OUKA
The University of Osaka
Institutional Knowledge Archive

Title	FORTRANとの比較によるPL/Ｉ入門 (5)
Author(s)	塩見, 充
Citation	大阪大学大型計算機センターニュース. 1981, 40, p. 91-110
Version Type	VoR
URL	https://hdl.handle.net/11094/65476
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

FORTRANとの比較によるPL／1入門(5)

大阪大学工学部　塩野　充

８０年代の夜明けである１９８０年もあっという間に終り、１９８１年の正月（これを書いている時点が）となった。大みそかの恒例の紅白歌合戦なるものに全く興味を失ってしまってから久しいが、結果だけ聞くところによると今回も紅組が勝ったとのこと、最近は女性上位か女性迎合かは知らないが毎年紅組が勝っているようである。しかし紅白いずれが勝っても負けてもどうということはないが、あのような番組に莫大な費用が湯水のように使われているのを見ると、受信料を払うのが馬鹿々々しいといおうか、情ない気持ちになるのはへそまがりの筆者一人だけではないようである。このような感じ方をするのは年のせいだと言われそうな気もするが、参考までに書くと筆者の年令は四捨五入して三十才である。しかし、紅白歌合戦を毎年楽しみに見ておられる方々にケチをつけるつもりは毛頭なく、筆者の個人的な偏見（？）を述べたにすぎない。

ところで私事で恐縮だが、筆者は年末から年始にかけて中国の上海，南京付近をぶらついてきた。大みそかの夜は蘇州という所にある寒山寺という寺の除夜の鐘を聞いた。この寺は中国に少し興味のある人なら知っている有名な寺で、日本のテレビでも除夜の鐘が放映されたことがある。この寺の除夜の鐘は筆者が行ったとき、夜の１１時４０分から１２時までのわずか２０分間だけ撞かれた。２０分だからとても１０８つは撞けない。それが中国の定まった風習なのか、あるいは以下略という、いかにも中国人のいわゆる大陸的な、小さいことにこだわらないという考え方から来ているのか、あるいは数えるのが面倒なのか、理由は分からないが日本の除夜の鐘のように１０８つも撞かなかったことはたしかである。

ところで、大みそかの除夜の鐘を聞いていていつも思うことは、こういう古いものは日本のコンピュータ社会がいかに進んでも変わることはないだろうということである。しかし、ふと考えついたことがある。最近は趣味としてのマイコンが急速に普及している。お寺の坊さんがマイコンを趣味としても決して不思議ではなくなっている。だからマイコン内蔵の釣鐘を作る坊さんも現われるのではないだろうか。筆者が坊さんなら、撞いた回数をカウントして釣鐘の側面に発光ダイオードか何かでデジタル表示する装置を作るだろう。あるいはもっと凝って釣鐘を撞く棒（撞木）にモーターか電磁プランジャをつないで、セットした時間が来れば無人で鐘を撞く装置を作り、自分は暖かい部屋で高いびきをかいているだろう。更には高精度なツインクォーツの時計でも内蔵させて、毎年大みそかの何時何分何秒になれば除夜の鐘がスタートするようプログラムしておけば何十年間も釣鐘を撞く必要がなくなる。名付けて、プログラム釣鐘（？）である。スピーカーから録音テープを流す方法は音質が本物には及ばないし、第一、有難味がなさすぎる（？）。

閑話休題（英語ではTo return to the main subjectというそうな）。今回はON文と構造体について解説しよう。

第16章　ON文

（16－1）FORTRANのENDパラメータ

JISを越えたFORTRANの入出力文にENDパラメータとERR（エラー）パラメータというのがある。ENDパラメータというのは例えば、

```
      READ(5, 100, END=50)  A
100   FORMAT(F7.3)
      ..................
 50   ..................
```

というREAD文では、EOF（End of file, ファイル終了記録）を検出すると、文番号50へ飛ぶ。この場合はファイルコード5でカードリーダゆえ、読み込むべきカードがなくなっているのにまだ読み込もうとした場合に文番号50へ飛ぶ。

ERRパラメータというのは例えば、

```
      READ(10, 200, ERR=60)  B
200   FORMAT(F7.3)
      ..................
 60   ..................
```

というREAD文では、ハードウェア的なエラー（磁気テープ装置などでテープが古かったり、ヘッドが汚れていたりすると発生することがある）を検出すると文番号60へ飛ぶ。

ENDパラメータとERRパラメータは同時に指定してもよい。又、順序はどちらが先でもよい。例えば、

```
      READ(10, 300, END=70, ERR=80)  C
300   FORMAT(F7.3)
      ..................
 70   ..................
      ..................
 80   ..................
```

