINT の機能を中止させます。なお、この PRINT, NOPRINT の指定は他の option と一緒に OPTSET の中で書いてはいけません。

§3 FORMAC-PL/I Interface

ここでは FORMAC variable と PL/I variable との結合、及び Error の生じた時の処理の仕方、FORMAC source deck の構造について記します。

3.1 FORMAC statement の中で PL/I variable の使用

a. Double Quates operator

Double Quates operator (") で囲まれた PL/I variable は LET, PRINT _ OUT, SAVE の中の FORMAC expression の中において使用することが出来ます。

<例>

PL/I variable A, B, C がそれぞれ character で 3, 22/7, A*X**2+DERIV (Y, X) であり

```
LET(Y= SIN(X); Z="A" * A * "C" + "B");
```

が実行されると $Z \leftarrow 3A(AX^2 + \cos(X)) + 22/7$ となります。

<例>

次の例は DO ステートメントを使用するとき有効であります。

```
DO I=1 TO N; LET(I="I");
```

```
  :
```

```
LET(A(I)=B(3*I)+I);
```

```
  :
```

```
END;
```

この中では DO I=1 TO N; の I 及び "I" は PL/I variable であり、

LET(I="I") の I は FORMAC variable であります。

<例>

FORMAC expression を PL/I variable として取りだすためには LET("PL/I var"=FORMAC expression) と指定します。

```
LET(Y=X**2+3*X; "A"=Y+1);
```

この結果、A は 'X**(2)+3*X+1' となります。なお、この時 PL/I variable character string A は FORMAC variable を表わしうるに十分な長さを持っている

なければなりません。

b. Argument Passing

internal又はexternal procedureへargumentとしてFORMAC variableをpassするための一般的な方法はCALL statementのargument Listのvariableをsingle quotes (')で囲んでやる必要があります。

<例>

```
P:PROC(A, B);
  DCL(A, B) CHAR(8);
  LET("B"="A"+3);
  END P;
  :
  CALL P('X+2', 'BETA');
```

この結果FORMAC variable BETA←X+5

なお、"called" Procedureの方ではFORMAC variableに対応するparameterをPL/I character string variableとDeclareし、LET, PRINT_OUT, SAVE statementで使用するときは、常にdouble quote('')で囲んで使用します。

c. CHAREX

Format: CHAREX(var=variable)

Result: PL/I varying character string variable varに右辺のFORMAC variableのcharacter string valueをassignする。

<例>

```
LET(Y=A/B; Z=SIN(3*Y*B)/B); CHAREX(X=Z);
X←'Z=B**(-1)*SIN(3*A)'
```

となります。

3.2. PL/Iステートメント中でのFORMACの使用。

次の2つのステートメントはFORMAC constantをPL/I constantに変換します。

```
Format: INTEGER(var)
        ARITH(var)
```

Result: FORMAC variable varがnumeric constantの時、INTEGERはvarの値をfixed point binary(31,0)numberに変換します。varがfloating point又はrational numberの時はroundされます。ARITHはvarの値をdouble precisionのfloating point numberにします。

<例>

```
LET(A=4; B=5.2; C=11/4);
X=INTEGER(A)+INTEGER(B)+INTEGER(C)
```

Y = ARITH(A) + ARITH(B) + ARITH(C)

Result: X ← 4 + 5 + 3 = 12, Y ← 1.195E01

3.3. FORMAC ON CONDITION

FORMAC ステートメントを実行中におきるError は5つに大別されます。

- a. DENSIZE ……使用出来る core space がなくなった場合です。SAVE又はATOMIZE を使用してspace を解放する。
- b. DENSYN ……使用したFORMAC ステートメントの構造上の間違いの時です。
- c. DENSEM ……使用したFORMAC ステートメントの機能上の間違いの時です。
- d. DENYSM ……FORMAC System Error です。
- e. DENERRR ……上記のエラーが発生した時にそのメッセージをプリントするためのものです。上記のエラーが発生した時、特別な処理をするためには次の様になります。

ON CONDITION(DENSYN) GO TO L2;

L2: 特別な処理をするステートメント

⋮

なお、ON CONDITIONがなければ上記DENSIZ, DENSYN, DENSEM, DENYSM のエラーが起きたすぐ後にDENERRR が発生してエラーに関するメッセージをプリントします。

3.4. FORMAC プログラムの構成

PL/I FORMAC プログラムは、2つのmoduleで構成されている。第1のmoduleは Preprocessor と呼ばれ、FORMAC Languageを、その機能に応じた様々な subprogram を含んだPL/I Language にほん訳すると同時にエラー・メッセージをプリントします。

第2のmodule はFORMAC Library と呼ばれ、様々な subprogramを含んでおり、これらはLink Editor によって実行可能なmoduleの中に組みこまれます。

FORMAC JOB の流れ

FORMAC JOB の流れをステップに分けて考えると次のようになります。

