連載講座
3

電子計算機의 프로그래밍과 그 應用 (3)

宋 吉 永*
(Kilyeong, Song)

3. FORTRAN에 依한 Programming

3—1 自動 Programming과 FORTRAN 概說

앞서 說明한 바와 같이 電子計算機는 各機種에 固有한 特定된 命令語(이 言語를 機械語라고 함) 밖에 實行못하는 것이였다.

곧 電子計算機는 一連의 機械語를 順次 解讀하여 實行해 나가는 것임으로 必要로 하는 일을 計算機에 시키려면은 이와같은 모든 順序를 機械語로 넣어주어야만 되었고 이러한 Programming을 Machine Coding이라고 하였다.

機械語는 電子計算機 自身에 對하여서는 곧 實行할 수 있는 言語로서 理解하기 쉬운 것이겠지만 우리들에게 對하여서는 極히 不便한 것이였다. 곧 하나의 機械語를 構成하는 要素가 여러가지 있으면 그 全體에 注意하여 指示를 준다는 것은 번잡한 일이며 또 틀리기 쉬운 일이기도 한 것이다. 또 機械語는 普通 數字로 表現되는데 이러한 數字만 가지고 곧 命令語와 計算內容이 直感的으로 結付한다는 것은 힘들기 때문에 不得已 이것들을 完全히 記憶해 버린다던가 Programming할 때 Reference Book를 언제나 參照하여야만 한다는 不便이 있었던 것이다.

이러한 不便을 없애기 爲하여 機械語(Machine Langnage) 代身에 나온 것이 Symbolic Language이다. 이것을 크게 나누어 두개의 Type, 곧 Assembler와 Compiler가 있다는 것은 앞서 說明하였다.

이 兩者는 機械語 그 自體로서가 아니고 우리에게 알기쉬운 Symbolic한 表現으로 Programming을 하고저 하는 것이다. Assembler는 大體的으로 機械語의 하나 하나와 1對 1로 對應하는 Symbolic한 命令語로 이루워지고 있다.

Compiler는 이것을 다시 더 發展시켜 機械自體에서 떠난 獨自의 文法을 가지고 하나의 文을 多數個의 機械語와 對應케 한 것이다.

Assembler는 그 機能으로 보아 電子計算機의 各機種에 固有된 것이며 따라서 Programmer는 各機種마다 그 Symbolic Language를 記憶해 두지 않으면 안된다.

그러나 Compiler는 各機種에 關係없는 共通言語로서의 性格을 가지게 할 수 있는 것이다.

Cobol이라고 불리는 것이 그 代表的인 것이며 本章에서 說明코저 하는 FORTRAN도 共通言語로서의 性格을 가지는 것이다.

Symbolic Language로 Coding된 Program을 Source program이라 말한다.

앞서 說明한 바와같이 電子計算機自體가 解讀할 수 있는 것은 어디까지나 機械語이며 Symbolic Language 그대로서는 實行이 안된다. 따라서 Source Program을 機械語로 고칠 必要가 있으며 이것을 "Assemble한다" 또는 "Compile한다"라고 말하고, 그 結果 만들어진 Program을 Object Program이라고 하고 있다.

또 Source Program을 Assemble 또는 Compile하는 것도 電子計算機가 實行하고 있으며 이 變換을 맡아하는 것을 Assembler 또는 Compiler (때로는 Processor)라고 부른다.

한편 電子計算機에 對한 言語를 다른 見地에서 보아 Machnie Dependent Language와 Machine Independent Langnage로 나눌 수가 있다.

特히 後者는 電子計算機自體에 依存하지 않고, 곧 그 計算機가 어떤 命令을 가지고 있으며 또 어떤 機能을 가지고 있는가를 全然몰라도 Source Program을 만들 수 있다는 것이다.

FORTRAN은 이와같이 가장 進步된 program이며 때로는 이것을 Problem-Oriented Language에 屬한다고 부르고 있다.

3—2 FORTRAN 解說

FORTRAN은 Formnlar Translator의 略稱으로 本來는 科學技術計算을 便利하게끔 한다는 意圖에서 만들어진 Compiler로서 上述한 바와같이 Machine Independent한 共通言語로서의 性格을 갖춘 것이다.

FORTRAN言語는 하나의 文(Statement)을 基本的인 單位로 하고 있다. 곧 一定한 形式에 따라 여러가지 Statement를 組立하여 하나의 Program으로서 完成하는 것이다.