などと書ける。ENDパラメータとERRパラメータはFACOM OSⅣ／F4 FORTRANではREAD文にしか書けないが、ACOS－6 FORTRANではERRパラメータの方だけはWRITE文にも書ける。例えば、

```
      WRITE（10，400，ERR＝90）  D
400   FORMAT（F7．3）
      ……
 90   ……
```

というWRITE文では書き込み中にハードウェア的なエラーを検出すると文番号90へ飛ぶ。ENDパラメータはWRITE文には書けない。

ERRパラメータで注意すべきことは、読み込み失敗や書き込み失敗のようなハードウェア的なエラーに対してしか効力をもたないことで、データの型とFORMATが一致していないというようなソフトウェア的なエラーには効力をもたないということである。

ENDパラメータやERRパラメータを書いていないとき、EOFの検出やハードウェア的なエラーの検出が起こるとプログラムの実行は終了してしまう。こういう意味で、ENDパラメータとERRパラメータはFORTRANプログラムにおいて、実行中に不測の事態が発生したときの処置をプログラマーが指定できる、たった2つの場合である。

（16－2）コンディション

PL／Iでは上記のような、実行中における不測の事態に対する処置の指定がもっと沢山用意されている。それがON文である。ここで、"不測の事態"と書いたがこれをPL／Iでは、コンディション（Condition，条件と訳している本やマニュアルがあるが筆者の私見ではむしろ状態と訳した方がぴったりくるのではないかと思う）と呼ぶ。前述のFORTRANの入出力文ではコンディションにあたるものはENDとERRの2種類だけであったが、PL／Iには様々なコンディションがある。コンディションが発生することを、コンディションが立てられた（raised）ともいう。コンディションには、"割込み可能（enable）"と"割込み不能（disable）"の2つの状態がある。あるコンディションが割込み可能のとき、そのコンディションが発生すると中断が生じる。中断とは何らかの処置がなされるための一時停止である。又、割込み不能のときは、そのコンディションが発生しても中断も生じないで、いわば素通りの形になる。コンディションには、常に割込み可能のものや、プログラムによって割込み可能と割込み不能をどちらにでも切換えることができるものがある。表16－1に主なコンディションを分類して示す。これ以外にも種々のコンディションがあるが初心者にはこれで十分であろう。常に割込み可能なコンディションはプログラムによって割込み不能にすることはできない。

表16－1の各コンディションの意味について以下に説明しよう。コンディション名は全て省略形で示す。

表１６－１　主なコンディションの分類（〔 〕内はコンディション名の省略形である）

機能分野	常に割込み可能なコンディション	切換えできるコンディション	
		省略時解釈では割込み可能なコンディション	省略時解釈では割込み不能なコンディション
演算関係		①CONVERSION〔CONV〕 ②FIXEDOVER-FLOW〔FOFL〕 ③OVERFLOW〔OFL〕 ④UNDERFLOW〔UFL〕 ⑤ZERODIVIDE〔ZDIV〕	⑥SIZE ⑦STRINGSIZE〔STRZ〕
入出力関係	⑧ENDFILE（ファイル名） ⑨ENDPAGE（ファイル名） ⑩UNDEFINED-FILE（ファイル名）〔UNDF（ファイル名）〕 ⑪NAME（ファイル名）		
デバッグ関係			⑫SUBSCRIPTRANGE〔SUBRG〕 ⑬CHECK（……）
システム動作関係	⑭ERROR		

①CONV

このコンディションは文字列を算術型やビット型に変換しようとするときに、変換できない文字列である場合に発生する。すなわち、数字以外の文字を含む文字列を算術型に変換しようとしたり、0，1以外の文字を含む文字列をビット型に変換しようとする場合に発生する。次のような例である。

```
DCL A CHAR(4) INIT('A123');
DCL B FIXED BIN;
B=A;
```

②FOFL

固定小数点の数値の演算でオーバーフローしたとき発生する。絶対値がどれくらいの大きさを越えるとオーバーフローするかはシステムによって異なるが、2進固定小数点（FIXED BIN）の場合、ACOS－6のTYPE（A）では71桁，（AB）でも71桁，FACOM OSⅣ／F4では31桁である。10進固定小数点（FIXED DEC）の場合、TYPE（A）では59桁，（AB）では31桁，OSⅣ／F4では15桁である。

③OFL

浮動小数点の数値の演算でオーバーフローしたとき発生する。絶対値がどれくらいの大きさを越えるとオーバーフローするかはシステムによって異なるが、TYPE（A），（AB）では、10^{127}，OSⅣ／F4では10^{75}である。

④UFL

浮動小数点の数値の演算でアンダーフローしたとき発生する。絶対値がどれくらいの小ささ（?）を下回るとアンダーフローするかはシステムによって異なるが、TYPE（A），(AB）では10^{-128}，OSⅣ／F4では10^{-78}である。なお、ここで注意すべきことは、例えば、$x=y^{-z}$（$y>0$，$z>0$ とする）を計算するとき、

$$x=y^{-z}=1/y^{z}$$

ゆえ、y^{z}を計算してその逆数をとる計算が行われるので、yやzが大きいとy^{-z}でUFLコンディションが発生すると思いきや、y^{z}でOFLコンディションが発生することがある。

