Title	（第6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）：算術代入文
Author(s)	磯本, 征雄
Citation	大阪大学大型計算機センターニュース. 20 P.1-P.19
Issue Date	1976-02
Text Version	publisher
URL	http://hdl.handle.net/11094/65293
DOI	
rights	

大阪大学大型計算機センター・ニュース (No.20) 1976—2

解　　説

（第6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)

——算術代入文——

研究開発部　磯　本　征　雄

1．はじめに

フォートランは科学計算のために開発された計算機言語であり，その主な目的は数式の数値計算処理である。当然，フォートランでの算術代入文は重要な役割りをはたす文の1つであり，定式のプログラム化にとって非常に便利である。一方，フォートランでは算術式の記述が容易であるため，それがかえってプログラマーの緊張を欠き，実につまらない，しかしアルゴリズム上非常に重大な誤りを犯すことが多いようである。今回は，このような重要にもかかわらず誤りやすい算術代入文に関する様々な種類のバッグについて整理する。

フォートランにおける数式の記述の容易さが，却て初心者にとって非常に大きな誤解，錯覚の原因になる場合がある。フォートランでは，予め厳密に数学的検討のなされた定式をほとんどそのままプログラミングすることも可能であり，一見両者間には差異が見られないことが多い。しかし，数学的定式とフォートランにおける算術代入文の間には，その意味及び機能上大きな差異がある。特にフォートラン文法の上で，演算結果の精度に対する説明が少ないために，計算式の近似に関連する誤りが多い。その他，算術代入文の機能等に対する単純な誤解や理解不足が原因で重大な誤りを犯すことも多い。これらについては文法規則についての第2節の解説でも説明を加える。

次の問題点は，算術代入文に対するコンパイラーでのチェック事項に限度があるということに由来する。すなわち，コンパイラーでチェックされるのは，単に形式的な部分のみであって，当然のことながらプログラムの意味にまで立入らない。このために，現実には単純な誤りを無意識の内にプログラム中にしのび込ませることが多い。

例えば，算術代入文K＝J＋Mにおいて，Mを誤ってNと取り違えK＝J＋Nとしてしまってもフォートラン文法上誤りではない。当然，これはアルゴリズム上では誤りであり，しかもMとNの見かけ上の形が類似しているために，この取り違えに気づくことは容易でない。MとNはキーボード上で隣接した所にあり，このような取り違えはしばしば起こるようである。この種の誤りは、学術的意義やプログラミング技法についての一般論としてはほとんど意味をなさないであろう。しかし，時計のような精密機械の小ネジ1つの狂いが機械全体を狂わせるように，現実のプログラムにおいてこれらは最大級の誤りの1つとして考えておかなければならない。

この解説では，文法規則はJIS　FORTRAN（水準7000）[1]に従い，エラー・メッセージはFORTRAN-700[2]による。また，ここでのバッグに対する様々の解説は，NEAC 2200 シリーズModel 700を使った場合の状況を具体的に想定した上でおこなう。

2．算術代入文に関する文法規則

算術代入文は，変数名，定数又は配列要素名，等号及び算術式より成る。算術式は，次の規則によってつくられる[1]。

算術式　算術式(arithmetic expression)は，算術演算子と算術要素によって作る。算術式とその構成要素は，ともに整数型，実数型，倍精度実数型または複素数型の値を持つものとする。算術演算子(arithmetic operator)は，つぎのものとする。

演算子	意味
＋	加算，正符号
－	減算，負符号
＊	乗算
／	除算
＊＊	べき乗

算術要素(arithmetic element)は，1次子，因子，項，符号付項および算術式とする。

1次子(primary)は，定数，変数の引用，配列要素の引用，関数の引用またはかっこでくくられた算術式とする。

因子(factor)は，1次子かまたはべき，すなわち

1次子＊＊1次子

の形をしたものとする。後者の場合，左側の1次子を底，右側の1次子をべき指数という。

項(term)は，因子か　または　項／因子　または　項＊項　の形をしたものとする。

符号付項(signed term)は，項の直前に＋か－があるものとする。

算術式は，一つの項か，一つの符号付項か，またはそれらのあとにいくつかの項を＋や－でつないで並べたものとする。

べきのべき指数が整数型であれば，底の型が何であっても結果の因子の型は底の型に一致する。べき指数が実数型で底も実数型のときは因子は実数型，底が倍精度実数型のときは因子は倍精度実数型とする。べき指数が倍精度実数型であれば，底が実数型または倍精度実数型のとき，因子は倍精度実数型とする。
べきが定義されるのは以上の場合にかぎられる。

べき乗以外の算術演算子を使用することによって，同じ型の二つの要素を結びつけることができるものとする。その結果は，それらと同じ型とする。さらに実数型の要素は，倍精度実数型または複素数型の要素と結びつけてもよく，その結果の要素は，それぞれ倍精度実数型または複素数型とする。

例：算術式

```
1次子：
  3    12.3D－5    ALPHA    SIN( X)
  B(2＊1＋1,5)    (A＊＊2)
  (2.8＊A＋G＊F(K＋3))
  (－14.9,1.64E－4)
因子：
  ALPHA      (A＊＊2)＊＊3
  (2.8＊A＋G＊F(K＋3))＊＊3
項：
  ALPHA      ALPHA/(A＊＊2)＊＊3
  12.3D－5/B(2＊1＋1,5)
  (2.8＊A＋G＊F(K＋3))    SIN(X)
  (2.8＊A＋G＊F(K＋3))  ＊SIN(X)
  (－14.9, 1.64E－4)/GAMMA
符号付項：
  ＋12.3D－5      －ALPHA
  ＋(2.8＊A＋G＊F(K＋3))
算術式：
  ALPHA      ＋12.3D－5
  ALPHA/(A＊＊2)＊＊3
  －(A＊＊2)＊＊3＋12.3D－5/B(2＊I＋1,5) － (2.8＊A
  ＋G＊F(K＋3))＊SIN(X)
  －(14.9,1.64E－4)/GAMMA＋(6.55,7.89)
```

