（最終回）

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）

DATA文
型宣言文
文関数定義文

研究開発部　磯　本　征　雄

1. はじめに

フォートラン・プログラムでは，バッグを追跡する場合，たいてい実行文からはじめます。そして，バッグ追跡の過程で必要に応じて宣言文その他非実行文を調べます。このような理由から，非実行文のデバッグはとかく最後になりやすいものです。副題の“型宣言文„，“DATA„，“文関数定義文„を最終回にした理由もこの点にあります。しかし，これらのものを後まわしにしたほうが良いと言う意味ではなく，本文に見るように，これらのデバッグは比較的容易なことが多いので，臨機応変に時には早い段階で行うことが良いでしょう。

ここで解説する3つのフォートラン文は，相互に強い関係をもつものではないが，単に都合上ひとまとめにしたまでです。これらの共通の特徴は，他の非実行文と同様に，実行文の前に置くのが無難であるという点です。厳密な文法の問題としてではなく，デバッグの際の見透しをよくする点でお奨めします。文法に関する次節の説明に違反しない限り記述の場所を自由に選ぶことも可能ですが，それはあくまでも文の特徴を十分に理解した上で行って下さい。

以下，文法はＪＩＳ－ＦＯＲＴＲＡＮ（水準7000）によるものとして，解説します。また，本文中の具体例におけるテスト・プログラムはＮＥＡＣ　ＡＣＯＳ－６ＦＯＲＴＲＡＮにより実行したものです。

2. 文法規則

2.1 DATA文

ＤＡＴＡ文は，ＪＩＳ－ＦＯＲＴＲＡＮにおいて次のように規定されています。

DATA文　DATA文(data initialization statement)は，つぎの形とする。

DATA $k_1/d_1, k_2/d_2, \cdots k_n/d_n$

ここで，$k_1, k_2, \cdots k_n$ は，いずれも変数名や配列名や配列要素名の並びで，仮引数を含まないものとし，添字式は整定数でなければならない。$d_1, d_2, \cdots d_n$ は，いずれも一般の定数の並びとし．それぞれの前にはJ＊をつけてもよい。Jは整定数であり，J＊はその直後の一般の定数をJ回くり返すことを意味するものとする。

二つ以上の項目が並びに含まれている場合には，それらの項目はコンマで区切る。

ＤＡＴＡ文は，変数や配列や配列要素の初期値を定義するのに使用する。変数名や配列要素名の項目と一般の定数の項目との間には1対1の対応がなければならない。このとき，配

例：DATA文

文　例　1：

```
DATA A,I,C/2 9.3,5,5.0/
```

文　例　2：

```
DATA A,I,C/2 9.3,5,5.0/,X,
   Y/4 0.1,2.7/
```

は

```
DATA A,I,C,X,Y/2 9.3,5,5.0
   4 0.1,2.7/
```

と同じ。

文　例　3：

```
DATA A,B,C/3*1.0/
```

は

```
DATA A,B,C/1.0,1.0,1.0/
```

に同じ。

列名は，そのすべての配列要素名が添字の値の順序に並んでいることを意味する。この対応によって初期値が定まるものとする。

使 用 例

```
DIMENSION A(2,2)
DATA A/1.0,2.0,3.0,4.0/
```

は，

```
DIMENSION A(2,2)
DATA A(1,1),A(2,1),A(1,2),A(2,2)
/1.0,2.0,3.0,4.0/
```

に同じ。

初期値が与えられる変数や配列要素は無名共通ブロックに含まれてはならず，名前付共通ブロックに含まれうるのも初期値設定副プログラムの中だけとする。

ＤＡＴＡ文は初期値設定を行うけれども，それは代入文によるものとは本質的に異るものです。ＤＡＴＡ文は非実行文です。そして変数の初期値設定は，コンパイルの段階で１度行われるのみで，実行段階でもし再定義されたならば，ＤＡＴＡ文の効果は以後関係のないものとなります。プログラム・デバッグの際には上記の文法規則のほかにこのような実行時の効果についても十分意識した上でＤＡＴＡ文を使って下さい。