⑤ZDIV

0で割算をしようとすると発生する。例えば、

```
A=0．0；
B=5／A；
```

とすると発生する。

以上の②～⑤のコンディションはACOS－6　FORTRANの¥FORTRANオプショ

ンにおけるＦＬＴＣＨＫオプションに類似しているが、ＦＬＴＣＨＫオプションではエラー発生箇所としての行番号等の出力をして終了してしまうだけであるが、これらのコンディションでは後述するＯＮ文により、プログラマーの望むように処置できる。

⑥ＳＩＺＥ

代入や入出力において上位の桁が落ちると発生する。ＦＯＦＬとの違いは、ＦＯＦＬではシステムの最大精度を越えるとき発生するが、ＳＩＺＥはシステムの最大精度ではなくて、ＤＣＬ文で宣言した精度を越えると発生する。例えば、

```
DCL A FIXED BIN(2);
A=1234;
```

などとすると発生する。

⑦ＳＴＲＺ

長い文字列を短かい文字列宣言をした変数に代入したときに発生する。例えば、

```
DCL A CHAR(4) INIT('ABCD');
DCL B CHAR(2);
B=A;
```

などとすると発生する。結果は左から詰められて、Ｂ＝'ＡＢ'となる。

⑧ＥＮＤＦＩＬＥ（ファイル名）

前述したＦＯＲＴＲＡＮのＲＥＡＤ文におけるＥＮＤパラメータに相当するものである。使用法は後述のＯＮ文のところで説明する。

⑨ＥＮＤＰＡＧＥ（ファイル名）

ラインプリンタ等で、ページの終りに来たときに発生する。ＰＡＧＥＳＩＺＥ（１ページ当りに印刷する行数）が例えば６０としてあれば、６１行目に来たときにこのコンディションが発生する。このコンディションの使用法も後述のＯＮ文のところで説明する。

⑩ＵＮＤＦ（ファイル名）

ファイルのオープンがうまくいかなかったときに発生する。ＤＣＬ文でのファイルの定義などに誤りや矛盾がある場合に発生する。

⑪ＮＡＭＥ（ファイル名）

このコンディションは第２回の第８章（８－２）＜Ｃ＞のところで少し触れたが、ＤＡＴＡ型入力文に関連するものである。例えば、カード上に、

```
S=1.8, T=3, U=5.5;
```

とパンチされているデータを、

```
GET DATA(S, T);
```

で読み込むと、Uの入る場所がないので誤りとなり、NAMEコンディションが発生する。すなわち、DATA型のGET文によって読み込まれるデータが不適当である場合に発生する。

⑫SUBRG

配列の添字がDCL文で宣言した範囲を逸脱しているときに発生する。例えば、

```
DCL  A(10);
A(11)=0.0;
```

などとすると発生する。これは、¥FORTRANオプションのSUBCHKオプションに類似しているが、SUBCHKの場合はエラー発生箇所の行番号等を出力して終了してしまうが、SUBRGコンディションでは後述のON文によってプログラマーの望む処置が行える。

⑬CHECK（変数名等）

このコンディションはACOS－6　FORTRANのトレースパッケージや、OSⅣ／F4　FORTRAN　GEのDEBUG文に相当するもので、プログラムのデバッグに有力なコンディションである。CHECKの後のカッコ内には、変数名，配列名，文の名札，手続き名を書くことができる。CHECKコンディションが発生するのはそれぞれ次のような場合である。変数に関しては、代入文によって値が変ったり、GET文によって新しい値が与えられたり、DOの制御変数として値が変化するときなどに発生する。配列に関しては、配列の１つの要素でも上述の変数のように変化したときに発生する。文の名札に関しては、その名札の付いている文の直前に来たときに発生する。手続き名に関しては、その手続きが呼び出される直前に発生する。

CHECKコンディションが発生したときの標準システム動作は、変数や配列要素の場合は、それらの値をDATA型PUT文によって出力する。文の名札や手続き名の場合はそれらの名前を出力する。その後は実行を続ける。

CHECKの後のカッコを省略して、単にCHECKと書いた場合は、全ての変数名，配列名，文の名札，手続き名が対象となる。CHECKコンディションに関しては、発生したときの標準システム動作が非常に親切なので、後述するON文を用いて、発生したときの処置をあえてプログラマーが指定する必要はあまりないと思われる。ただ、印刷量が膨大になるおそれもあるので、それを防ぐために、出力するときの制限を付加して印刷量を抑える方がよいこともある。

