解　　説

（第5回）

フォートラン・プログラミングにおける バッグ（誤り）とデバッグ（修正）

{算術IF文 / 論理IF文}

研究開発部　磯　本　征　雄

1．はじめに

フォートラン・プログラムは，一般に上から下に向って流れ図として表わされる。しかし，プログラムの実行順序は，始め（上）から終り（下）に向って無条件にすすめられるとは限らない。初期状態の与えかた，あるいは実行の中間結果により，いくつかの異なる手続きの内の1つを選ぶ必要に迫られることが多い。このような実行手続きの条件付き選択のために使われるのがIF文である。

IF文は，表現形式の点では単純である。しかし，IF文によってプログラムのアルゴリズム全体を制御することもでき，その機能の影響する範囲は大きい。また，IF文によっておこなわれる実行手続きの選択方法は，それまでに実行された中間結果に合せておこなわれる。このように，IF文はアルゴリズム全体に関わる要素が多いことと，その振舞いが状況により異なるために，デバッグの際には非常な困難さを引き起こすことがある。今回は，このような含蓄多いIF文について解説する。

IF文に関するバッグは，他の文との関わりが強いために，その追跡，発見が困難である。個々の文は正しくても，それら文相互の関わり方に誤りがある場合には，やはりバッグとなる。IF文のバッグには，特にこの種の文相互の関係に関するものが多い。したがって，プログラムの誤動作が確認され，しかもバッグの所在が不明である場合には，一度はIF文の誤りを疑ってみる必要がある。その際のバッグ追跡の手順，ないしはバッグの起す症状の伝播の因果関係について，ここでは調べて見た。デバッグの際に，なすべき手段もなく暗中模索状態に陥った時には，ここで示す事項を流れ図に従って確認すれば，何らかの手掛りが得られるであろう。

ここでは，JIS-FORTRAN（水準7000）[1]に従って文法規則を解説し，(NEAC) FORTRAN-700 [2]によるプログラムをもとにエラー・メッセージ及びバッグについて解説する。

2．IFに関する文法規則

IF文は，制御文である。但し，その制御の及ぶ範囲は，そのIF文を含むプログラム単位内である。IF文には，算術IF文と論理IF文がある。これらは形の上では類似している。一方，機能

の面から見ると，部分的には類似している所もあるが，基本的には別ものである。算術IF文は，算術形GO TO文[3]に似ている。つまり，分岐点における実行の流れの制御をおこなうのが算術IF文の機能である。これに対して，論理IF文は論理式の“真”，“偽”の判定をおこない，真に対してはそれにつづく右の実行文を実行し，偽ならばそれを無視する。

以下では，算術IF文と論理IF文について，個々にそれらの文法規則を説明する。

2.1. 算術IF文の文法規則

算術IF文は，3分岐をもつ流れの制御をおこなう。すなわち，算術IFのカッコ内の算術式の値が，正，零又は負のいずれであるかにより，算術IF文実行後に実行される文を指定することができる。算術IF文の表現上の文法規則は次の通りである。

算術IF文 算術IF文（arithmetic IF statement）は，つぎの形とする。

IF (e) k_1, k_2, k_3

例：算術IF文

```
IF (A (J, K)－3.0)4, 8, 10
IF (ALPHA+12.3)6, 14, 5
```

ここで，eは整数型か，実数型か，倍精度実数型の算術式とし，k_1, k_2, k_3はいずれも文の番号とする。算術IF文は3方向への分岐であり，この文の実行の際に式cが評価され，その値がゼロより小さいか，ゼロか，ゼロより大きいかによって，それぞれ番号k_1か，k_2か，k_3を持つ文が，つぎに実行されるものとする。

ここで，“ゼロ”と言われているのは，正確にゼロになっていることを意味する。すなわち，算術式実行の結果，近似的にゼロであるが，ほんのわずかながら誤差が残る場合であっても，それはゼロとは見なされない。特に，算術式が実数型変数より成っている場合には，このことは重要な意味をもつ。

2.2. 論理IF文の文法規則

論理IF文は，機能の点から見ると，算術IF文と全く違っている。表現形式上，一部分類似している点もあるが，むしろ両者の差異に注目する必要がある。論理IF文は，形式の上で文法規則として次のように定められている。