2.2 型 宣 言 文

フォートランでは，各々の変数に対して固有の取扱い規則があります。もうすこし計算機に密着した書き方をすれば，変数の型ごとに計算機内部での言わゆる“内部表現„が異ります。たとえば，数学的には整数は実数の一部分であり，値が同じであれば同じものと考えることができます。ところが，フォートランでは，それらの内部表現が異るために計算機内部の取り扱われ方が異ります。したがって，型の異る変数間での混合演算や代入文の取り扱いでは，この点を十分に注意する必要があります。

ＪＩＳ－ＦＯＲＴＲＡＮ（水準７０００）では，変数に対する型宣言の方法を次のように定めています。

型宣言文　型宣言文（type statement）は，つぎの形とする。

t $v_1, \cdots, v_n$

ここで，tはＩＮＴＥＧＥＲ　ＲＥＡＬ，ＤＯＵＢＬＥ　ＰＲＥＣＳＩＯＮ，ＣＯＮＰＬＥＸまたはＬＯＧＩＣＡＬで，それぞれ整数型，実数型，倍精度実数型または論理型を意味し，$v_1, v_2, \cdots v_n$は，いずれも変数名，配列名，関数名または配列宣言子とする。

例　型宣言文

```
INTEGER BIXE,X,QF,LSL
REAL IMIN,LOG,GRN,KLW
DOUBLE PRECISION Q,J
INTEGER A(10,10),B
COMPLEX C,D(4,5,3)
```

上記の最後の二つの例は，つぎの三つの宣言文の集まりと同等である。

```
DIMENSION A(10.,10),D(4,5,3)
INTEGER A,B
COMPLEX C,D
```

型宣言文は，それに含まれる英字名に関連づけられる型を定めるものとする。この関連づけは暗黙の型宣言に優先する。また型宣言文は配列の寸法を規定してもよい。

型宣言は，暗黙の型宣言以外の使われ方をする場合にのみ有効です。暗黙の型宣言とは，Ｉ，Ｊ，Ｋ，Ｌ，Ｍ，Ｎを先頭第１文字目にもつ変数を整数型変数とみなすということです。計算機における内部表現については，プログラミング説明書に詳細にしるされているのでここでは省略いたします。

2.3 文　関　数

文関数は，外部関数と異なり一つの副プログラム内部のみで有効なフォートラン文の一つです。文関数には，それ自体のもつ意味又は文関数定義文としての意味と同時に，文関数を引用する仕方や引用された時の働きが問題となります。この点では，外部関数と似ているともいえます。このようなことから，文関数はＪＩＳ－ＦＯＲＴＲＡＮ（水準７０００）においても以下のように現定されています。

文関数　文関数（statement function）は，それを引用するプログラム単位内で，算術代入文または論理代入文と同じような形の文関数定義文で定義する。

一つのプログラム単位内では，すべての文関数定義文はそのプログラム単位の実行文よりも前で，かつ，宣言文があるならば，それよりもあとに置かなければならない。文関数の名前はEXTERNAL文中に現われてはならないしまた同じプログラム単位中で変数名や配列名として現われてはならない。

文関数定義文　文関数定義文（statement function defining statement）は，文関数を定義するもので，つぎの形とする。

$f(a_1, a_2, a_n) = e$

ここで，ｆは関数名，ｅは式とし，ｆとｅとの関係は7.1.1の(1)と(2)の割当ての規則（脚注参照）を満足しなければならないものとする。$a_1, \cdots a_n$ は互いに異なった変数名とし，文関数の仮引数（dummy argument of statement function）という。これらは仮の引数で，引数の型，個数および順序を示すことだけに役立つものとし，そのプログラム単位内の他の場所で現われる同じ型の変数名と同じであってもよい。仮引数以外に，式ｅはつぎに示すもののみを含みうるものとする。