算術式は，言わゆる数学における数式とその表現形式において非常に類似している。特に，乗算が※により，ベキ乗算が※※により演算子として表わされることを例外とするならば，両者はほとんど同じである。

フォートランにおける算術式は，数式の如き静的なものではなく，算術式の内の特定の部分から逐次評価をすすめ，最後に全体の値が評価される。その手順を指示するための動的な命令である。ここで“特定の部分”及び“逐次”の意味は，計算機システムごとに少しづつ違う。また，値の評価のされ方も，変数の型に依存する。これら評価の手順については，JIS FORTRAN（水準7000）では次のように定められている。

式の評価　式の値を得るために，必要に応じて式の一部が評価されるものとする。

備　考　この結果，評価されない部分が生じることがある。

演算子の結合の強さは式の構成規則に依存するものとし，式の評価は，その式の構成順序に従って行なわれるものとする。一つの式を構成規則に従って構成してゆく順序はいろいろありうるが，式の評価は原則としてどの順序に従ってもよいものとする。

二つの要素が演算子によって結合されている場合には，要素の評価順序は任意とする。演算子が結合則や交換則やまたはその両方に従う場合には，かっこでくくられた式のまとまりが失われないかぎり，結合の順序を変更してもよいものとする。整数型の結果が得られる演算で，数学的な結果が整数にならない場合には，絶対値の小数点以下が切捨てられるものとする。結合則および交換則は除算を含む整数型の項の評価には適用しないものとし，このような項の評価は，左から右へと行なわれるものとする。

例：式の評価

（1）A＋Bという式では，A，Bのどちらかを先に評価してもよいし，これをB＋Aとして評価してもよい。

（2）（A＋B）＋Cのいう式では，（A＋B），Cのどちらを先に評価してもよいが，A＋(B＋C)という評価をしてはならない。

（3）K/2＊2という式では，Kは整数型の変数でその値を5とする。このとき左から右へ，すなわち，(K/2) ＊2と評価すべきである。この式の値は4となる。

この式を，たとえば，(2＊K)/2，(K＊2)/2，またはK＊(2/2) として評価してはならない。このとき，この式の値は5となる。

配列要素名を使うと，その添字の評価が必要になる。式の中に現われる関数が評価されるときに，その式やその式を含む代入文やＣＡＬＬ文のほかの要素の値を変更してはならない。

底が負の値を持ち，べき指数が実数型または倍精度実数型のべきは評価されないものとし，底がゼロでべき数がゼロのものも評価されないものとする。

値が数学的に定義されていない要素は，評価されないものとする。

参　考　ここでは通常の式の構造や評価の方法が定義されている。正常の法則に従うかぎり，評価の順序は原則として自由である。この自由性を許す目的は，処理系に計算機の備えている能力を最も効果的に利用させようとすることである。この自由性があいまいさを引起こさないように，文の内部での“副作用”を禁止する必要がある。すなわち，関数の引用は，その文の他の要素を定義したり，再定義したりしてはならない。

計算の際に，評価の順序が近似の程度に影響を及ぼすかもしれないから，それを使用者が制御できるように，処理系は勝手にかっこでくくったり，かっこをはずしたりしてはならない。

算術式において，演算子をはさんで同じ型の２つの要素あるいは異った型の２つの要素を結びつけることができる。(但し，ベキ乗については禁止されているものもある。) 算術式の評価の結果は，演算子を狭む２要素の型に依存する。これに関してはFORTAN-700については文献2) に示されているので参照して欲い。

算術代入文は，変数名又は配列要素名と等号と算術式によってつくられる。算術代入文は文法規則として次のように定められている。

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，表1に従ってvの値が変更されるものとする。

例：算術代入文

```
DELTA＝ALPHA/(A**2)**3
A(1,3)＝B(J,1)
Y＝－(A**2)**3+12.3D－5/B(2*1+1,5)－
    (2.8*A+G*F(K+3))*SIN(X)
I ＝I＋1
KAJ1＝109
XI＝XI+DELTAX
KAPPA＝(－14.9,1.64E－4)/(6.55,7.89)
```

表 1 eをvに割当てるための規則

v の 型	e の 型	割当ての規則*
整 数 型	整 数 型	代 入
整 数 型	実 数 型	整数化して代入
整 数 型	倍精度実数型	整数化して代入
整 数 型	複素数型	禁 止
実 数 型	整 数 型	実数化して代入
実 数 型	実 数 型	代 入
実 数 型	倍精度実数型	単精度化して代入
実 数 型	複素数型	禁 止
倍精度実数型	整 数 型	倍精度実数化して代入
倍精度実数型	実 数 型	倍精度化して代入
倍精度実数型	倍精度実数型	代 入
倍精度実数型	複素数型	禁 止
複素数型	整 数 型	禁 止
複素数型	実 数 型	禁 止
複素数型	倍精度実数型	禁 止
複素数型	複素数型	代 入

*ここで，

(a) 禁止はこの組合せが禁じられていることを示す。

(b) 代入は結果の値をそのままvに割当てることを示す。

(c) 整数化は結果の絶対値の小数点以下を切り捨て，整数型のデータの形に変換することを示す。

(d) 実数化は結果の値を実数型のデータの型に変換することを示す（8.2参照）。

(e) 倍精度実数化は，結果の値を倍精度実数型のデータの形に変換することを示す。

(f) 単精度化は結果の値の高位の部分をとりだして実数型のデータの形にすることを示す。

(g) 倍精度化は，結果の値を倍精度実数型のデータの形にすることを示す。

ここで，eをvに割り当てるための規則（表1）は，FORTRAN－700[2]では幾分異る所もあるので注意を要する（文献[2]表7－1参照）。表1の割当て規則は，演算された結果がもつ数値の精度を示すものであり，この点に注意を払っておかなければ，予想外に非能率的な計算を行うことになる。