⑭ERROR

プログラムの実行中に、対応するコンディションのないエラーが起きたときに発生する。又、前述のコンディション①～⑥，⑧，⑩，⑫が発生したとき、ON文による処置の指定が省略されていれば、引続きこのERRORコンディションが発生し、標準システム動作としてプログラムの実行を打切る。

（１６－３）ＯＮ文

ＯＮ文は、あるコンディションが発生したときの処置を指定する文である。例えば（１６－１）に示したＦＯＲＴＲＡＮのＥＮＤパラメータの例をＰＬ／Ｉで書くと、

```
ON ENDFILE(SYSIN) GO TO L50;
GET EDIT(A) (F(7, 3));
.....
L50:.....
.....
```

となる。オーバーフローしたときラベルＬ１の文へ飛ばしたい時は、

```
ON OFL GO TO L1;
```

と書く。ゼロ除算が起きたとき、ラベルＬ２の文へ飛ばしたい場合は、

```
ON ZDIV GO TO L2;
```

と書く。これらの例におけるＧＯ　ＴＯ文の部分をＯＮ単位（ＯＮ－ｕｎｉｔ）と呼ぶ。ＯＮ単位はＧＯ　ＴＯ文に限らず、他の単一の実行文でもよい。例えば、

```
ON FOFL N=N+1;
```

などとも書ける。ＯＮ文を一般的に書くと、

```
ON コンディション名 ON単位;
```

となる。ＯＮ単位として次の４種類の場合が考えられる。

＜ａ＞単一の実行文

これは前述した例の通りで、ＧＯ　ＴＯ文その他、単一の実行文が書ける。

＜ｂ＞複数の実行文

ＯＮ単位が単一の実行文で書けない場合は、ＢＥＧＩＮブロックを用いて複数の実行文を一まとめにして書く。例えば、

```
ON CONV BEGIN;
        PUT DATA(X, Y);
        N=N+1;
        END;
```

などと書ける。この場合，ＢＥＧＩＮ文にラベルを付けることはできない。又、複数の実行文を一まとめにするのに、ＩＦ文のＴＨＥＮ節やＥＬＳＥ節で用いた繰返しなしのＤＯ文（単純ＤＯ文）を用いることはできず、ＢＥＧＩＮブロックを用いなければならない。

＜ｃ＞空　文

ＯＮ単位が空文（すなわちセミコロンのみ）の場合は、そのコンディションが発生しても何の

処置もしないことを表わす。これは例えば次のような場合に用いられる。ラインプリンタで印刷する場合、TYPE（AB）やOSⅣ／F4では、SYSPRINTの標準のPAGESIZEが60行であり、実際のラインプリンタ用紙は普通1ページ66行なので各ページの下の6行がいつも空白になってしまう。これは行番号のカウンタが61になったとき、ENDPAGEコンディションが発生して、標準システム動作として自動的に改ページが行われ、行番号のカウンタが1に戻されるためである。従って、1ページに66行フルに印刷したいときは、次のようにすればよい。

```
ON ENDPAGE（SYSPRINT）；
```

すなわち、ON単位が空文（；）なので、SYSPRINTでENDPAGEコンディションが発生しても何も処置が行われない（勿論、標準システム動作も行われない）ので、改ページも行われず、66行全て印刷される。この場合、行番号のカウンタはそのページを過ぎても1には戻らず、67，68，69，……と増えてゆく。

＜d＞SYSTEM

ON単位にSYSTEMと書くと、標準システム動作を行うことを指示する。標準システム動作は、ON文を省略したときに行われるのだから、わざわざON単位にSYSTEMと書くようなON文ならはじめから書かないで省略すればよいではないか、と思われるかも知れない。しかし、あるコンディションに関して最初に1つのON単位を設定しておいて、途中から標準システム動作に戻したい場合にこれが必要となる。例えば、

```
ON UFL K=K+1；
……
ON UFL SYSTEM；
……
```

と書くと、最初はアンダーフローが起こるとKに1を加えるが、2つ目のON文以降はアンダーフローに対して標準システム動作を要請する。ON文は宣言文ではなくて実行文なので、このようにその位置が重要な意味をもつ。

以上が4種類のON単位の説明である。ON文は前述したようにその位置に気を付けなければならない。ON文は、そのコンディションが発生しそうな部分より前になければならない。コンディションが発生した部分の後にON文があっても意味がない。例えば、

```
DCL X（50）；
I＝100；
X（I）＝0．0；
ON SUBRG PUT LIST（I）；
```

とした場合、配列破りが起こっているが、ON文は働かないのでＩは印刷されない。これは

```
DCL X(50);
ON SUBRG PUT LIST(I);
I=100;
X(I)=0.0;
```