文字型でない定数
定数の引用
組込み関数の引用
すでに定義されている文関数の引用
外部関数の引用

文関数の引用　文関数は，算術式または論理式で１次子として関数の引用を用いることによって引用する。引数の並びに書かれている実引数は，その対応する仮引数と，順序，個数および型が一致していなければならない。文関数の引用の中の実引数は，対応する仮引数と型が同じならば式であってもよい。

文関数の引用が実行されると，実引数の値とこの関数を定義している式中の対応する仮引数との結合がおこり，その式の評価が行なわれ，割当ての規則に従って得られた値が，この関数の引用を含む式で使用できるようになるものとする。

例：文関数定義文

```
ENORM(X,Y)=SQRT(X**2+Y**2)
```

3. プログラム実行時に発見される誤り

コンパイル時には，多くの場合コンパイラーからのメッセージにより誤りの発見が可能です。したがって，ここではプログラマー自身でバッグを追跡しなければならないような誤りについて，実行時に焦点をしぼって解説します。

3.1 ＤＡＴＡ文の実行時の誤り

ＤＡＴＡ文には，変数名，定数の値そして変数名と定数の間の一対一の対応関係の３つの

（脚注）

7.1.1 代入文　代入文（assignment statement）

（１）**算術代入文**　算術代入文（arithmetic assignment statement）は，つぎの形とする。

$v = e$

ここで，ｖは論理型以外の変数名または配列要素名とし，ｅは算術式とする。

この文の実行によって式ｅが評価され，

例：算術代入文

```
DELTA=ALPHA(A**2)**3
A(1,3)=B(J,I)
Y=-(A**2)**3+12.31)-5/B(2*I+1.5)-
  (2.8*A+G*F(K+3))*SIN(X)
I=I+1
KAJI=109
XI=XI+DELTAX
KAPPA=(-14.9,1.64E-4)(6.55,7.89)
```

（２）**論理代入文**　論理代入文（logical assignment statement）は，つぎの形とする。

ここで，ｖは論理型の変数名または配列要素名とし，ｅは論理式とする。

例：論理代入文

```
F=A.AND.B.OR.C
G(5)=.NOT.L(1)
H=(F.OR.GEORGE).AND.G(I)
A=NOT.A
```

この文の実行によって式ｅが評価され．その値がｖに割当てられるものとする。

因子があります。したがって，ＤＡＴＡ文における誤りとしては概略次の３つの場合があります。

(1)　ＤＡＴＡ文中の変数名の誤り。

(2)　ＤＡＴＡ文中の定数の値の誤り。

(3)　ＤＡＴＡ文中の変数名と定数の対応関係の誤り。

これらは見掛け上

(4)　ＤＡＴＡ文で定義された変数の値に誤りがある。

ことにより，一括して把握されます。

これら(1)，(2)，(3)の内のいずれかであるかは，プログラマ自身のプログラミング上の意図に合せて調べなければなりません。

なぜならば，正しくは

ＤＡＴＡ　Ｘ，Ｙ，Ｚ／1.2，2.3，3.4／

であるものを間違えて

ＤＡＴＡ　Ｘ，Ｙ，Ｚ／2.3，1.2，3.4／

としてしまっても，これだけでは正誤の判断がつかないことからも明白です。

次に時々見掛けるＤＡＴＡ文の誤りとして

(5)　ＤＡＴＡ文の機能に関する誤解

があげられます。例えば図3.1に見るように，ＤＡＴＡ文を代入文のかわりに用いることはできません。図3.1ではプログラム前半で用いられていた変数ＡとＢを文番号１０以下の所でＤＡＴＡ文を使ってＡとＢを再定義しています。しかし非実行文であるＤＡＴＡ文は結局無いに等しいものとなります。