算術代入文は数学の等式とは本質的に異るものである。例えば，単純な次の例を考えてみよう。

$$I = J \qquad (2.1)$$

$$J = I \qquad (2.2)$$

式(2.1)と式(2.2)は，数学としては本質的に同じものである。 一方，フォートランの算術代入文としてみた場合，両式(2.1)と(2.2)は全く異るものである。

式(2.1)：Jの値をIに入れる。この式の後，旧いJの値はIの値に置きかえられる。

式(2.2)：Iの値をJに入れる。この式の後，旧いIの値はJの値に置きかえられる。

このように，式の意味は異なる。すなわち，算術代入文は，右辺の算術式によって左辺の変数名又は配列要素名の値を定義するためのある種の操作又は命令であって，言わゆる単に等価であることを示す数学の等式とは異なる。もし，これらの点について充分な認識がなければ，デバッギングの際に意外な困難に遭遇するであろう。

3．算術代入文に関するコンパイル時のエラー・メッセージ

コンパイル時にコンパイラーより出力されるエラー・メッセージは，フォートラン文法にかかわる極く形式的事項に関してである。一方，次節で示すように，算術代入文に関するバッグは，形式的事項に関するものよりもむしろプログラムの意味にかかわるもののほうが多く，しかも深刻である。したがって，せめて形式的に発見されたコンパイル時のエラー・メッセージくらいは完全にデバッグしておく事を日常から心掛けておくことが必要である。以下に，万一不用意にも不完全なデバッグしかなされなかった場合に何如なる困難に遭遇するかを想定しながら，個々のエラー・メッセージに関するその意味及び具体例について解説する。

3.1.WARNING

コンパイラーは，WARNINGによって，フォートラン文法に関する誤解があるかもしれないことを，プログラマーに警告する。したがって，コンパイル時のWARNINGは，プログラム実行上必ずしも致命的誤りとはならない。WARNINGがあっても，プログラムはLINKされ実行される。それは，WARNINGとなった事柄が誤りであるか否かを判断できる者がプログラマーにほかならないからである。

ところが，WARNINGの出力された所には，えてして文法の誤解以上にプログラムの意味上の誤りが潜んでいる場合が多い。したがって，初心者に時々見掛けるように，WARNINGを無視して次の段階へ進むことは，デバッギングの点から見て危険である。このように，WARNINGは場合によってはFATAL ERROR以上にバッグ潜入の要因を示唆するものであるので，充分な注意を要する。以下，個々のWARNINGについて具体的に注意すべき事項を述べる。

メッセージ番号	説明
004	WARNING: NON-JIS COMBINATION OF DATA TYPES IN ARITHMETIC ASSIGNMENT ○算術代入文または右辺に算術式をもつ文関数定義文において，FORTRAN 700の文法では許されているが，JIS FORTRAN(水準7000)では許されない代入を行う。

(例)

```
COMPLEX     Z
      :
I=Z
      :
```

Iは整数型変数である。JIS FORTRAN（水準7000 ）では，整数型変数を複素数型変数で定義することは許されていない。プログラマーは，このような代入文を，計算機の特殊性を充分に承知の上で行っているのであろうか？ また代入文の意味は，すでに内部表現の問題にまで立入って吟味されているのであろうか？ もし，上式がこれらの問題に対して充分な吟味なしに使われている場合，ややもすればとんでもないバッグとなる事がある。

メッセージ番号	説明
006	WARNING: INITIAL VALUE FOR "name" CONVERTED ○DATA文または一般的型宣言文で与えられた初期値の型が，対応する変数または配列要素の型と異っており，その初期値は算術代入文の規則に従って変換された後に代入された。

(例)

```
DATA     J/10.0/
```

この例は，必ずしも算術代入文に対するものではない。しかし，変数Jがはたして実数型なのか整数型なのかが疑問となる場合の好例としてここに示した。すなわち，変数Jが後に算術代入文の中で実数として使われているならば，Jに対する実数型宣言が欠落している可能性を示唆するWARNINGである。

メッセージ番号	説明
025	WARNING: UNDEFINED VARIABLE "name" ○未定義の変数名が引用されている。即ちREAD文あるいはDECODE文の入力並び，EQUIVALENCE文，NAMELIST並びの要素，DOの制御変数，実引数のいずれとしても表われないか，あるいはDATA文や一般的型宣言文によって初期値を設定

	されていない変数名（仮引数となっているものあるいはCOMMON文中にあるものは除く)が，WRITE文の出力並び，ENCODE文の出力並び，代入文の右辺のいずれかに現われた。そのままコンパイルされる。

（例）

```
Y＝ＹＹ＋1.0
```

この例において，変数ＹＹは副プログラム中で一度も算術代入文の左辺に現われない。また他のいかなる方法によっても定義されない。

このWARNINGに対しては，

なぜYYが定義されなかったか？

YYは何か他の変数，例えばＹ等のミスパンチではないのか？

等の様々の誤りの可能性を吟味した上で，アルゴリズム全体から見て納得のゆく状態に修正すべきである。

メッセージ番号	説明
032	WARNING: REAL CONSTANT REGARDED AS DOUBLE PRECISION CONSTANT ○基本実定数の部分あるいは整定数の部分が10進で10けたを越えた実定数であるので，倍精度実定数と見なされてコンパイルされる。

（例）

```
Y＝１．２３４５６７８９０１２３
```

FORTRAN-700における単精度実数の有効桁数は10桁である。この例に見るように，13桁以上の実数は無意味である（文献[4]参照)。より高い精度を望むならば，倍精度で定義するほかない。

メッセージ番号	説明
033	WARNING: SIGNIFICANT DIGIT LOST IN DOUBLE PRECISION CONSTANT ○倍精度実定数の基本実定数部あるいは整定数部の有効けた数が21けたを越しているので，上位21けたがとられる。