とすればよい。

ON文において、コンディション名とON単位の間に、ＳＮＡＰというキーワードを入れておくと、そのコンディションが発生したとき、起動しているブロック（メインプログラムと、現在呼び出されている状態のサブプログラム）の一覧表などのメッセージがＳＹＳＰＲＩＮＴに出力される。例えば、

```
ON FOFL SNAP K=K+1;
```

などと書く。ＳＮＡＰはON単位が実行されたことを確認するのに便利である。

（１６－４）コンディション接頭語

表１６－１に示したように、コンディションには割込み可能と割込み不能の２つの状態を切換えることのできるものがある。これらのコンディションにおいて実際に切換えを行うには、コンディション接頭語（Condition prefix）なるものを用いる。

例えば、ＳＵＢＲＧは省略時解釈では割込不能なので、これを割込み可能にするには、コンディション名をカッコで包んだもの、すなわち、（ＳＵＢＲＧ）をＰＲＯＣ文の名札の前にコロンを介して付ければよい。例えば、ＥＸ１という名のメインプログラムならば

```
(SUBRG):EX1:PROC OPTIONS(MAIN);
```

となる。但し、この場合、（ＳＵＢＲＧ）はメインプログラムＥＸ１だけにしか効かないので、サブプログラムにも（ＳＵＢＲＧ）を効かせたい場合は、サブプログラム毎に同様のコンディション接頭語を付けなければならない。例えば、サブプログラム名をＳＢ１とすると、

```
(SUBRG):SB1:PROC(.....);
```

となる。この場合勿論，ＳＢ１でＳＵＢＲＧを割込み可能にしなくてもよいのなら書く必要はない。

上例のようにＰＲＯＣ文の前に（ＳＵＢＲＧ）を付けると、その手続きブロック全体でコンディションＳＵＢＲＧが割込み可能になるが、そうではなくて１つの文だけについてＳＵＢＲＧを割込み可能にしたい場合は、その文の前に（ＳＵＢＲＧ）を付ければよい。例えば、

```
.....
(SUBRG):A(I)=1.23;
.....
```

などとする。その文にラベル（例えばＬ５）が付いているときは、更にその前に付けて、

```
                    ……………………
（SUBRG）：L5：A（I）＝1。23；
                    ……………………
```

とすればよい。手続きブロック全体ではないが、複数個の文に渡ってＳＵＢＲＧを割込み可能にしたいときは、ＢＥＧＩＮブロックを用いる。例えば、

```
                    ……………………
（SUBRG）：BEGIN；
          A（I）＝I＊＊2；
          B（J）＝J＊＊2；
          C（K）＝K＊＊2；
          END；
                    ……………………
```

というように書く。ＢＥＧＩＮブロックにしなくても、各文の前に全て（ＳＵＢＲＧ）を付けてもよいが、沢山になると大変である。ＢＥＧＩＮブロックが長くなるとラベルを付けた方が見やすい。すなわち、

```
                    ……………………
（SUBRG）：LB1：BEGIN；
                A（I）＝I＊＊2；
                ……………………
                ……………………
                ……………………
                END　LB1；
                ……………………
```

などと書く。前述のＯＮ単位にＢＥＧＩＮブロックを使うときはラベルを付けてはいけなかったが、ここのコンディション接頭語のＢＥＧＩＮブロックの場合はラベルを付けてもかまわない。この点、混同しやすいから注意されたい。

以上、コンディションＳＵＢＲＧを例にとって説明したが、他のコンディションについても全く同様である。この逆に、省略時解釈では割込み可能のコンディションを割込み不能に切換えたいときは次のようにする。例えば、コンディションＵＦＬでは、

```
（NOUFL）：EX1：PROC　OPTIONS（MAIN）；
```

というように、ＵＦＬの前にＮＯをくっつける。ＰＲＯＣ文でなくて、単一の文やＢＥＧＩＮブロ

ックに付ける場合も全く同様である。UFLを例にとったが、他のコンディションでも同様で、とにかくコンディション名の頭にNOをくっつければよい。NOCONV，NOFOFL，NOOFLなどとなる。（NOとコンディション名の間に空白を入れてはいけない）。

省略時解釈で割込み可能のコンディション（UFLなど）を割込み不能に切換えるときだけでなく、もともと省略時解釈で割込み不能のコンディション（SUBRGなど）を割込み可能にしたブロックの中で更に一部分を割込み不能にすることもコンディション名にNOをくっつけることによってできる。文章では分かりにくいが、次の例を見ればお分かり頂けよう。

```
(SUBRG)：P1：PROC；
                ……… ⓐ
(NOSUBRG)：BEGIN；
                ……… ⓑ
                END；
                ……… ⓒ
                END P1；
```

区間ⓐとⓒではコンディションSUBRGは割込み可能であるが、区間ⓑでは割込み不能である。

なお、当然のことながらENDFILEなどの、常に割込み可能なコンディションにはNOを付けることはできない。

¥FORTRANオプションで、SUBCHKを入れると処理時間が長くなるのと同様に、SUBRGなど、省略時解釈で割込み不能となっているコンディションは、割込可能にすると処理時間を喰うので、デバッグ段階だけで使用し、デバッグが済めばコンディション接頭語をはずして割込み不能に戻しておく方がよい。