このような

(6)　ＤＡＴＡ文を実行時に変数の再定義に用いようとした。

誤りなどはうっかり落ち入りやすい落し穴の一例とでもいえるでしょう。さて，前述の(4)及びここでの(6)のいずれも，プログラム中では

(7)　変数の値に誤りがある。

ことによりバッグの存在を知らされるでしょう。

この変数がＩＦ文中で使われた場合などは，

(8)　算術ＩＦ文，論理ＩＦ文等の誤動作による実行順序の異常がある。

ことを発見することもあり得ます。いずれにせよ，これらは

(9)　演算結果に誤りがあることを知ることにより

わかることです。

```
１０　Ｂ＝1.0
　　Ａ＝4.0
　　Ｗ＝1.0
　　DATA　A, B／1.0, 2.0／
　　Ｘ＝Ａ＊Ｂ＊Ｗ
```

図3.1 ＤＡＴＡ文の誤用例。ＤＡＴＡ文は非実行文なので変数の再定義には使えない。

ＤＡＴＡ文の誤りのために起こる諸々の症状は，入力文による初期設定や代入文による変数値の定義の際の誤りによって生ずる症状とほとんど同じです。したがって，

ＤＡＴＡ文のみについて見れば，上記の通りですが，一般にはＲＥＡＤ文の誤りとも合せて調べる必要があります。図3.2にＤＡＴＡ文にのみ注目したバックの症状についてまとめました。