（例）

```
DOUBLE     DX
      ⋮
DX＝１．２３４５６７８９０１２３４５６７８９０１２３４Ｄ０
      ⋮
```

FOTRAN-700において，倍精度実数型定数の有効桁数は21桁である。この例に示したように，もしそれ以上の高精度を望むなら，文法を再吟味の上で工夫が必要である。

メッセージ番号	説明
035	WARNING: TYPE MISMATCH IN COMPLEX CONSTANT OR DOUBLE PRECISION COMPLEX CONSTANT ○コンマで区切られた2つの一般の定数がかっこでかこまれているとき， (1) 両方共に一般の整定数であるか (2) 一方が実数型の一般の定数で，他方が整数型の一般の定数であるか (3) 一方が倍精度実数型の一般の定数で，他方が整数型あるいは実数型の一般の定数のいずれかである。(1)と(2)の場合は複素定数，(3)の場合は倍精度複素定数とみなされる。

(例)

```
COMPLEX     Z
      ⋮
Z=(1.0,  4)
      ⋮
```

複素数型定数は2つの実数型定数により定義される。

例における算術代入文の右辺にこれに対する矛盾がある。

3.2.FATAL ERROR

FATAL ERRORはコンパイラーにとって処理不可能な誤りである。したがって，FATAL ERRORを修正せずに次へ進むことはできない。ところで，ここで問題になるのは，エラーを如何に修正するかということである。単にFATAL ERRORメッセージの出力を防止するだけでなく，修正された結果がプログラムの意味としても正しくなければならない。特に，算術代入文ではこの点で非常にむづかしいものが多い。ここでは，エラー・メッセージの意味の説明に対する補足だけでなく，アルゴリズム上の困難とも関連させながら解説をみる。

メッセージ番号	説明
262	FLOATING POINT CONSTANT OUT OF RANGE ○実数型定数あるいは倍精度実数型定数が許される大きさ（-10^{616}～10^{616}）の範囲外である。DATAと一般的型宣言文における場合は，その定数の含まれる並びとそれに対応する変数名等がないものとしてコンパイルされるが，その文が含まれるプログラム単位はGOファイル上に出力されない。実行文における場合は，その文は削除されてコンパイルされ，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。

(例)

X＝10000.0E＋616 ＊A

ここでエラーとなっているのは定数10000.0E＋616である。-10^{616}～10^{616}の範囲外の数値は，FORTRAN-700では取扱えない(内部表現を調べればわかる)。どうしてもこのように大きな数値を取扱いたければ，数値を分割して取扱うほかない。できることならば，プログラミング以前の段階に立ち返り，このように大きな数値を取扱わなくてもよいように工夫することが望ましい。

メッセージ番号	説明
263	FIXED POINT CONSTANT OUT OF RANGE ○整定数あるいは整数型に変換される実定数が許される範囲 (-2^{47}～$2^{47}-1$)を越えている。DATA文と一般的型宣言文における場合は，この定数の並びとそれに対応する変数名等の並びがないものとしてコンパイルされるが，その文が含まれるプログラム単位はGOファイル上に出力されない。実行文における場合は，その文は削除されてコンパイルされ，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。

(例)

I＝1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9

整数型定数の2進数による内部表現において，-2^{47}～$2^{47}-1$の範囲しか取扱えないことに起因するエラーである。このように大きな整数を計算機で取扱わなければならないならば，2つの数の積に分解し，個々のものを別々に取扱って，最後までover flowが生じないよう工夫する必要がある。できることならば，プログラミング以前の段階に立ち返って，このように大きな数値を取扱わなくてもよいように工夫することが望ましい。

メッセージ番号	説明
286	ILLEGAL EXPRESSION SYNTAX ○式の中に文法違反がある。 実行文における場合は，その文が削除されてコンパイルされ，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。デバッグ文における場合は，そのデバック文が削除されてコンパイルされ，プログラムの実行時にそのデバッグ文により指定された範囲内の実行文に実行の制御が渡るとUEPになる。

(例1)

A＝B(1)C

(例2)

A＝B＊－C

例１は配列要素B(1)と変数Ｃの間に演算子が落ちている。

例２では，変数ＢとＣの間に演算子＊と－の２つがある。

このような場合，デバッギングの際には，元の定式と充分に照らし合せて修正しなければならない。なぜならば，例２に対して，修正の方法として

Ａ＝Ｂ＊Ｃ

Ａ＝Ｂ－Ｃ

Ａ＝Ｂ＊(－Ｃ)

等の様々の可能性があり，これらはいずれも文法的には正しい。しかし，アルゴリズムの面からいずれに正されるべきかを充分に検討しなければならない。

メッセージ番号	説　明
287	TOO MANY LEFT OR RIGHT PARENTHESES IN EXPRESSION ○式の中で左かっこの数と右かっこの数とが一致しない。実行文における場合は，その文が削除されてコンパイルされ，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。デバッグ文における場合は，デバッグ文が削除されてコンパイルされ，プログラムの実行時にそのデバッグ文により指定された範囲の実行文に実行の制御が渡るとＵＥＰになる。

（例）

Ａ＝（Ｘ＋Ｙ))＊Ｂ

ここでは，右カッコが左カッコに対して１つ多いことが誤りの原因である。ただし，単純に右カッコを１つ減すだけで良いか否かは問題である。このような場合，元の定式に立ち返ってそれらと比較しながら訂正すべきである。

メッセージ番号	説　明
290	ILLEGAL EXPONENTIATION OR NUMERIC OPERATION ON NON-NUMERIC DATA ○式の中で算術演算子の被演算子として許されない型のデータがあるか，べき乗演算において許されない組合せのデータがある。実行文における場合は，その文が削除されてコンパイルされ，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。デバッグ文における場合は，そのデバッグ文が削除されてコンパイルされ，プログラムの実行時にそのデバッグ文により指定された範囲内の実行文に実行の制御が渡るとＵＥＰになる。