또 이 Fortran의 特徵으로는 Statement의 表現이 퍽 數式에 가까워 日常 우리가 쓰는 數式을 거의 그대

* 正會員韓國電力株式會社技術部

로 쓸수 있다는 것이다. 例를 들면

$$F=(A+B+C)D/E$$

라는 問題를 풀기 爲하여서는 從前까지의 Machine Coding 에서는 이 數式을 1 step 씩의 命令으로 細分하여

A 를 Load하고
B 를 加하고
C 를 加하고
D 를 곱하고
E 로 나누어
F에 Stor한다.

라는 內容의 命令을 Coding 하지 않으면 안되었는데 對하여 Fortran 言語에서는 이것을 $F=(A+B+C)*D/E$ 로만 쓰면 끝나는 것이다.

Statement 는 基本的으로는 Coding Sheet의 1 Line 에 하나씩 쓰여진다. 또 이 Coding Sheet의 1 Line 이 한장의 Card에 Punch 되고 이것이 Compiler 에의 Input 로 되는 것이다.

Statement 는 Input 된 順序로 實行되는 것이 基本的인 Rule 임으로 Coding Sheet 上에 實行하고저 하는 順序로 Statement 를 써나가면 되는 것이다.

다음에 具體的인 例를 하나 들어 Fortran Program 이 어떻게 만들어져 나가는 가를 說明하여 보겠다.

지금 三角形의 底邊 (a)와 높이 (h)를 주고 그 面積 (S)를 求하는 簡單한 問題를 생각하여 본다.

이 問題는

$$S=\frac{1}{2}ah$$

를 計算하는 것인데 Fortran 에서는 이것을 다음과 같이 쓰고 있다.

```
    READ 100, A, H
    S=(A*H)*0.5
    PRINT 110, S
100 FORMAT (2F 5 1)
110 FORMAT (2HS=, F 5.1)
    END
```

이것을 說明한다면 먼저 DATA A및 H를 100番의 數値表現形式(Format)으로 準備된 Card 를 읽어드린 다음 $S=\frac{1}{2}ah$ 를 求하고 다시 이 結果를 110番의 表現으로 印刷한다는 것이다.

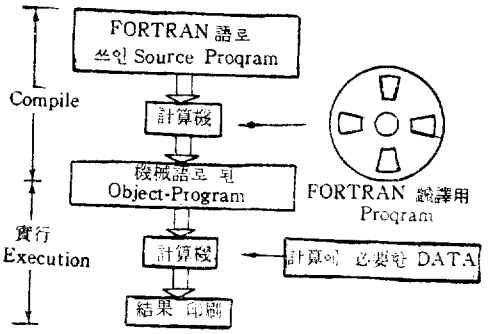
이와같이 Fortran 으로 쓰여진 Program 을 Source Program 이라고 한다. 다만 이것은 計算株가 直接理解할 수 있는 言語——곧機械語——가 아니기 때문에 일단 이것을 計算하고저 하는 機械獨自의 機械語로 번역해 주어야 할 것이다. 따라서 電子計算機는 計算을 시작하기 前에 먼저 計算機 自身이 이러한 번역을 하게되는데 (이 作業을 Compile 이라고 한다) 이때 이 번역을 담당하는 Program 이 實은 Fortran 이라고 불리어지는 Programming System 인 것이다.

다음 Fortran 번역 Program에 依하여 번역된 機械語의 計算 Program 을 Object Program 이라고 부르고 있다.

따라서 實質的인 計算은 이 Object Program 이 計算機에 들어간 後에라야만 實行되는 것으로서 一般으로 電子計算機는 그림 3—1에 보이는 바와같이 Compile 과 實行(Execution)의 두가지 Process 를 거치게되는 것이다.

이 外에 Fortran System 에서는 Source Program 의 Error 를 發見하여 주는 診斷 Program 도 아울러 가지고 있음으로 從前과 比較하여 Program 訂正作業(Debugging)이 아주 容易하게 되고 있다.

따라서 이와같은 Fortran Program 을 배운다는 것은 上述한 바와같이 번역 可能한 數式을 배운다는 것으로 式自體는 別것아니고 다만 어려운 것이 있다면 入出力에 關한 表現法에 若干 복잡한 것이 있다는 정도 일것이다.