ＤＡＴＡ文

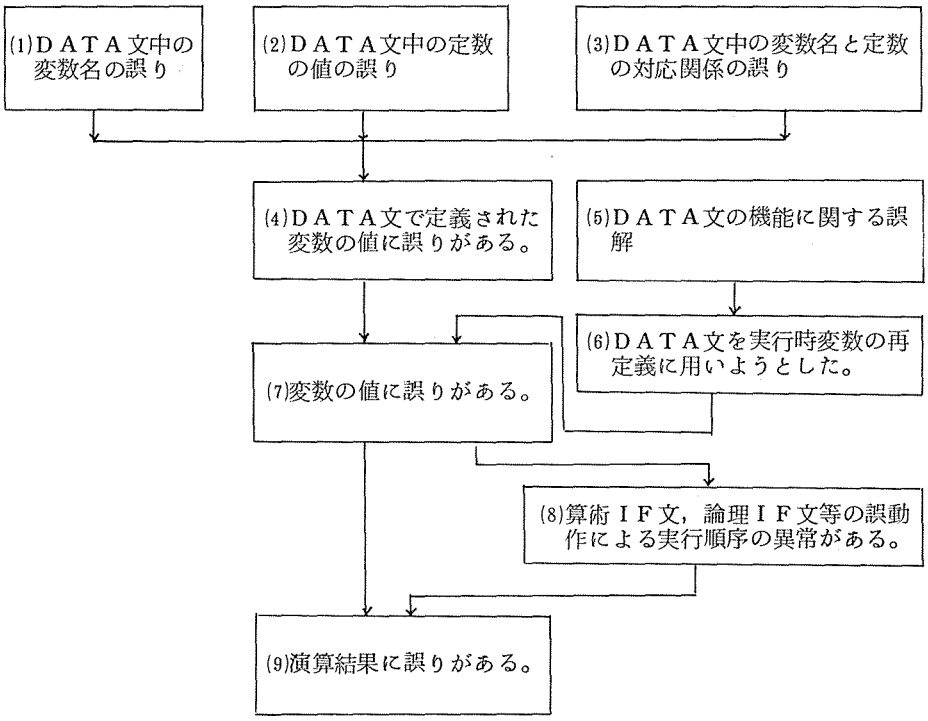

図3.2　ＤＡＴＡ文に関するバッグとその症状。
矢印は上から下へ各々因果関係を示す。

3.2　型　宣　言　文

フォートランにおいて変数の型が重要になるのは，主に論理式と算術式の区別及び変数の精度に関する配慮の2点があります。ここでは

(10)　変数名，配列名，関数名に対する型宣言の方法に誤りがある。

場合の具体的事項を1つずつ見てゆくことにします。

(11)　型宣言ＲＥＡＬが無い場合

の状況としては，

(12)　実数型変数が整数型にみなされる

ことになる。

例えば，Ｉ，Ｊ，Ｋを実数型変数であるとして，次の式を考えます。

Ｗ＝（Ｉ／Ｊ）＊Ｋ

ただしＷは実数型であるとします。この前に，

Ｉ＝１０.０

Ｊ＝２０.０

Ｋ＝１０.０

なる代入文があったとして，これを上の式に代入してみて下さい。フォートランでは

（Ｉ／Ｊ）＊Ｋ＝（１０.０／２０.０）＊１０.０＝5.０００

となるはずです。（ただし内部表現にかかわる小さな誤差はこの式に限り，無視して解説します。）ところが，ここで

ＲＥＡＬ　　Ｉ，Ｊ，Ｋ

が無かったならば上の式は

（Ｉ／Ｊ）＊Ｋ＝（１０／２０）　＊１０＝０

となります。それは，（１０／２０）＝０が先に処理され，その後にＫ＝１０が掛けられるからです。このように，型宣言の誤りは大きな数値上の誤りをおこさせます。

同様に，

⑬　型宣言ＤＯＵＢＬＥが無い。

場合においては

⑭　実数型変数において倍精度の精度が無い。

ことになってしまいます。また

⑮　型宣言ＣＯＭＰＬＥＸが無い。

場合には，変数において虚数部分が無視されるために

⑯　虚数部に関連する数値に誤りがある。

ことになります。算術式実行の結果は，虚数部が実数部に影響することが常にあるので，ＣＯＭＰＬＥＸ宣言の欠落は決して虚数部にとどまるものではありません。

上記⑪の実例の所にも示したように，フォートランにおける整数型と実数型の変数では形式だけでなく演算処理上も大変異ります。したがって，

⑰　型宣言１ＮＴＥＧＥＲが無い

場合には当然のことながら

⑱　整数型の特徴が生きていない

ことがあります。ここで，その具体例を見ることにしましょう。今，変数Ｘを整数型であるとして，Ｘが偶数ならばＷに0.0を代入し，Ｘが奇数ならばＷに－1.0を代入することを考えます。これは次の式で簡単に行えます。

W＝（Ｘ／２）×２－Ｘ

ただしＷは実数型変数であるとします。当然Ｘに対しては

ＩＮＴＥＧＥＲ　Ｘ

がなければなりません。もし型宣言ＩＮＴＥＧＥＲが無かった場合，代入文Ｘ＝９でＸの値を定義すれば上式の値は

（Ｘ／２）＊２－Ｘ＝（９.０／２）＊２－９.０

＝９.０－９.０

＝０.０　　⟶　　Ｗ＝０.０

となります。したがって整数型宣言のある場合の

（Ｘ／２）＊２－Ｘ＝（９／２）＊２－９＝－１→Ｗ＝－１.０

とは結果が異ります。このように整数型変数の特徴を利用した算法では，型宣言を落してしまうか否かにより結果に大きな影響を与えます。

これら⑿，⒁，⒃，⒅は大雑把にいって，㉑計算結果に誤りがあるという一言でまとめられます。また逆に，計算結果の誤りが他の原因によることもありますが，型宣言の誤りも決して見落してはならないということです。