（例）

```
STRING    ST
  ⋮
```

A＝X＋ＳＴ

⋮

ここで，Ａ及びＸは定数型変数であり，ＳＴはストリング宣言のなされた文字型変数である。したがって代入文Ａ＝Ｘ＋ＳＴにおいて，算術式Ｘ＋ＳＴは無意味な演算である。これらの代入文Ａ＝Ｘ＋ＳＴの修正において，変数Ａ，Ｘ及びＳＴの意味する内容を充分に吟味し，プログラム単位全体の様子を充分に調べなければならない。特にSTRING宣言された変数ＳＴがなぜ算術代入文で使われてしまったかが充分に理解されないままで，例えばSTRING宣言文を除去する等のことを軽率に行うべきでない。

4．算術代入文の実行時誤動作の振舞い

算術代入文に関する誤りは，プログラマーにとって重要でないと思われているものほど発見が遅れるようである。このことは，プログラム作成時におけるプログラマーの意識の集中度に関連しているように思われ，言わゆる人の心理の裏面を見る思いがする。このような理由から，ここでは組込み関数やベキ乗算等において生ずるバッグで，エラー・メッセージの出力される場合については議論を省略し，もっぱら表面に現われにくいバッグについて調べることにする。この種のバッグとは，言語形式の上からは正しいのではあるが，プログラムの目的又はその意図を考慮する時にのみバッグとして判断されるものである。

ところが，正しいプログラムとは何如なるものであるかは必ずしも自明ではない。したがって一見もっともらしく実行されたプログラムに対して，それが正しいか否かを決論づけることは容易ではない。結局，「プログラムは正しい」との判定を下すのは，プログラマーの判断にゆだねるほかない。このような理由から，ここではプログラマーが“バッグ有り”と判定を下した場合に，そのバッグの追跡されてゆく過程を筆者の経験を通して整理する。

さて，プログラムにバッグが紛れ込む状況をプログラム作成の段階から眺めると大きく次のように分類できるだろう。

Ⅰ）プログラムすべき定式を作る段階

Ⅱ）定式の数値計算法の工夫及び流れ図の作成段階

Ⅲ）フォートラン・プログラミングの段階

Ⅳ）プログラム・デッグの作成段階

ただし，このいずれの段階にバッグが紛れ込んだにせよ，それの発見される時点は，実行時である。したがって，この分類は必ずしも本質的ではないが，プログラマーが反省する上での参考にはなるであろう。以下，各段階ごとに解説する。

Ⅰ）プログラムすべき定式を作る段階

元の定式の誤りは，プログラミングにとって全く場違いの話しである。しかし，最終的に誤った結果を得る点では，プログラム自体の誤りと同等である。さらに，定式の誤りは，一度プログラムされてしまえば意外に発見しにくいことも事実である。いずれにせよこれらのことは，

常に念頭においておくことが必要である。いずれにせよ，

(1)定式の誤り

はプログラムの中をいくら調べても発見されず，単に

(2)出力数値に誤りがある

ことによってわかる。

Ⅱ）定式の数値計算法の工夫及び流れ図の作成段階

この段階で問題になる流れ図の作成については，算術代入文には直接関係しないので省略する。定式はプログラムとの間に１対１の対応関係はあるが，決して同等ではない。定式をフォートランで記述するためには，様々の教科書[3]にも示されているように，数値計算法上の注意を充分に払う必要がある。もし，

(3)数値計算法の上で工夫が欠けている，

場合には，

(4)桁落のため精度が悪くなる。

(5)丸め誤差の累積の結果，精度が悪くなる

等のことが起こる。(4)の桁落ちは，

(6)非常に近い数値間の引き算がおこなわれた

場合に発生する。例えば，次に示す有効桁数10桁の実数の引き算によっても理解できよう。

```
   1.234567891          1.234567891
-) 1.234566666       -) 0.034212345
-------------        --------------
   0.000001225          1.200355546
```

左側の式は有効桁数４桁に桁落ちしている。右側は，同じ10桁の数の引き算であっても，有効桁数は10桁に保存されている。このように，数値の精度の悪化は，時には数値解の精度に重大な影響をもつので，数値計算のための工夫を充分に検討する必要がある。この場合，値の近い数の間の引き算を避けなければならない*)(脚注参照)。

丸め誤差の累積(5)は，例えば

(7)絶対値に大差のある変数間の四則演算が多数回行われた

場合に起こる。１例として，有効桁数10桁の変数の和を考える。但し，計算はできるかぎり計算機と同様に行ってみる。

```
   1.234567891
+) 0.000000000 1234567891
-------------------------
   1.234567891
```

---

*）予め $X_1 \cong X_2$ であることが知れている場合，X_1 及び X_2 に対して座標変換をおこない

$$Y_1 = X_1 - X_0,\quad Y_2 = X_2 - X_0,$$

とすることにより，（$X_1 - X_2$）なる引き算を（$Y_1 - Y_2$）におきかえる。

$Z = X_1 - X_2$ を $Z = Y_1 - Y_2$ とすることにより，桁落ちを防ぐことができる。

FORTRANにおいて，加算の結果の和もやはり有効桁数の最大は10桁である。この結果，上式に見るように，和の値は加算を行う前の大きい方の数と同じで変化がなく，小さい方の数が全く無視される**)(脚注参照) このような計算が繰り返されると，その累積の結果が何如に大きくなるかが想像できるであろう。

上記(6)及び(7)の演算が一度行われると，その結果

(8)出力数値の精度が悪い，

ために，出力された数値の信頼性を欠くことになる。精度の悪さがさらに強調されれば，

(2)出力数値に誤りがある

こととなる。

一方，四則演算における数字の絶対値の大きさに関して見るならば，

(9)大きい数どうしの演算がくりかえされた

場合及び

(10)非常に小さい数で除算がおこなわれた

場合等において，いずれもその結果は非常に大きくなるために

(11) OVER FLOW

となり，数値上次のことがおこる。

(12)実数型の場合10^{616}を越えた

(13)整数型の場合，10^{14} を越えて，その符号が逆転した。

これらはいずれも，

(2)出力数値に誤りがある

ことによって確認される。

Ⅲ) フォートラン・プログラミングの段階

計算機によって実行されるのは，入力されたプログラムによって与えられた命令である。したがって，計算機にとって，プログラムがプログラマーの意図どおりに書かれたものであるか否かは問題にはならない。このため，フォートラン文法を誤解した状態でつくられたプログラムは，例えコンパイル時のエラー・メッセージの出力がなくても，当然プログラマーにとって好ましくない結果を与えるであろう。