コンディション接頭語は2つ以上一緒に書いても構わない。そのときはカンマで区切る。例えば、

```
(SUBRG，CHECK（A，B，C），SIZE，NOOFL，NOUFL）：EX1：
                                                PROC；
```

などと書ける。CHECKの後のカッコ内のA，B，CはCHECKコンディションを効かせたい変数名，配列名，文の名札，手続き名のいずれかである。

（16－5）SIGNAL文

SIGNAL文とは、デバッグのために、あるコンディションが発生していないのに、あたかも発生したかのような状況を作り出す文であり、ON文の働きをテストするための文である。いわば、火事や地震の避難訓練のときの警報のようなものである。一般的には次のような形である。

```
SIGNAL　コンディション名；
```

例えば、

```
SIGNAL SUBRG;
SIGNAL CONV;
SIGNAL SIZE;
```

などである。

（16－6）REVERT文

前述したようにコンディション接頭語は、それをPROC文に付けると、その手続き内のみで有効であり、その手続きが呼び出す他の（外部）手続き、すなわちサブプログラムまでは効力が及ばず、有効にするためにはサブプログラム毎にコンディション接頭語を付けなければならない。これに対して、ON文はサブプログラムにまでも効力が及ぶ。従って、メインプログラムにON文を書いておけば、サブプログラムには同じON文を書く必要はない。しかし、サブプログラムの途中の一部分で、異なるON文を使用し、そこを過ぎると又、メインプログラムのON文が有効になるように戻したいときなどに、REVERT文が使われる。

```
EX1：PROC OPTIONS（MAIN）;
      ON FOFL PUT LIST（X）;
      ……………
      CALL SB1;
      ……………                  ⓐ
      END;
SB1：PROC;
      ……………
      ON FOFL;
      ……………                  ⓑ
      REVERT FOFL;
      ……………
      END;                       ⓒ
```

この例では、区間ⓐとⓒでコンディションFOFLが発生すると、Xの値を印刷するが、区間ⓑでFOFLが発生しても何もしない。このように、REVERT文の一般形は、

```
REVERT コンディション名;
```

であり、その働きはON文を元のON文に戻す役割りである。

第１７章　構造体

ＦＯＲＴＲＡＮには、スカラを表現するのに用いる変数と、ベクトルやマトリクスなどを表現するのに用いる配列という２種類のデータ構造があるが、ＰＬ／Ｉには更に、構造体（Ｓｔｒｕｃ－ｔｕｒｅｓ）という、ツリー状のデータを表現するのに便利なデータ構造がある。しかし、このようなデータ構造は一般の科学技術計算を行うＦＯＲＴＲＡＮユーザにはあまり使われる機会が多いとも思われず、どちらかと言えば事務計算向きかも知れない。ただ、科学技術計算でもリスト処理をするような特殊な場合は有用である。従って、ここでは構造体について簡単に説明するにとどめる。

（１７－１）構造体の概要

図１７－１にツリー構造の例としてプログラム言語の分類ツリーを示した。（単なる例ゆえ、厳密なものではない）。

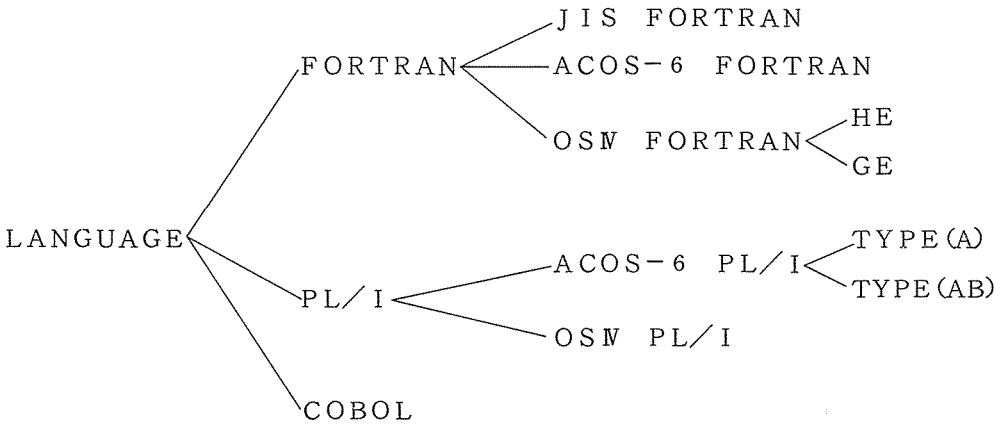

図１７－１　ツリー構造の例（プログラム言語）

これを構造体宣言として書くと、

```
DCL  1  LANGUAGE,
          2  FORTRAN,
             3  JIS_FORTRAN,
             3  ACOS_6_FORTRAN,
             3  OS4_FORTRAN,
                4  HE,
                4  GE,
          2  PL1,
             3  ACOS_6_PL1,
                4  TYPE_A,
                4  TYPE_AB,
             3  OS4_PL1,
          2  COBOL;
```

