プログラム・ノート

（第３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）

DIMENSION 文
COMMON 文
EQUIVALENCE 文
EXTERNAL 文

研究開発部 磯 本 征 雄

1．はじめに

フォートランは4則演算を主な機能としてもつ計算機言語である。したがって変数や配列要素の値を定める式や代入文等が重要な役割をはたすことは当然である。しかし，せっかく正しい値を得ても，それらが正しい取扱いのもとで正しく記憶されていなければ，次の段階でその変数又は配列要素を引用した時には誤った結果になる。このような変数や配列の取扱い方や記憶場所を定めるのが宣言文である。

実行文がプログラムの振舞いであるとすれば，非実行文である宣言文はプログラムの振舞いの場を定める文であるといえる。フォートラン・プログラミングにおいては，とかく実行文の実行の流れの追跡に注意が向くために，非実行文である宣言文が注意力の盲点になりやすい。このために，宣言文におけるバッグは，発見が困難である。またデバッグも実行文との関連を無視できないためにさらに困難である。宣言文は，副プログラム—副プログラム間，宣言文—実行文間及び宣言文—宣言文間等のように多くの場合2個所以上の文の間での文法上あるいは意味上の矛盾がバッグの原因となる。したがってプログラムにおける局所的検査だけではバッグの発見はできず，広範囲にわたって変数や配例の値の振舞いの矛盾を調べなければならないことが多い。このことは，プログラム化すべきもとの式の論理が正しくフォートランで表現されているか否かということのほかに，さらにプログラムの枠内でフォートランで矛盾する事項はないかという点を調べなければならないことになる。

このように宣言文は影の立役者とでも言うべき重要なものである。今回は式や代入文の解説に先駆けて標題の4宣言文について解説する。文法はJIS-FORTRAN（水準7000）による。また解説の形式と手順はすべて第2回に同じにした。

2．配列宣言

2.1，配列宣言に関する文法

配列により，1群の変数を行列として取扱うことができる。フォートラン文法においては，

配列は次元の数と各次元ごとに定められる寸法により規定される。これらの配列の次元の数と寸法を規定することを配列宣言という。配列宣言は次の形式でなされる。

配列宣言子

配列宣言子(array declarator)は，プログラム単位で使用する配列を規定する。

配列宣言子は，英字名，次元の数およびそれぞれの次元の寸法を示す。配列宣言子は配列宣言文，DIMENSION文またはCOMMON文の中に現われることができるものとする。配列宣言子を含む文を配列宣言子文(array declarator statement)という。

配列宣言子は，つぎの形とする。

v(i)

ここで，vは英字名とし，宣言子名(declarator name)という。

(i)を宣言子添字(declarator subscript)といい，iは1個，2個あるいは3個の式から成り(但しNEAC FORTRAN N-700では7次元まで許される)，それぞれの式は寸法(dimension)といい，整定数か整数型の変数名とする。式が2個以上ある場合には後続する式との間はコンマで区切る。iの中に変数名が含まれていない場合には，(i)を定数宣言子添字(constant declarator subscript)といい，そうでない場合，すなわち一つでも変数名が含まれているならば(i)を整合宣言子添字(adjustable declarator subscript)といい，その変数を整合寸法(adjustable dimension)という。

例：配列宣言子

A(10)　B(5, 12)　C(7, 8, 3)

D(I, 5)　E(M)

配列宣言子における次元の数及び寸法は次の規則に従う。

配列宣言子文の中に現われる宣言子添字は，その宣言子名が配列名であることを処理系に知らせるものとする。宣言子添字の式の個数は，その配列の次元の数を示す。宣言子添字のそれぞれの寸法の値は，配列要素名の中でそれぞれの添字式がとりうる最大値を示す。

実行可能プログラムの実行中には，配列要素名は1未満あるいは配列宣言子で指定された寸法より大きな値を持つ添字式を持つことはできない。

配列宣言子における次元の数と宣言子添子は配列要素の数を与える。それらは次の規則による。

配列要素関数と添字の値　次元の数，宣言子添字，添字が与えられたとき，それに対応する添字の値および添字がとりうる最大値を表に示す。添字式の値は1以上でなければならない。

例：配列要素関数と添字の値

配列宣言子F(2，3)によって規定された配列Fの配列要素は，つぎの順序に並んでいる。

F(1, 1), F(2, 1), F(1, 2), F(2, 2), F(1, 3), F(2, 3)