論理IF文 論理IF文（logical IF statement）は，つぎの形とする。

IF (e) σ

ここで，eは論理式とし，σはDO文と論理IF文以外の実行文とする。

この文の実行の際に論理式eが評価され，eの値が真ならば文σが実行され，eの値が偽ならば文σはCONTINUE文とみなされて実行されるものとする。

例：論理IF文

文 例：

```
IF (.NOT. L (I))GO TO 3
IF (ALPHA. LT. BETA+3.7) I=I+1
```

誤 例：

```
IF (.NOT .L (I)) DO 4 I=2, 10
```

文の効果：

```
IF (ALPHA .LT .BETA+3.7) I=I+1
K=K+2
```

において，ALPHA＜BETA＋3.7が成立するときは，I＝I＋1が実行され，つぎにK＝K＋2が実行される。

ALPHA≧BETA＋3.7が成立するときは，

```
      ⋮
IF (ALPHA .LT .BETA+3.7) CONTINUE
K=K+2
      ⋮
```

と同等の効果を持つ。

文法規則としてσは，DO文と論理IF文以外の実行文であれば何であっても良い。すなわち，論理IF文は一般にDO文以外のいかなる実行文とも組合せて用いられる。したがって，プログラム作成時の便利さとは逆に，デバッギングにおける困難さの1因ともなり得る。

3．コンパイル時のエラー・メッセージ

コンパイルは，主プログラム又は副プログラム単位ごとにおこなわれる。IF文は,COMMON文等と違って，その制御の及ぶ範囲が主プログラム又は副プログラムの範囲内に限られる。したがって，IF文に関連する誤りの形式的な部分は，コンパイル時に発見される。ここでは，(NEAC) FORTRAN-700におけるIF文に関する，コンパイル時のエラー・メッセージ及びその例を示す。

3.1. WARNING

文法違反ではないが，プログラム上論理的に誤っている可能性のある文については“WARNING”が出力される。特に，以下で示すWARNINGについては，プログラム実行の流れの異常が発生された時に，しばしば確認されるものである。このことは，算術IF文，あるいはGO TO文を共なった論理IF文においてその可能性がある。

メッセージ番号	説明
022	WARNING：UNREFERENCED STATEMENTLABEL “label” ○実行文あるいはFORMAT文に付けられた文の番号が，いずれの文によっても参照されない。 そのままコンパイルされる。

(例)　内部文番号

```
0010      IF (I. GT. 0) GO TO 10
                 ⋮
0100      20  X=Y
                 ⋮
```

内部文番号0100における文番号20は全く引用されない。他の文において引用すべき文番号を間違えた可能性がある。例えば上の例では，論理IF文において，GO TO 10が実はGO TO 20であったためにこのWAPNINGが出力された。

3.2. FATAL ERROR

明らかに文法違反であるためにフォートラン文としては許されないものについては“FATAL ERROR”が出力される。この時，FATAL ERRORとなった文は無視され（すなわち無かったものとして）コンパイルはつづけられる。FATAL ERRORのあったプログラム単位を結合処理して実行するか否かは，システムに対するOPTIONの指定により可能なものもある。

以下，エラー・メッセージ及びそれらに対する具体例を示します。

メッセージ番号	説明
217	ILLEGAL REFERENCE TO FORMAT ○ASSIGN文，GO TO文あるいは算術IF文でFORMAT文の文番号が参照されている。この実行文は削除されてコンパイルされるが，プログラムの実行時にこの文に実行の制御が渡るとUEPになる。

（例）　内部文番号

```
                    ⋮
0010          IF (I—J)     10, 20, 30
0011    10    X=Y
                    ⋮
0020    20    FORMAT (1H  , @EXAMPLE @)
                    ⋮
0030    30    X=Z
                    ⋮
```

算術IF文の後の実行制御の行き先きは，実行文でなければならない。この例では，内部文番号0020のFORMAT文へ移っているために誤りである。

221	ILLEGAL STATEMENT TYPE FOLLOWS LOGICAL IF ○論理IF文に許されない文が含まれている。この論理IF文に含まれている文は削除されてコンパイルされる。プログラムの実行時にこの論理IF文の論理式の値が真のときUEPになる。 備考：　論理IF文に含まれてもよい文は，DO文と論理IF文を除く実行文である。

（例）　内部文番号

```
                    ⋮
0010          IF ( .NOT .L (I))   DO  120  I=1, 15
                    ⋮
```