となる。1，2，3，4という数字はツリーの深さを表わしており、レベル番号と呼ぶ。レベル番号が大きい程、レベルが低いという。レベル番号の次には空白を1つ以上置く。書き方は必らずしも上例のようにレベルの深さによって右へずらせる必要はなく、改行もせずにぎっしり詰めて書いてもよいし、レベル番号を右へずらさずに同じカラムに揃えて書いてもよい。しかし、見易さから言えば上例のようにレベル番号を右へずらして書くのがベストであろう。

レベル1の構造体を主構造体（大構造体，major structure）といい、レベル2以下の構造体を副構造体（小構造体，従構造体，minor structure, substructure）という。上例では、LANGUAGEが主構造体で、それ以外が副構造体である。構造体のレベルの許される深さは、TYPE（A），（AB）では63まで、OSⅣ／F4では15までである。

構造体宣言の中にデータの型宣言を書くことができる。（書くのが普通である）。例えば、

```
DCL  1  A,
          2  B  FLOAT  DEC,
          2  C  FIXED  BIN,
          2  D,
             3  E  CHAR(5),
             3  F  FIXED  BIN,
          2  G  CHAR(10);
```

などと書ける。但し、データの型宣言を書けるのはツリーの末端（枝の一番先っぽ）である。

（１７－２）構造体の修飾付き名

副構造体の中には同じ名前の構造体があってもよい。例えば、

```
DCL  1  P,
        2  Q,
           3  A,
           3  B,
        2  R,
           3  A;
```

では、Aが２つある。この２つのAを区別するためには、単にAと呼んだのではどちらのAか分からないので、Qに含まれるAという意味で、Q．A，又、Rに含まれるAという意味で、R．Aと書く。このように、それを含む構造体の名前をピリオドをはさんで前に書き加えればよい。それを含む構造体が又同じ名前のときは、更にその上の構造体名をピリオドをはさんで前に書き加える。区別できるまでいくらでも上の構造体名を書き加えることができる。例えば、

```
DCL  1  JAPAN,
        2  TOKYO,
           3  MINATOKU,
              4  YAMADA,
                 5  TARO,
                 5  HANAKO,
        2  OSAKA,
           3  MINATOKU,
              4  YAMADA,
                 5  TARO,
                 5  JIRO;
```

において、２人のTAROは、

TOKYO．MINATOKU．YAMADA．TARO

OSAKA．MINATOKU．YAMADA．TARO

として区別できる。枝の途中の構造体でも同様にして区別する。２つのYAMADAは、

TOKYO．MINATOKU．YAMADA

OSAKA．MINATOKU．YAMADA

として区別し、２つのMINATOKUは、

```
TOKYO．MINATOKU
OSAKA．MINATOKU
```

として区別する。このようにある構造体名に、その上のレベルの構造体名をピリオドをはさんで前に書き加えた形を、修飾付き名という。修飾付き名にしなくても区別できるときは無論、修飾付き名にする必要はない。

（１７－３）構造体配列

構造体の１つ１つが配列の形をとることもありうる。例えば、

```
DCL  1  A,
       2  B,
          3  C(5)  FLOAT  DEC,
       2  D(3)  FIXED  BIN;
```

という構造体は、図１７－２のようなツリーになる。

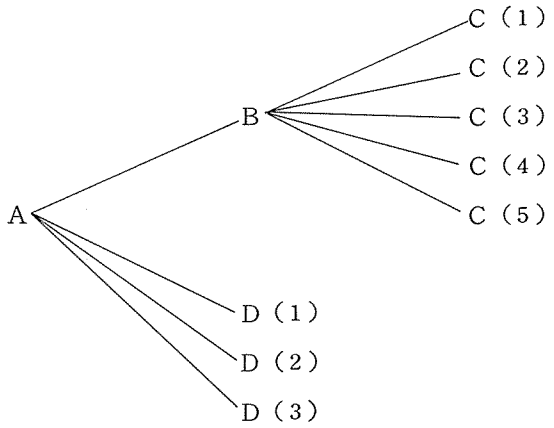

図１７－２　構造体配列の例１。

上例は末端の構造体だけが配列になっている場合であるが、枝の途中の構造体が配列になっている場合は少しややこしいので注意が必要である。例えば、

```
DCL  1  A,
       2  B(2),
          3  C(3)  CHAR(4),
          3  D(2)  FIXED BIN,
       2  E(3),
          3  F  FIXED BIN,
          3  G  CHAR(5);
```