配列要素関数(array element successor function)の値は，添字の値に1を加えたものとする。ある配列要素の配列要素関数の値と等しい値の添字をもつ配列要素を，はじめの配列要素の直後の要素とする。配列の要後の要素時，添字の値が最大のものとし，これには直後の要素はない。

表　添字の値

次元の数	宣言子添字	添 字	添 字 の 値	添字の最大値
1	(A)	(a)	a	A
2	(A, B)	(a, b)	a+A・(b−1)	A・B
3	(A, B, C)	(a, b, c)	a+A・(b−1)+A・B・(c−1)	A・B・C

ここで，a, b, cは添字式とし，A, B, Cは寸法を示す。

配列宣言のみをその機能としてもつ宣言文が，DIMENSION文である。DIMENSION文は次の形式で表現される。

DIMENSION文　DIMENSION文(DIMENSION statement)は，つぎの形とする。

DIMENSION $v_1(i_1), v_2(i_2), \cdots, v_n(i_n)$

ここで，$v_1(i_1), \cdots, v_n(i_n)$は，いずれも配列宣言子とする。

例：DIMENSION文

DIMENSION A(10), C(7, 8, 3)

2.2　コンパイル時における配列宣言に関するエラー・メッセージ

以下，FORTRAN-700によるエラー・メッセージについて具体的例を添えて説明する。

2.2.1　WARNING

メッセージ番号	説　明
060	WARNING: ARRAY ELEMENT OF “name” USED IN STATEMENT FUNCTION EXPRESSION—ACCEPTED ○文関数定義文の中で配列要素を引用している。そのままコンパイルされる。 (プログラム　コード　-02_8)

(例)

```
内部文番号        フォートラン文
0002              DIMENSION F(10,10)
0003              G(I, J, X)=F(I, J)*X*FLOAT(I+J)
                  ⋮
```

配列要素を文関数（内部文番号０００３）内で使う時は，添字は整定数のみゆるされる。したがって上記例題はＦ（Ｉ，Ｊ）の使用の点でまちがいである。

2.2.2 FATAL ERROR

240	NON-DATA NAME USED ON LEET SIDE OF ASSIGNMENT STATEMENT ○代入文の左辺にデータ名を表わす英字名以外のものが現われた。この代入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の実行に制御が渡るとＵＥＰになる。

（例）

内部文番号	フォートラン文
０００１	ＤＩＭＥＮＳＩＯＮ　Ｂ（１０）
	⋮
００１０	Ａ（Ｉ）＝ＦＬＯＡＴ（Ｉ＋１０）
	⋮

配列宣言のなされていない配列要素Ａ（Ｉ）が代入文の左辺にあらわれている。

そのほか，次の場合にも上記エラー・メッセージが出力される。

例：EXTERNAL SUB
　　SUB = A * B + C

例：REAL A(10)
　　A/5) = SIN(X)

256	VARIABLE NAME "name" FOLLOWED BY LEFT PARENTHESES ○変数名の後に左かっこがある。実行文の場合に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。非実行文の場合にはその文が削除されてコンパイルされるが，その文を含むプログラム単位はＧＯファイル上に出力されない。デバッグ文の場合にはそのデバッグ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デバッグ文により指定された範囲内の文に実行の制御が渡るとＵＥＰになる。

（例１）

内部文番号		フォートラン文
０００８		ＤＯ　１０　Ｉ＝１，１０
０００９		Ａ　＝　Ｉ　（Ｊ）
		⋮
００２０	１０	ＣＯＮＴＩＮＵＥ

例では当然配列要素にはなり得ないDO文の制御変数に添字がつけられて誤りとなった。このほか，配列宣言を落したために配列要素が変数とみなされ上記エラーを生じる場合もある。

（例２）

内部文番号	フォートラン文
0001	DIMENSION　A (5, 10
⋮	⋮
0010	A (I, J) =B (I, J)
⋮	⋮

DIMENSION文のA（5，10が誤りである。したがって配列宣言は無視される。これは，配列Aについての配列宣言が欠落した場合の誤りと同じ結果になる。

276	MORE THAN 7 SUBSCRIPTS OR SYNTAX ERROR IN REFERENCE TO ELEMENT OF ARRAY "name" ○7次元を越える添字があるか，配列要素の参照に文法違反がある。 ＤＡＴＡ文における場合は，この配列要素を含む並びとそれに対応する定数の並びがないものとしてコンパイルされ，EQUIVALENCE文における場合は，この配列要素を含む並びがないものとしてコンパイルされ，いずれの場合もこの文を含むプログラム単位はＧＯファイル上に出力されない。実行文における場合は、その実行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。デバッグ文における場合は，そのデバッグ文は削除され，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）