論理IF文にDO文を含めてはいけない。

222	NON-LOGICAL CONDITION FOR LOGICAL IF ○論理IF文において，論理型以外の型の式が引用されている。この文は削除されてコンパイルされる。プログラムの実行時にこの文に実行の制御が渡るとUEPになる。

(例)　内部文番号

0010　　REAL　A, B

0020　　IF (A＊B) GO　TO　55

IF文の右側に文番号以外の実行文が書かれている場合には，これは論理IF文とみなされる。上の例では，算術式がIF文中に有りながら，実行文（GO TO 55）が右側に書かれているために誤りとなる。一見，算術IF文の形をとった，誤った論理IFとみなされている。

226	COMPLEX OR NON-NUMERIC ARITHMETIC IF EXPRESSION ○算術IF文において，整数型，実数型あるいは倍精度実数型以外の型の式を引用しているこの文は削除されてコンパイルされる。プログラム実行時にこの文に実行の制御が渡るとUEPになる。

(例)　COMPLEX　A, B

IF (A+B)　10, 11, 12

この例では，算術IF文中の算術式に複素数型変数が使われている。算術IF文中の算術式は，整数型，実数型あるいは倍精度実数型の内いずれかでなければいけない。

227	ILLEGAL ARITHMETIC OR LOGICAL IF STATEMENT SYNTAX ○算術IF文あるいは論理IF文において文法違反がある。この文は削除してコンパイルされる。プログラムの実行時にこの文に実行の制御が渡るとＵＥＰになる。

(例)　(1)　IF (A+B−C)　10, 20

算術IF文において，その右側にある文番号は，3個書かれなければならない。上の例では，文番号が2個しか書かれていないので誤りである。

(例)　(2)　IF (SIN (X) −B)　10　＊　20, 30

算術IF文において，文番号の書かれるべき所に10 ＊ 20なる算術式があるために，誤りである。“,”となるべき所が“＊”とミスパンチした場合等にこのようになる。

251	UNDEFINED STATENMENT LABEL REFERENCE TO “label” ○入出力文を除く実行文かデバッグ文で参照されている文の番号が実行文で定義されていない。 実行文の場合はその文が削除されてコンパイルされる。プログラムの実行時にその文に実行の制御が渡るとＵＥＰになる。デバッグ文の場合はそのデバッグ文を含むプログラム準位はGOファイル上に出力されない。

（例）　内部文番号

```
                    ⋮
0010          IF (K.EQ.I)  GO  TO  100
                    ⋮
0040    1000  X=Y
                    ⋮
```

この例は，内部文番号0040において

100　X=Y

とすべき所を誤って

1000　X=Y

としてしまった場合である。したがって内部文番号0010における論理IF文中のGO TO 100における文番号100が未定義となって，誤りである。

4．IF文の実行時における振舞い

IF文には，算術IF文と論理IF文がある。これらの実行時における振舞いは異なる。これら両者の共通点は，IF（　　）のカッコ内に変数及び式があることである。論理IF文中には論理式があり，算術IF文中には算術式がある。このようにIF文中には，常に変数が存在する。したがって，IF文に関する実行時の誤動作は，IF文中の式の誤りだけでなく，IF文が実行される以前に定義された変数自体の誤りの可能性もある。

ここでは，以上の点に注意しながら，算術IF文及び論理IF文の各について，実行時の振舞いを調べる。

4.1. 算術IF文の実行時における振舞い

算術IF文において，プログラム実行の異常が発見された場合，注意すべきことは，その原因が算術IF文自体にあるのか，あるいは算術IF文以前の段階にその原因があるのかを見極めることである。

算術IF文中には整数型又は実数型の変数がある。今，ここで次の2つの場合を考えて見よう。

(1)　変数のREAD文での入力に誤りがある。

(2)　変数の値を定義する代入文等に誤りがある。

これらは，いずれの場合にも，変数の値が誤りとなる。もし，これらの変数が算術IFで使われている場合，

(3)　算術IF文中で使われている変数の値が誤りである。

ということになる。

ここで，(3)の誤りが起こっている場合，例え算術IF文中の算術式が正しくても，次の結果を引き起こす。

(4)　算術IF文での算術式の値が誤りである。

もちろん，

(5) 算術IF文中の算術式に誤りがある，

場合には，この結果として，(4)の誤りが生ずる。誤り(4)が確認された段階では，その原因が(5)によるものか，又は(3)によるものかは，具体的にプログラムを調べるしかない。

さて，算術IF文は

IF (e) k_1, k_2, k_3

で書かれるが，算術IF文の実行結果が正しくない理由としては，次の２つが考えられる。第１は，上述の誤り(4)である。第２は，

(6) 算術IF文実行後の，実行の移り先きの文番号k_1 k_2又はk_3等の誤りである。

但し，誤り(6)の場合には，コンパイル時に

(7) 022 WARNING ; UNREFERENCED STATEMENT LABEL “lebel”の出力がなされている場合が有るので，合せて調べておく事が必要である。