という構造体は図１７－３のようなツリーになる。

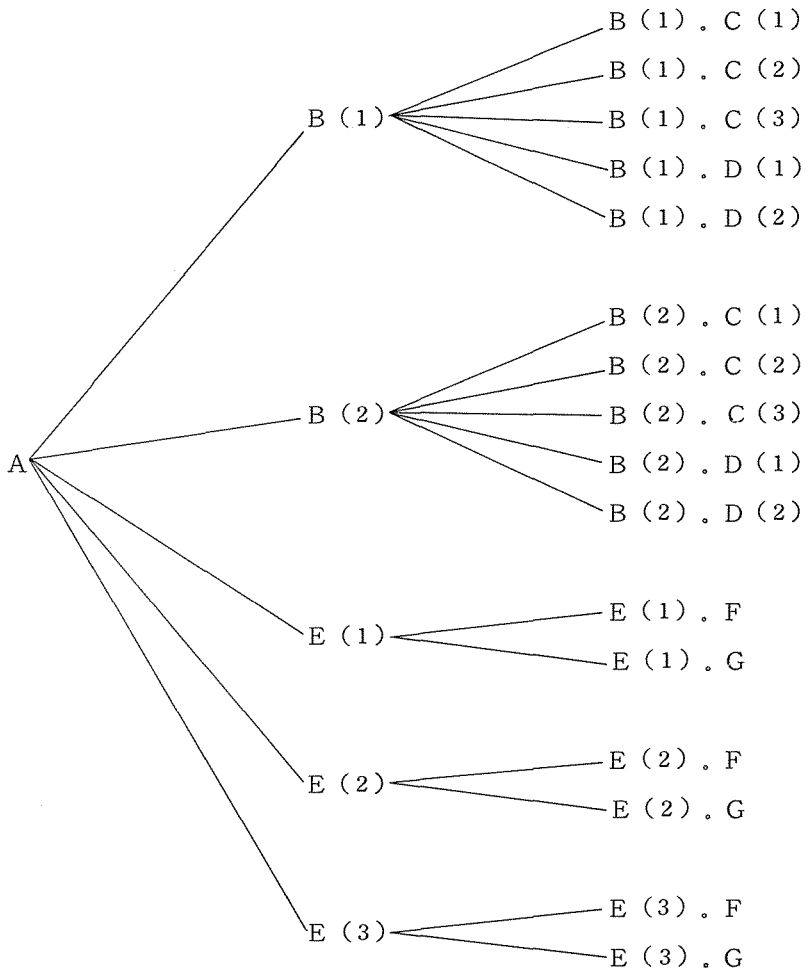

図１７－３　構造体配列の例２.

（１７－４）ＬＩＫＥ属性とＢＹ　ＮＡＭＥ演算

主構造体のみが異なり、副構造体は全て同じである２つの構造体を宣言するとき、例えば、

```
DCL  1  P,
         2  A,
            3  B,
            3  C,
         2  D,
       1  Q,
         2  A,
            3  B,
            3  C,
         2  D;
```

という文は、ＬＩＫＥ属性を用いて簡単に表わせる。すなわち、

```
DCL  1  P,
         2  A,
            3  B,
            3  C,
         2  D,
       1  Q  LIKE  P;
```

とすればよい。

構造体どうしの演算は、同じツリー構造をもつ構造体であれば可能である。例えば、

```
DCL  1  P,
         2  A,
            3  B,
         2  C,
       1  Q,
         2  D,
            3  E,
         2  F,
       1  R,
         2  S,
            3  T,
         2  U;
```

というような構造体ではツリー構造が同じなので、

```
P＝Q＋R；
```

などの算術式が書ける。勿論、たし算に限らず普通の四則演算などが可能である。異なるツリー構造の構造体どうしでも、同じ名前の副構造体どうしを演算する方法があり、ＢＹ　ＮＡＭＥというキーワードを用いる方法である。例えば、

```
DCL  1  P,
        2  A,
           3  B,
           3  C,
        2  X,
           3  Y,
     1  Q,
        2  D,
           3  Y,
           3  C；
```

において、

```
P＝Q, BY  NAME；
```

と書けば、

```
P．X．Y＝Q．D．Y；
P．A．C＝Q．D．C；
```

と書いたのと同じことになる。

今回はＯＮ文と構造体について解説した。構造体はＦＯＲＴＲＡＮには全くない概念なので、本稿の題名のようにＦＯＲＴＲＡＮとの比較ができず、少し分かりにくい説明になってしまったかも知れない。次回は、ＪＣＬのまとめ等について述べる予定である。　　（プログラム相談員）

／＊前回（第４回）の誤植訂正＊／

６１ページと６２ページの内容全体が入れ換わっている。