次に，

⑲　型宣言ＬＯＧＩＣＡＬが無い

場合を考えてみましょう。この場合，たいていはコンパイル時のエラー・チェックに掛かると思われますが，例えば

ＩＦ（Ｘ．ＥＱ．　Ｙ）

等のように，ＩＦ文の中で論理式が現われた場合にはＸ，Ｙが論理型であるか否かが問われないままに処理されます。この時には当然

⑳　論理演算結果に誤りがある

ことになります。したがって，

㉑　論理ＩＦ文で誤動作が生じます。

これは結局のところ

㉒　実行手順に異常が生じます。

以上，型宣言に関する症状について主要と思われる部分のみについて解説しました。これらをまとめて，因果関係の順に矢印でつないだものが図３.３の結果です。

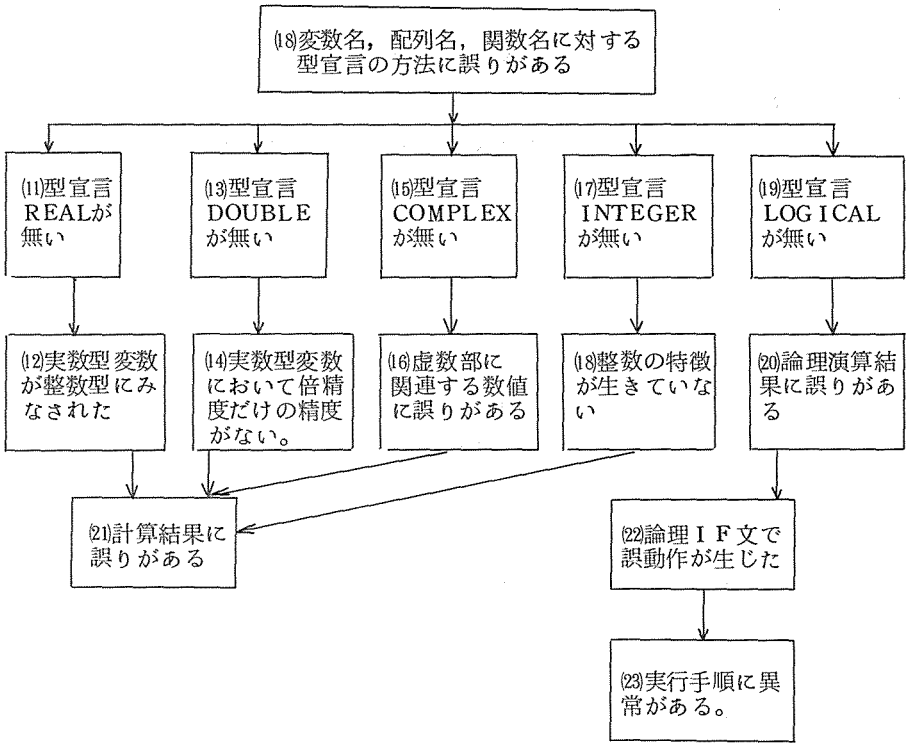

図3.3　型宣言文に関するバッグとその症状。
矢印は上から下に向って因果関係を示す。

3.3　文　関　数

文関数が外部関数に類似したものであることは，文法の所でみたとおりです。また，表現形式上，左辺に配列要素をもつ算術代入文にも似ています。これらのことは，当然プログラム実行時の誤動作の症状に反映してきます。

まず文関数の定義及び文関数引用の例を図3.4に示しましょう。文関数の定義において誤りやすいのは，

(24)　文関数定義文において，右辺の算術式の誤り

が起る場合です。

通常の算術式でもしばしばおこることであり，文関数の場合も例外ではありません。

(25)　文関数定義文の左辺仮引数の誤り

があります。左辺の仮引数は，変数の型，順序，

```
      ⋮
F(X,Y)=2.0*X+Y*Y
      ⋮
R=F(A,B)
      ⋮
```

図3.4　文関数定義と文関数引用の具体例

及び個数が正しく右辺と対応していなければなりません。たとえば図3.4において，定義文をF（Y，X）＝2.0※X＋Y※Yとしてしまったならば，文関数の定義自体が全く異るものとなってしまいます。このことは，