算術IF文の誤りは，一般に

(8) 算術IF文実行後の実行の流れが異常である，

現象として発見される。実行の流れの異常は，デバッグ用又は，他の目的でおこなわれた出力の結果知り得る。すなわち，

(9) WRITE文による高速製表印字装置への出力順序が誤りである。

として確認される。しかし，WRITE文が必ずしも適切な場所に有るとは限らない。このために，結局，最終結果において，

(10) 処理結果における出力数値等に異常がある，

か又は，

(11) 処理結果における出力数値の異常と同時にCPU-timeが予測値に比べて非常に違っている。

等の現象を確認するに至って，算術IF文の誤りを知ることになる。(但し，上記(10)及び(11)は，GO TO文等他の文とも同じなので，現実には算術IF文にのみ限定することはできない。)

さて，これまでの議論を確認するために，図４－１に算術IF文の使われたプログラム例を示した。このプログラムは，フォートラン文法上正しいものである。今，このプログラムの処理結果が最初に予定していたものとは違っているという仮定の下にいくつかの考察をしてみよう。但し，誤りは，算術IF文に原因する場合のみに限定する。

図4.1　算術IF文を使ったプログラムの例

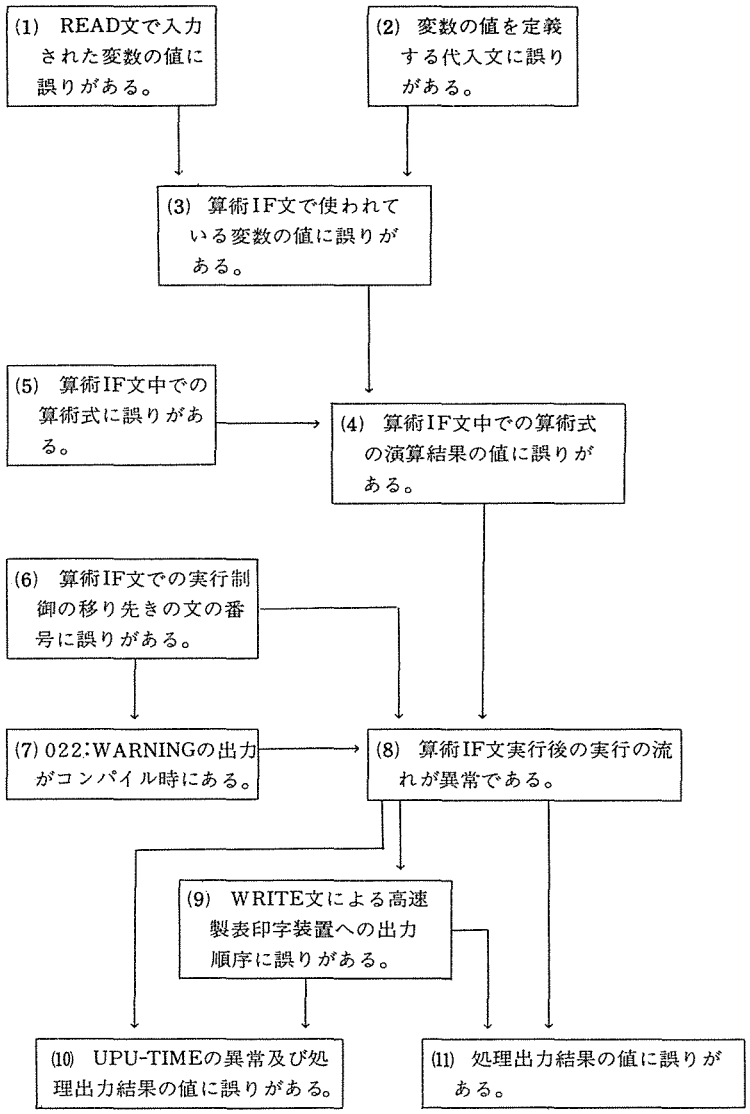
```
      DATA I, J, X, Y, Z/1, 2, 1.0, 2.0, 3.0/
C     ARITHMETIC IF STATEMENT

      IF (I−J)   10, 20, 30
10    A=X+1.0
C        :
      GO TO  40
20    B=Y+2.0
C        :
      GO TO  40
30    C=Z+3.0
C        :
40    CONTINUE
C        :
      STOP
      END
```