ここで，

(14)フォートラン文法に対する誤解がある。

場合として，図4.1 にその実例を示す。

今，X 1 =1.4, X 2 =0.3であったとする。図4.1のプログラムを実行した後，

X 1 =0.0

X 2 =1.4

**) 実数の内部表現の面から調べればこの点はわかるであろう。これらの困難を避けるためには，大きさの近い数値どうしの演算から逐次処理すれば良い。

となる。プログラムの目的であるＸ１とＸ２の数値の入れかたはうまくいっていない。これは，途中で一時的に使われた変数ＩＸが暗黙の整数型変数であり，このために小数部分が無視されることになる。これらのことから，次のことが結論づけられる。

```
      ⋮
      IX＝X2
      X2＝X1
      X1＝IX
      ⋮
```

図4.1　実数型変数Ｘ1とＸ2の数値を入れかえるためのプログラム。ここでＩＸに対して，実数型宣言文はないものとする。

⑮変数に対する型宣言に誤りがある場合には，

⑯整数型変数を実数として使い，これらに対して実数型宣言がなされていない

ために，算術代入文をくりかえしてゆくうちに

⑰実数型変数において，小数点以下の桁が切り捨てられる

ことになる。この結果，

⑱出力数値の精度が悪い

だけでなく，ゼロとならないはずの実数において

⑲数値がゼロとなった

等のことが生ずる。

この結果がさらに次の代入文の算術式に派生して

⑳エラー・メッセージZERO　DIVIDEが出力される

ことにもなる。

実数型と整数型の変数の型の取り違えと同じことが，実数型と倍精度実数型についても起こる。すなわち，

㉑倍精度実数型として使っているつもりの変数に対して，倍精度実数型宣言がなされていない，ために

㉒倍精度実数型変数の精度が悪くなる。

さらに，文法に対する誤解の結果，時には次のようなバッグもある。

㉓READ文，DATA文による変数の初期値設定の方法に誤りがある

この時には，例えプログラム中に他の誤りがなくても，

㉔出力数値に誤りがある

ことになる。DATA文における変数名と定数間の1対1の対応関係に誤りがある場合や，READ文による入力データの誤りや入力変数の定義の誤り等がその具体例である。但し，これらは今回の主旨でないので詳細は省略する。

フォートラン文法に関する誤解に対して，直接バッグとしてはねかえってくる例として，ほかに

㉕式の評価に関して誤解がある

場合である。

(24)例えば，次の如き誤解が存在すると，それはバッグとなる

$$\left[\begin{array}{l} y = (-x)^2 \text{ なる式を} \\ Y = -X**2 \text{ とプログラムした。} \end{array}\right]$$