F（X，Y）＝2.0＊X＋Y＊Y

F（Y，X）＝2.0＊Y＋X＊X

なる2つの定義文が全く同じものであることを強調すれば十分理解できるでしょう。

文関数は定義されると同時に，同じプログラム単位内で引用もされます。したがって正しく定義されると同時に正しく引用されることが必要です。引用の誤りとしてまず，

(27) 文関数引用代入文における，実引数の変数名の誤り

があります。たとえば図3.4において対応関係

実引数　　仮引数

A ⟷ X

B ⟷ Y

が正しくなければなりません。ところが，実引数の変数名が正しい場合であっても，それ以前の実引数の変数の値を定義する段階で誤っていたならばやはり

(26) 文関数引用の代入文において実引数の値の誤り

が生じます。

このように(24)，(25)，(26)，(27)，は総て一括して

(28) 文関数引用の代入文で，値が正しく得られない

という結果にいたります。さらに一般的な言い方をすれば，

(29) 算術式演算結果の誤り

ということになります。これらを因果関係の順に矢印でむすび，まとめたものが図3.5です。

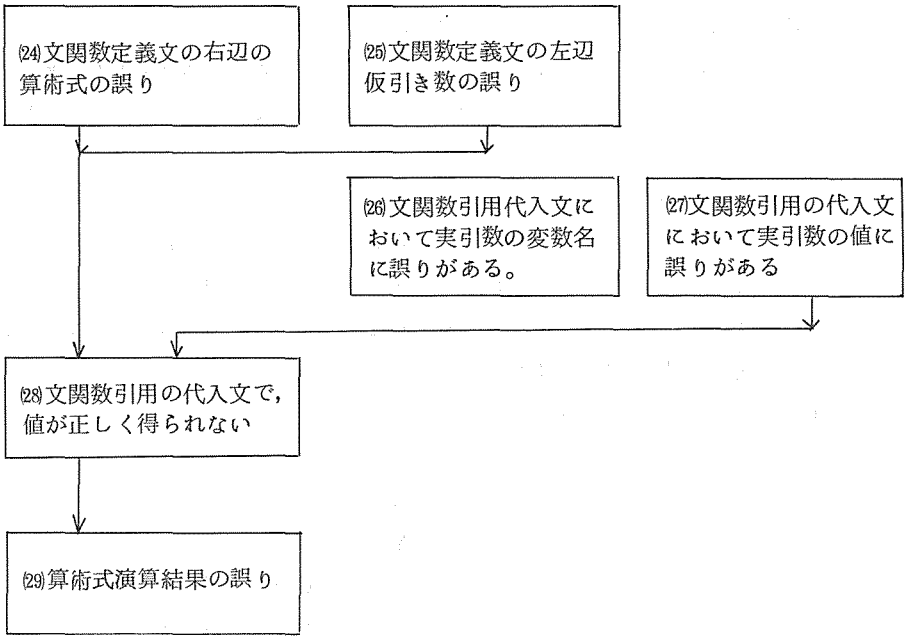

図3.5　文関数定義文に関するバッグとその症状。
矢印は上から下に向って因果関係を示す。

4. ま　と　め

今回の解説におけるＤＡＴＡ文，型宣言文，文関数定義文については，プログラムの中に記述されているステートメントが正しいか否かを判定する基準がほとんどの場合プログラマの意図に関係します。したがって，一般論としてのデバッグ手法の解説は，どうしても中途半端ないい方になってしまいます。そこで，本文の解説は“予測されるバッグの可能性をいくつか数えあげる„ことを主眼としました。バッグの可能性の多様さを理解していただければ，筆者の意図は満たされたものと考えております。

これまで9回にわたるバッグとデバッグに関する解説は今回をもって終了させていただきます。これまでの解説に例題として取りあげましたプログラム例のいくつかは，これまでに利用者諸氏から筆者の所へ御質問に来られました方々のものを借用させていただきました。毎回その都度おことわりすべきところですが，シリーズの最後のまとめにおいてここにお許しを願う次第です。このシリーズが利用者の方々のお役に立つことを願っております。

（お　わ　り）