3-3 Fortran 規約

Fortran 은 本來 IBM 704에 對하여 開發되었으나 IBM 650에 對한 것도 만들어져 所謂 Automatic Programming 을 처음으로 實現하였다는 意味에서 Software 發展史上 劃期的인 業績이라고 할 수 있는 것이다.

IBM 社가 650, 704, 用의 Basic Fortran 을 發表한 以來, 各메이카도 서로 다투어 가면서 自社의 計算機에 使用할 수 있는 Fortran 을 開發하여 왔다. 이때 Compiler 쪽은Hardware 에의 依存性이 强했기 때문에 各社마다 제나름의 特徵을 가진것이 만들어졌지만 Langnage 쪽은 될 수 있는대로 IBM 의 것을 그대로 採用한다는 方針을 세웠던 것이다. 이것은 IBM 의 實績을 利用하겠다는 商業政策의 結果였지만 歷史的으로는 Program Langnage 의 共通化라고 하는 커다란 意義를 남기게 되

었다고 보겠다.

IBM社는 그後 709, 7090이라고 하는 大形計算機用의 Fortran Ⅱ를 發表하였다.

Fortran Ⅱ는 Full Fortran이라고도 불러지고 있으며 Basic Fortran에比하여 Sub-program의 取扱이 包含되고 있어 그 內容도 相當히 擴張된 것이다.

Fortran Ⅲ은 特히 그 進法의 計算機에 對하여 만들어진 것으로 Fortran Ⅱ보다 조금 더 擴張된 部分이 있으나 現在 別로 쓰이지 않고 있다.

Fortran Ⅳ는 가장 擴張된 것으로 Fortran Ⅱ와 함께 現在 가장 많이 利用되고 있는 것이다.

本節에서의 說明도 主로 이 Fortran Ⅳ에 의거하여 하고 있다.

一般的으로 Fortran 言語(要素)는 아래와 같은 열가지로 分類할 수 있다.

1. Constant : 1.235……72等
2. Variable : x. y 等
3. Subscripted Variable : x: yi→x(j) Y(I)
4. Control Statement : Program 實行順序를 變化시키는 것.
5. Arithmetic Statement : 計算을 하는 數式等
6. Input Output Statement : DATA의 移行
7. SpecificationStatement :
8. Sub-Routine Statement :
9. Comment Statement :
10. Type, DATA Statement :

以下 이것들을 簡單하게 說明하여 보겠다.

(A) 定數, 變數 및 添字

Fortran에서는 數字의 定數(Constant), 變數, 添字를 가진 變數等을 表現할 수가 있다.

이들 量을 나타내기 爲한 規則은 普通의 數學的인 表記法과 거의 비슷한 것이다.

다만 이들 量은 各各 아래와 같은 두개의 基本的인 Mode (樣式)으로 나타내게 되는 것이다.

整數計算과 浮動 小數點 計算

Fortrant에서는 整數 (흔히 固定小數點 (Fix-Point)이라고도 함) 計算과 浮動小數點(Float-Point) 計算이라는 두 種類의 演算을 할 수 있음으로 이들 Mode는 極히 重要한 意味를 가지는 것이다.

浮動小數點計算은 10進의 數字 2桁부터 普通 7桁까지의 精度 (但, 計算機에 따라 相違)로 計算하고 있다.

浮動小數點計算의 例를 들면 :

演算 Statement	計算結果
$A=0.4301/1.7$	$A=0.253$
$B=5.0/2.0$	$B=2.5$
$C=1.6*0.7$	$C=1.12$
$D=-2.7+1.2$	$D=-1.5$

이와같이 浮動小數點計算은 "慣習的"인 形式으로 實行되어 一般으로 알기 쉬운 것이다.

但, 이 計算에서는 四捨五入으로 Rounding하지 않고 餘分桁은 그대로 均捨(Trancate) 된다.

한편 整數計算(Fix Point Calculcation)은 上記와 若干 달라 計算은 整數만 取扱하고 小數部分이 計算途中에 保存되는 일은 없다.

例를 들면,

演算 Statement	計算結果
$I=5/2$	$I=2$(2.5의 0.5는 切捨)
$I=5/2+7/2$	$I=5$ (中間의 切捨로 12/2가 아니고 2+3)
$J=5*2$	$J=10$
$K=-4+1$	$K=3$

定 數

定數(Constant)라고 하는 것은 Program이 하나의 實行(Execution)으로부터 다음 實行으로 옮겨가도 아무런 變化없이 計算에 使用되는 數이다.