算術IF文にはI，Jなる変数があり，これらは，DATA文（内部文番号0001）で定義される。誤りの可能性の第1はこれらの定義された値にある。次にIF文中の算術式（I-J）が誤りであれば，（例えば正しくは（I＋J）等であるならば）結果は，やはり誤りとなる。第3番目に文番号 10，20，30 がこの順序に算術IF文の右に並んでいるが，しばしば，この順序を誤ることがある。このように算術IF文自体を中心に，その前後に誤りの原因と結果が連なっていいることがわかるであろう。実際に，図4.1のプログラムが何如に訂正されるべきかは，プログラムの目的・意味と合せて考えなければならない。

このような算術IF文の誤りの関連を，因果関係の順序に従って整理したものを図4－2に示す。図において，上から下に向って，矢印の順序に原因から結果に向って書かれたものであ。実際にデバッグの際に利用する場合には，最下端の最終結果より上に向って原因をさぐってゆくことになる。

図4.2. 算術IF文に関するバッグの因果関係



4.2. 論理IF文の実行時における振舞い

論理IF文は，その右側に様々の実行文をつづけ得るので，誤りの判定は非常に困難である。誤りが発見された場合でも，その症状が論理IF文の右側の実行文と並発して起こるために，その副作用は非常に大きい。例えば，次の論理IF文を考える。

```
      X＝U
      IF (K .EQ .J)    X＝Y＋Z
        ⋮
      WRITE (6 , 100)     X
```

このプログラムにおいて，WRITE文による出力結果によりXの値が誤りであったことが確認されたとしよう。その原因が論理IF文の所に存在するとしても，論理IF自体の誤りであるのか，あるいは算術式X＝Y＋Zが誤りであるのかは一義的には決まらない。少くとも，論理IF文自体が誤りであれば，Xの値は誤りとなる確率が非常に高い。逆に例え論理IF文が正しく実行されても，その右側につづく実行文が正しく実行されたか否かは，また別の問題なのである。

このような論理IF文における誤りを，原因→結果の順に解きほぐして見ると，次のように説明することができる。まず，

(21) 論理IF文で使われている変数（実数型・整数型・論理型等）の値に誤りがある。

変数の値の誤りは，それ以後の処理にとって致命的である。このために論理IF文中の論理式が正しくても，その結果は誤りである。一方，

(24) 論理IF文中の論理式に誤りがある，

場合にも，誤り(21)の場合と同様に，その結果は誤りとなる。すなわち，

(25) 論理IF文において，論理式の値が誤りである。

という結果に終る。なお，誤り(24)についてさらに詳細に見るならば，

(22) 関係演算子に誤りがある。

場合と

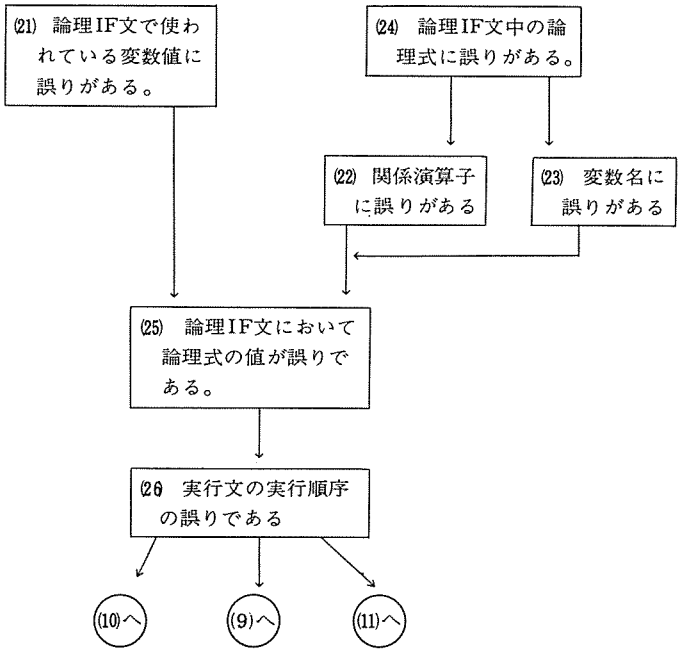
(23) 変数名に誤りがある

場合の２つつの状況が考えられる。これら論理IF文の誤りは，その帰結として，その右側につづく実行文が実行されるか否かにかかわってくる。非常にあいまいな言い方にしかならないが，これらは，