このような誤りでも，当然プログラム実行結果の

(2)出力数値に誤りがある。

Ⅳ）プログラム・デックの作成段階

プログラム・デック作成段階で生ずる誤りは，一般にその原因がはなはだあいまいである。多くのプログラマーの経験するバッグは

(25)ミスパンチや文の順序の入れ換わり等の不注意による誤りによって入り込むものである。これらのバッグは，具体的には次のような状態でプログラム中にひそんでいる。

(26)算術式中の変数名に誤りがある。

(27)算術式中の演算子の使い方に誤りがある。

(28)変数値定義の手順に誤りがある。

これら(24)，(25)及び(26)のいずれも，次のように大ずかみに知ることができる。

(29)変数の定義（算術代入文）に誤りがある。

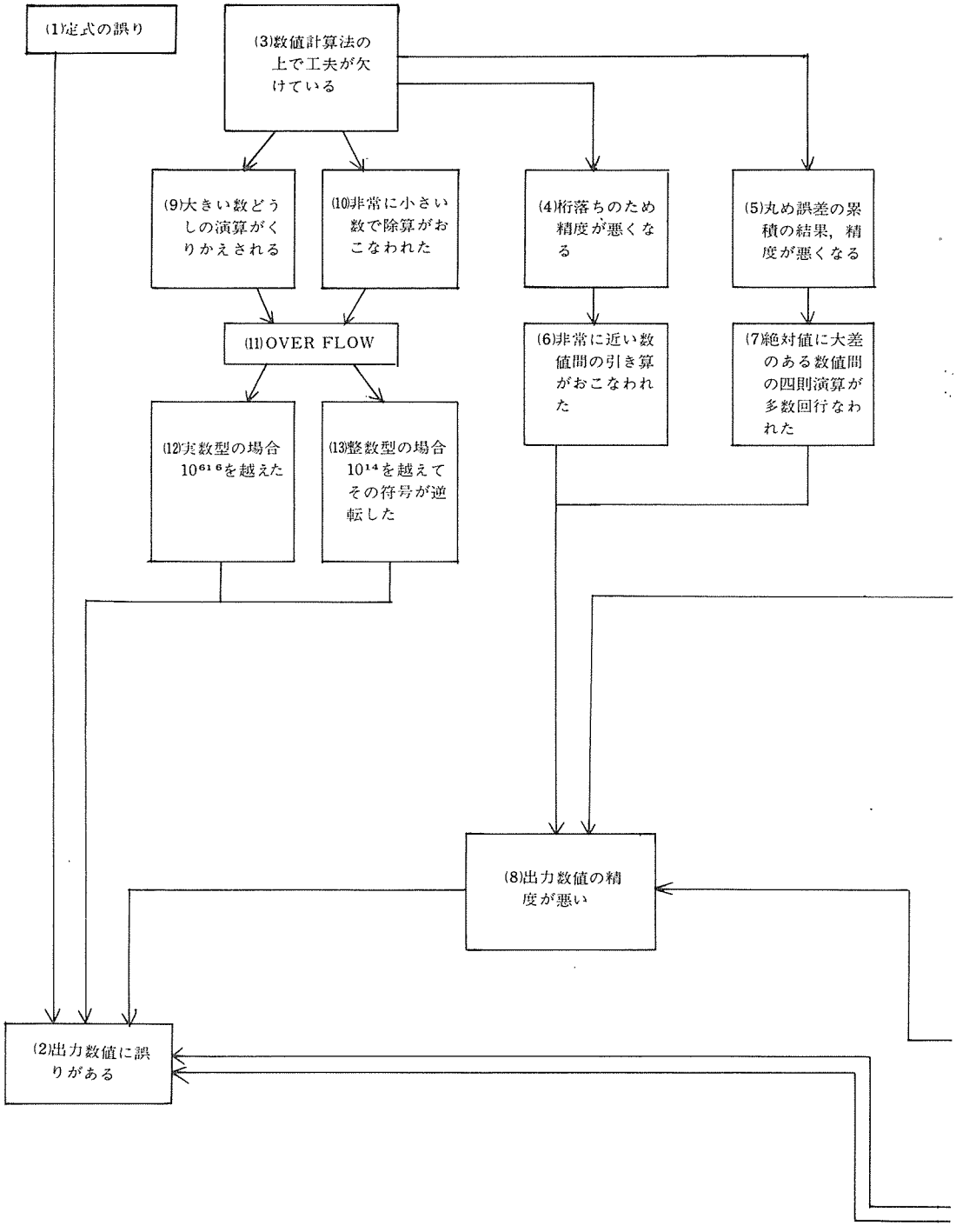
プログラム・デバッギングの過程でこれらは，

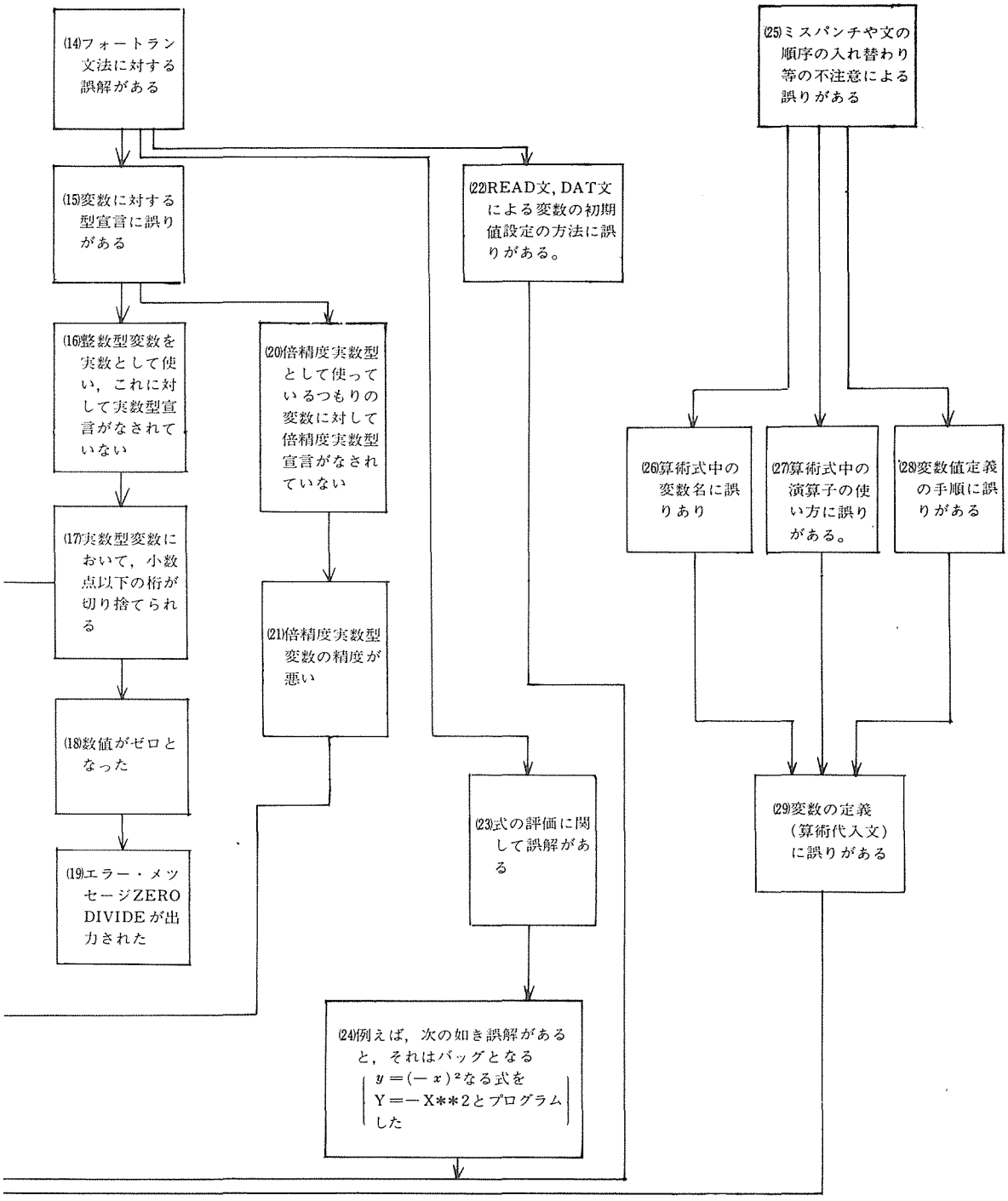
(2)出力数値に誤りがある。

ことにより確認される。

図4.2には，以上の事象の関係を原因→結果の順序付けによりまとめた。これらをたどることにより，デバッグの際の参考になるであろう。

図4.2 算術代入文の実行時における誤動作の振舞い




(14)フォートラン文法に対する誤解がある
(15)変数に対する型宣言に誤りがある
(16)整数型変数を実数として使い，これに対して実数型宣言がなされていない
(17)実数型変数において，小数点以下の桁が切り捨てられる
(18)数値がゼロとなった
(19)エラー・メッセージZERO DIVIDEが出力された
(20)倍精度実数型として使っているつもりの変数に対して倍精度実数型宣言がなされていない
(21)倍精度実数型変数の精度が悪い
(22)READ文，DAT文による変数の初期値設定の方法に誤りがある。
(23)式の評価に関して誤解がある
(24)例えば，次の如き誤解があると，それはバッグとなる
y＝(－x)²なる式を
Y＝－X**2とプログラムした
(25)ミスパンチや文の順序の入れ替わり等の不注意による誤りがある
(26)算術式中の変数名に誤りあり
(27)算術式中の演算子の使い方に誤りがある。
(28)変数値定義の手順に誤りがある
(29)変数の定義（算術代入文）に誤りがある

5．まとめ

前５回を含めてこれまでの解説に見たように，バックは誰にでも経験されていながら，いざデバッギングの際にその存在場所を発見することは意外に困難である。特に，算術代入文に関するバッグの現れ方は，個々のプログラム固有の特徴やプログラマーの癖に依存することも少くない。また，本文にも述べたように，往々にしてプログラマーが重要とは感じていない所にバッグが潜んでいるものである。これらバッグの現れ方の意外性のため，バッグの総ての可能性を網羅することは到底不可能である。したがって，本解説により〝様々のバッグがあり得るものだ〟との感想を懐いていただければ，筆者の目的の大半は達成されたものと考えている。