例를 들면 $J=3*K$라는 Statement에서는 3이 實際의 數로 주어진 定數이다.

Fortran에서는 整定數와 浮動小數點定數(小數點을 가지는 것이 特徵)의 두 種類의 定數를 使用할 수 있다.

다음에 이것을 쓸 경우의 規則을 說明하겠다.

○整定數의 一般形式

整數는 10進法 數字 0 1 2……9를 使用하고 小數點없이 쓴다. 數字앞에 +一符號를 부쳐도 좋다. (만약 符號가 없으면 正數로 取扱함.)

整定數의 크기는 各 Fortran Processor 마다 定해져 있으며 1桁~4桁 10進法 數字면 모든 Processor가 받아 들인다.

좋은 例	못쓸 例
0	−3.2 (小數點때문)
+9	5.0 (小數點때문)
−327	3.3 (小數點때문)
45	289138 (Processor 範圍 Over)

○浮動小數點定數의 一般形式

10進法數字 0, 1, 2……9를 使用하고 小數點을 부친 모든 數. +一符號는 上記와 同一. E를 앞에 가진 整數의 指數(Exponent)는 浮動小數點의 뒤에 부칠 수 있다. +, 一符號는 위 說明과 같음. 또 浮動小數點定數의 크기는 各 Fortran Procesor에 따라 限度가 있으나 普通은 10進法數字 1~8桁로 크기는 10^{-49}~10^{+49}를 많이 쓰고 있다.

浮動小數點定數는 日常 우리가 쓰는 小數點을 가진 10進數 그대로인 것이다. 다만 이것을 表現하는데 두가지

形式이 있다. 한 가지는 우리가 日常 쓰고 있는 表現 그대로 例를들면 0.036, 120.0 138.25……처럼 小數點을 有效數字의 제일 앞이나 맨 마지막 또는 任意로 數字사이에 넣어 그 크기를 나타내는 것이다.

다른 한 가지는 指數形式을 使用하는 것이다. 곧 이것은 有效數의 뒤에 文字 E를 부치고 그 뒤에 Exponent의 符號를 쓰고 다시 Exponent를 10進法으로 2桁까지 쓰는 것이다.

좋은 例	못쓸 例
1.23	4367 (小數點이 없음)
0.0094	5.0 *E* 123 (Processor Over)
5.0*E* 3(5×10^3 곧 5,000)	234.48394798363
5.0*E*+3(5×10^3 곧 5,000)	(桁數 Over)
5.0*E*−2(5×10^{-2}곧 0.05)	
−5.0*E*−3(-5×10^{-3} 곧 −0.005)	

變 數 (Variable)

Fortran의 變數는 어느 값(値)을 假定한 記號로 表現된다. 이것은 Program의 實行內容이 變하거나 Program 內의 段階(Stage)가 變할 경우에 딴 값으로 되어도 좋다.

例를 들면

$$K=3*I$$

라는 Statement에서 K와 I가 變數이다.

I의 값은 前 Statement에서 割當되며 隨時로 그 크기를 變動시켜 나갈 수가 있다. 또 K는 새로운 I에 따라 計算될 적마다 그 크기가 變化하게 된다.

定數의 경우와 마찬가지로 變數도 그것을 나타내는 값이 整數인가 浮動小數點인가에 따라 各各 整數 또는 浮動小數點의 形式을 取한다. 다만 整數値를 가지는 變數와 浮動小數點値를 가지는 變數이름(記號)은 取扱되는 數値의 Mode를 區別하기 위하여 兩者間에 明確한 記名法의 相異를 두고 있다.