(26) 実行文の実行順序の誤りである。

といういいかたでまとめられる。論理ＩＦ文は，常に実行文を共なう。このために論理ＩＦ文の誤りは，同時に論理ＩＦ文の右側の実行文の実行に影響する。言い換えるならば，論理ＩＦ文の誤りは一見ＤＯ文以外の他のすべての実行文のいずれかの誤りとも共通した症状を並発することである。デバッグの時には，この点を充分に配慮しておく必要がある。図４.３には，論理ＩＦ文に関する誤りの症状について，その影響の因果関係を示した。これにより，バッグの様相を見ることができるだろう。

図4.3. 論理IF文に関するバッグの因果関係



5. まとめ

IF文は，制御文であるために，それ自身の誤りのために生ずる症状は，明確でない。このために，デバッグの手続きも，IF文固有の方法がない。したがって，プログラムに誤動作が発見された場合には，その時のプログラム実行の流れを確認し，正しく実行された部分と，誤動作の生じた後の部分を区別し，この境界線上に誤ったIF文があるか否かを調べるほかない。特に，IF文に関しては，この方法でデバッグの発見されることが多い。

IF文の所で誤動作の起ったことが確認された場合，IF文の誤りだけでなく，そこに使われた式の変数を値が正しいものか否かをWRITE文により確実に確かめておくことが必要である。式が正しいということだけで，あるいはその右の文番号や実行文が正しいというだけでは，IF文が正しく実行されたか否かの決め手にはならない。この点を軽視すると，デバッグに非常に手間取ることがあることを最後に強調しておきます。

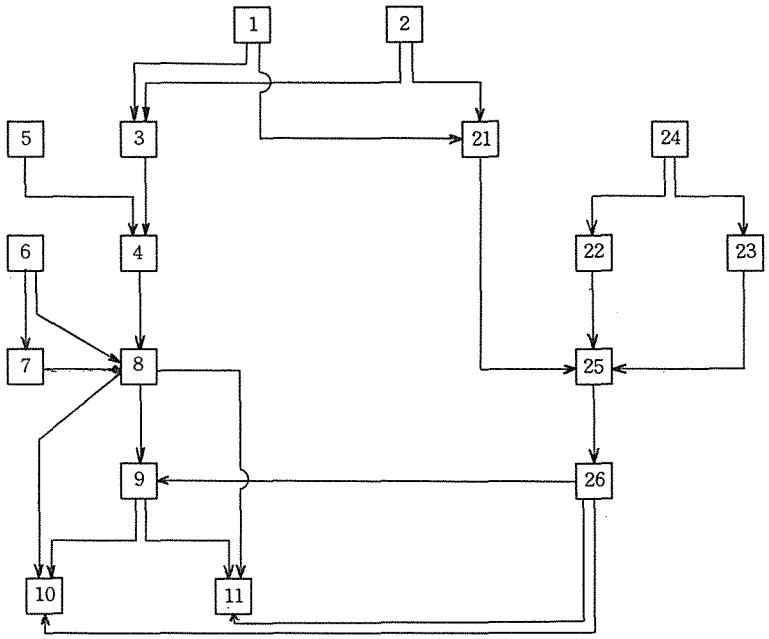
引用文献

1）JISハンドブック，情報処理，1973，日本規格協会。

2）EEAC-シリーズ2200オペレーテングシステム NODIV EX/Ⅶ FORAN700プログラミング説明書。

3）大阪大学大型計算機センター・ニュース No.16 (1975)。

付録　IF文に関する実行時の誤動作の流れ図



(1)　READ文で入力された変数の値に誤りがある。
(2)　変数の値を定義する代入文に誤りがある。
(3)　算術IF文で使われている変数の値に誤りがある。
(4)　算術IF文中での算術式の演算結果の値に誤りがある。
(5)　算術IF文中での算術式に誤りがある。
(6)　算術IF文での実行制御の移り先きの文の番号に誤りがある。
(7)　022：WARNINGの出力がコンパイル時にある。
(8)　算術IF文実行後の実行の流れが異常である。
(9)　WRITE文による高速製表印字装置への出力順序に誤りがある。
(10)　CPU-Timeの異常及び処理出力結果の値の誤りである。
(11)　処理出力結果の値に誤りがある。
(21)　論理IF文で使われている変数値に誤りがある。
(22)　関係演算子に誤りがある。
(23)　変数名に誤りがある。
(24)　論理IF文中の論理式に誤りがある。
(26)　論理IF文において論理式の値が誤りである。
(26)　実行文の実行順序の誤りである。