筆者が過去に経験したバッグを整理したにすぎない本解説は，算術代入文に関するバッグの極く一部分にすぎないと思う。これらに対するより広範囲の整理をすすめるため，利用者諸兄の本センター・ニュースへの寄稿による御協力を期待いたします。

参考文献

1）ＪＩＳハンドブック，情報処理，日本規格協会（1973）

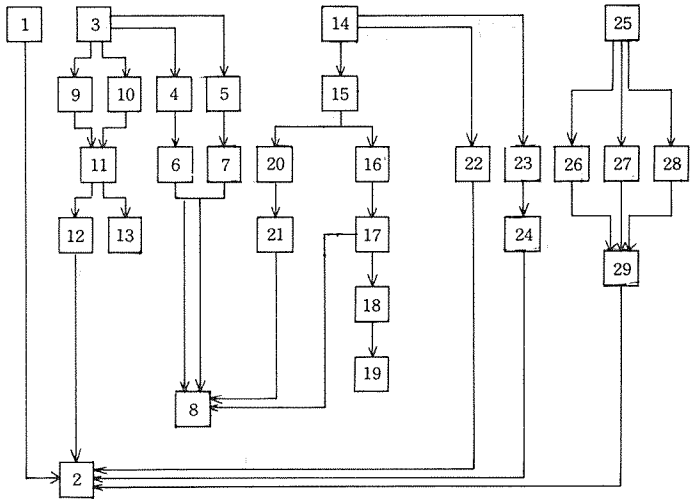
2）ＮＥＡＣシリーズ2200オペレーティングシステムＭＯＤⅣＥＸ／ⅦＦＯＲＴＲＡＮ-700文法説明書ＮＥＣ日本電気株式会社

3）Pat H. Sterbenz: Floating-Point Computation, Prentice-Hall Series in Automatic Computation, Prentice-Hall, Inc.（Engelwood Cliffs, New Jersey），（1974）．
戸川隼人；計算機のための誤差解析の基礎，サイエンス社，（1974）

4）ＮＥＡＣシリーズ2200オペレーティングシステムＭＯＤⅣＥＸ／ⅦＦＯＲＴＲＡＮ-700プログラミング説明書ＮＥＣ日本電気株式会社

付　録

算術代入文の実行時における誤動作の振舞い



(1) 定式の誤り。
(2) 出力数値に誤りがある。
(3) 数値計算法の上で工夫が欠けている。
(4) 桁落ちのため精度が悪くなる。
(5) 丸め誤差の累積の結果，精度が悪くなる。
(6) 非常に近い数値間の引き算がおこなわれた。
(7) 絶対値に大差のある数値間の四則演算が多数回行なわれた。
(8) 出力数値の精度が悪い。
(9) 大き数どうしの演算がくりかえされた。
(10) 非常に小さい数で除算がおこなわれた。
(11) OVER　FLOW
(12) 実数型の場合，$10^{61\,6}$を越えた。
(13) 整数型の場合，10^{11}を越えて，その符号が逆転した。
(14) フォートラン文法に対する誤解がある。
(15) 変数に対する型宣言に誤りがある。
(16) 整数型変数を実数として使い，これに対して実数型宣言がなされていない。
(17) 実数型変数において，小数点以下の桁が切り捨てられる。
(18) 数値がゼロとなった。
(19) エラー・メッセージZERO　DIVIDEが出力された。
(20) 倍精度実数型として使っているつもりの変数に対して倍精度実数型宣言がなされていない。
(21) 倍精度実数型変数の精度が悪い。
(22) READ文，DATA文ぬよる変数の初期値設定の方法に誤りがある。
(23) 式の評価に関して誤解がある。
(24) 例えば，次の如き誤解があると，それはバッグとなる。

{ y＝(－x)²となる式を
Y＝－X＊＊2とプログラムした }

(25) ミスパンチや文の順序の入れ替わり等の不注意による誤りがある。
(26) 算術式中の変数名に誤りあり。
(27) 算術式中の演算子の使い方に誤りがある。
(28) 変数値定義の手順に誤りがある。
(29) 変数の定義（算術代入文）に誤りがある。