○整變數의 一般形式

I.J.K.L.M 또는 N로 始作하는 一連의 英數字(英數字 : 英字 및 數字 A~Z, 0~9)
特殊文字除外 (+S*1.)……等)

좋은 例	못쓸 例
I	PMAX (最初文字가 整數指示 記號 아님)
JOB 1	8 LOX (最初文字가 英字가 아님)
JOB 2	IMIN $ ($는 特殊文字임)
NEXT	IABCDEFG (桁數 또는 字數가 Over)
MAX	(普通 6桁까지)

○浮動小數點 變數의 一般形式

最初가 整數指示記號(I,J,K,L,M,N)를 除外한 英字로 始作하는 一連의 英數字(特殊文字除外) 그 길이 곧 桁數는 各 FORTRAN Processor로 制限됨

좋은 例	못쓸 例
B7	ITRNS (最初文字가 整數指示記號)
DELTA	8 BETA (最初文字가 數字)
VALUE	ACST/ (特殊文字때문)
Z98XY	ADEFLXYZO (字數 Over)
	(普通은 6桁까지임)

위에서 說明한 Mode의 區別은 日常 우리들이 이것에 익숙하고 있지 않기 때문에 若干 奇異한 感을 가지게 될지 모르겠으나 計算機에서는 計算을 整數로 하느냐 또는 浮動 小數點數로 하느냐를 반드시 指定해 둘 必要가 있는 것이다. 그러나 變數에 이름을 부칠 경우에 지켜야 할 規則에도 相當한 융통성이 있다. 一般的으로 利用할 수 있는 限 意味內容을 記號에 包含시키는것이 全體의 計算內容을 理解하는데 便利할 것이다.

例를 들어 速度=距離/時間이라는 計算을 할 때 이것을 $X=Y/Z$ 라고 하는 變數를 使用할 수도 있을 것이다. 조금 더 풀이해서 이것을

Speed=Dist/Time 로 表現한다며

이 計算內容을 理解하기가 容易하게 될 것이다.

다만 이때 各變數가 取扱하는 數値가 浮動小數點일 때에는 이와같은 記號나 이름으로 表現하는데 別問題가 없지만 만일 이것이 整數를 取扱할 때에는 各變數이름의 제일 첫 英文字가 I,J,K,L,M,N의 여섯가지 中의 하나를 쓰도록 規定되어 있기 때물에 上述한 表現을

$I=J/K$ 또는 NSP=LDIS/MTIME와 같이 바꾸어 써야만 되는 것이다. 다시 말하면 變數는 最初의 文字 하나로 그것이 整數인가 또는 浮動小數點인가를 나타내고 그 뒤에 記憶을 하기 쉬운(可能하면 意味까지 있는) 文字를 써줌으로서 意味 內容을 包含하도록 하는 것이 좋을 것이다.

다음에 普通 變數이름은 1~6個의 英數字를 使用하기로 制限되어 있기 때문에 이 範圍內이라면 任意의 文字로 名變數를 表現할 수 있는 것이다.

添字 (Subscript)

排列(Array)라고 하는 것은 1 group의 量을 말한다. 이 group를 하나의 이름으로 나타내고 group內의 個個의 量을 그 group에 있어서의 位置로 나타낼 수 있다면 便利할 때가 많다.

例를 들면 다음의 一連의 數가 NEXT라는 이름의 排列일 때 (註 NEXT이니까 整數만 取扱함)

15, 12, 16, 18, 19, 23

이 group의 두번째에 있는 數(곧 여기서는 12)를 指定하고 싶을 때 普通의 數字的 表記法에서는 이것을 $NEXT_2$ 라고 쓰지만 FORTRAN에서는 NEXT(2)라고 表示한다.

"2"라는 量을 添字(Subscript)라고 한다.

따라서 NEXT(2)는 12라는 값을 가지고 NEXT(4)는 18이라는 값을 가진다.

이것을 더 一般的으로 하여 NEXT(I)라 두고 I를 指定함으로서 上記 數字를 찾아 쓸 수 있도록 하고 있다.

排列은 二次元的으로도 表示할 수 있다.

例를 들어 지금 아래와 같은 排列 PMAX가 있다고 한다.

	第 1 列	第 2 列	第 3 列
第 1 行	3.6	2.8	5.4
第 2 行	1.7	1.9	3.6
第 3 行	5.2	16.3	7.6

이때 PMAX(2,3)은 第2行 第3列의 3.6이 될 것이며 PMAX(3,2)는 第3行 第2列의 16.3이 될 것이다.

一般的으로는 이것도 PMAX (I,J)로 하여 I, J를 指示함으로서 任意의 排列內의 數値를 指定使用할 수 있게 되어 있다. 3次元 排列의 경우도 꼭 마찬가지이다.

○添字의 一般形式

添字는 다음 形式의 어느 것이라야만 된다.

V
C
V+*C* 또는 *V*−*C*
C✻*V*
C✻*V*+*C′* 또는 *C*✻*V*−*C′*

但, *V*는 無符號로 添字를 가지지 않는 任意의 整變數를 나타내고 *C* 및 *C′*는 任意의 無符號 整定數를 나타낸다.

特히 注意해야 할 點은 添字에 浮動小數點量을 使用못하게 되어 있고 또 定數에 符號를 부쳐서는 안된다는 것이다.

다음 排列(Array)은 그것이 Program 안에서 쓰여지기 前에 미리 그 크기를 指定해 두지 않으면 안되게 되어 있다.

가령 A(I)라는 Array가 있어 添字 I가 最高 100 까지의 값을 가진다고 하면 A라는 이름으로 불러지는 Array의 크기는 100個로 限定될 것이다.

이러한 경우에는 後述하겠지만 Dimension 이라는 Statement를 使用하여 미리 이 A排列의 크기를 宣言하도록 하고 있다. Arry를 使用한 簡單한 Statement 例를 다음에 몇가지 들어 보겠다.

VALUE (1)=1.0
VALUE (2)=5.9
VALUE (3)=3.6
VALUE (4)=2.17
VALUE (5)=15.0

지금 VALUE라고 하는 이름을 가진 Array의 크기가 미리 指定(宣言)되어 있다고 한다.

이 Statement에 의하여 VALUE라고 하는 Table에는 1.0, 5.9, 3.6, 2.17, 15.0, 이라는 값이 順次 들어가게 된다. 따라서 다음에

A=VALUE(1)+VALUE(2)

라고 하면 A라는 Variable은 1.0, +5.9, =6.9, 라는 數値를 가지게 되는 것이다.

다음에 添字에 Variable을 使用할 例로서

J=1
VALUE(J)=VALUE(J+1)+VALUE(J+2)

라고 하면, VALUE라고 하는 Array의 첫번째의 둘째번과 셋째번의 內容을 모댄 것 곧 5.9+3.6=9.5가 들어가게 된다는 것이다. 다음에 簡單한 例를 하나 더 들어 보겠다.

좋은 例	못쓸 例
P(IMAX)	P(−I) (變數는 無符號라야 함)
P(I 9)	P(A+2) (A는 整變數아님)
TOL(JOB+2)	TOL(I+2.) (2.는 整定數아님)
TOP(NEXT−3)	TOP(−2*)(定數는 無符號라야 함)
XEN(8✻IQUAN)	XEN(I(3)) (添字에 添字는 못 부침)
NIN(5✻L+7)	MIN(K✻2)(乘算일 때 定數✻變數라야 함)
MAX(4✻M−3)	MAX(2+JOB) (加算일 때 變數+定數의 順)

(B) 演算 Statement와 式

演算 Statement (Arithmetic Statement)

演算 Statement는 數値計算을 指定하는 것으로 普通의 演算式과 거의 비슷한 表現으로 되어 있다.

○演算 Statement의 一般形式

"*a*=*b*" 但, *a*는 變數(添字를 가질 수도 있다)이며 *b*는 以下에 定義하는 式 (Expression) 이다.

例 *A*=*B*+*C*
D(*I*)=*E*(*I*)+2−*F*

FORTRAN 演算 Statement에서는 等號 "="의 意味는 "~와 같다"라고 하는것 보다도 오히려 "~에 依하여 置換된다"라고 理解하여야 할 것이다.

上記 例에서 A=B+C는 A가 B+C와 같다는것 보다도 B+C의 內容이 A의 內容으로 置換된다는 것이다. 이와 마찬가지로 A=3.0라면 A에 3.0을 Store 한다는 것이며 NEXT=3+MIN은 MIN+3을 NEXT에 Store 한다는 것이다.

다시한번 注意하건데 "="는 普通의 意味와 달라 右邊의 計算結果로 左邊의 變數의 값을 置換하라는 意味를 가지는 것이다.

式(Expression)

FORTRAN에서 말하는 式(Expression)이라는 것은 定數, 變數(添數를 가질수도 있음) 및 演算記號를 連結

하여 어느 量 또는 一連의 計算을 指示하는 것이다.

이것은 式을 만들 境遇의 規則에 따라 쓰여져야만 되는 것이며 또 여기에서는 Comma나 괄호와 같은 것도 함께 使用할 수도 있고 또 函數 (function: 後述)도 포함 시칼 수 있다.

演算記號

演算記號 (Operation Symbol)에는 다음과 같이 操作記號를 使用하여 加減乘除算 및 累乘算의 計算을 할 수가 있다.

+	加 算
−	減 算
*	乘 算
/	除 算
**	累乘算 A**B=A^B

式을 쓰는 方法

定數, 變數 및 添字付 變數는 整數量이거나 浮動小數點量이 될 수 있으므로 式에는 整數量 浮動小數點量을 같이 包含시칼 수 있다.

다만 이들 두가지 形의 量을 하나의 式에 表現시칼 때에는 몇가지 規則이 있다.

i) 가장 簡單한 式은 하나의 定數, 變數 또는 添字付 變數로 이루어진다. 그것이 整數量이면 그 式은 整數 Mode라고 말하며 浮動點小數의 量이면 浮動小數點 Mode라고 말한다.

例

式	量의 種類	式의 Mode
I=3	整定數	整數
F=3.0	浮動小數點定數	浮動小數點
M=I	整變數	整數
F=A	浮動小數點變數	浮動小數點
M=I(J)	添字付整變數	整數
F=A(J)	〃 浮動小數點變	數浮動小數點

마지막 側에서 添字——반드시 整數量이라야만 함——은 式의 Mode에 無關係라는 點을 注意해 두고 싶다. 式의 Mode는 量 그 自身의 Mode에 依하여 決定되는 것이다.

ii) 어느 量을 累乘하여도 그 量의 Mode는 變하지 않는다.

한편 整數量에 浮動小數點指數를 줄 수는 없다.

좋은 例	못쓸 例
J** I 整數	I**A (整數量에 浮動小數點指數
A** I 浮動小數點	를 못 주기 때문)
A**B 〃	

iii) 다음 條件을 滿足하고 있으면 量의 앞에 + 또는 −를 부칠수가 있고 또 量과 量을 演算子(+, −, *, /, **等)로 結付시킬 수 있다.

(a) 두개의 演算子가 連續되지 않을 때.

(b) 이와 같이 結付되는 數는 반드시 同一 Mode라야 한다. (但, 例外로서 浮動小數點量은 固定小數點指數를 가질수 있다)

좋은 例	못쓸 例
−A+B	A+−B (演算子가 連續임)
B+C−D	A+I (變數의 Mode가 다름)
I/J+M	3J (乘算이라면 (3*J로함)

iv) 式을 만들기 위하여 괄호를 使用하여도 式의 Mode는 變하지 않는다.

따라서 A (A) (A)는 모두 浮動小數點式이다.

v) 式中에 있어서의 演算順位를 指定하기 爲하여 괄호를 使用할 수 있다. 괄호가 없을 때에는 다음의 順番으로 왼편부터 오른편으로 計算되어 간다.

Operation	Weight
累 乘 **	3
乘算및除算 * 및 /	2
加 減 算 + 와 −	1

Weight의 큰것부터 먼저 計算되는데 괄호는 이 순서보다도 한칭 더 優先的으로 處理된다.

例 $A+B*C/D+E**F-G$

$=A+\frac{B*C}{D}+E^F-G$ 가 되며

$(A-B)*C/D+E**F-G=\frac{(A+B)*C}{D}+E^F-G$

가 된다.

이와 같이 괄호와 ±, *, / 및 **는 普通하는 計算法則과 마찬가지로 같은 세기(强度)의 것은 왼편부터 計算하게 된다.

그러나 될수 있는대로 괄호를 使用하여 計算順序를 틀리지 않도록 하는것이 좋을 것이다.

以下에 몇가지 例를 들어 보겠다.

$A=B+C/D**E*(F+2.0)$의 計算順은

(가) $F+2.0$

(나) D^E

(다) C/D^E

(라) $\frac{C}{D^E}(F+2.0)$

(마) $B+\frac{C}{D^E}(F+2.0)$

(바) $B+\frac{C(F+2.0)}{D^E}\rightarrow A$에 Store

다음에 괄호의 有無 Operation의 順位에 따른 計算順에 關한 몇가지 例를 들어 보겠다.

$5*(J+3)**2/12=\frac{5(J+3)^2}{12}$

$5*J+3**2/12=5J+\frac{3^2}{12}$

49페이지에 계속