内部文番号	フォートラン文
	⋮
0010	X=A (I1, I2, I3, I4, I5, I6, I7, I8)
	⋮

配列要素が８次元として代入文の右辺に用いられている。配列は７次元までしか許されない。

（例２）

内部文番号	フォートラン文
	⋮
0011	X=A (I1, I2, I3,)
	⋮

配列要素において添字の第４番目が欠けているか又は最後のコンマが余分についている。

277	THE NUMBER OF SUBSCRIPTS USED EXCEEDS THE NUMBER OF DIMENSIONS OF ARRAY "name" ○配列要素の使用で，添字が定義した次元の数を越えている。DATA文における場合は、その配列要素を含む並びとそれに対応する定数の並びがないものとしてコンパイルされ，EQUIVALENCE文における場合は，その配列要素を含む並びがないものとしてコンパイルされ，いずれの場合もその文を含むプログラム単位はＧＯファイル上に出力されない。実行文中における場合は，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)

```
内部文番号          フォートラン文
0001                DIMENSION  B (2, 3, 4)
 ⋮                        ⋮
0010                X=B (I1, I2, I3, I4)
```

配列Ｂ（２，３，４）において配列宣言は３次元であるのに文番号００１０において４次元として使用している。但し内部文番号００１０がX=B（Ｉ１，Ｉ２）の場合はFATAL ERRORにならない。

280	CONSTANT SUBSCRIPT EXCEEDS RANGE OF ARRAY "name" ○定数だけの添字式をもつ配列要素の使用で，添字の値が配列宣言子により定められた大きさを越えている。ＤＡＴＡ文における場合は、その配列要素を含む並びとそれに対応する定数の並びがないものとしてコンパイルされ，その文を含むプログラム単位はＧＯファイルに出力されない。実行文中における場合は，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。デバッグ文中における場合は，そのデバッグ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)

```
内部文番号          フォートラン文
0001                DIMENSION  A (10)
 ⋮
0010                X=A (12)
```

配列宣言での配列Aの添字の大きさは10であるのに，代入文ではA（12）で，宣言を越える定数添字が用いられている。但しAは手続き副プログラムの仮引数ではない。

408	CONFLICTING USE OF NAME "name" ○矛盾した名前の使用がある。これらの文を含むプログラム単位がはGOファイル上に出力されない。

（例1）

内部文番号	フォートラン文
0001	DIMENSION　L (5, 10)
	⋮
0050	L (I, J) =K (I, J)
0051	K (I, J) =I*J+2*I+3*J+1
	⋮

配列要素K（I，J）は配列宣言がなされていない。したがって内部文番号0050におけるKは関数副プログラムであるとみなされる。しかし内部文番号0051においてK（I，J）が代入文の左辺にあらわれた。エラー番号256に類似しているが，文の前後の状態が異なる。ここでは外部関数Kが一つのプログラム内で引用されると同時に代入文の左辺にもあらわれたものとみなされる。そのほか次の場合もエラー408の誤りである。

（例2）

```
NAMELIST/A/B, C
COMMON/D/A
```

ブロックDとその要素AはCOMMON文にないものとしてコンパイルされる。

（例3）

```
EXTERNAL MYFUNC
DATA MYFUNC/1. 51/
```

外部手続き名MYFUNCと定数1.51はDATA文にないものとしてコンパイルされる。

（例4）

```
EXTERNAL A, B
NAMELIST /C/ A, B
```

NAMELIST名Cとその要素A，BはNAMELIST文にないものとしてコンパイルされる。

410	MORE THAN 7 DIMENSIONS ARE DECLARED FOR "name" ○配列宣言子で次元の数が7を越えている。 その配列宣言子の7次元までの寸法をとり，8つ目以降の寸法を無視してコンパイルされるが，この配列宣言子を含むプログラム単位はGOファイル上に出力されない。

(例)　内部文番号　　フォートラン文

0001　　DIMENSION A (1, 2, 3, 4, 5, 6, 7, 8)

配列は7次元までしか許されない。上記配列宣言は8次元として宣言されている。

411	ILLEGAL DIMENSION DECLARED FOR ARRAY "name" ○配列宣言子文の宣言子添字に誤りがある。即ち寸法に整定数か整定数以外のものが使用されているか，あるいは寸法の区切り記号に誤りがある。この配列宣言子はないものとしてコンパイルされるが，この配列宣言子を含むプログラム単位はGOファイル上に出力されない。

(例)　内部文番号　　フォートラン文

0001　　DIMENSION ARRAY (20, 20. 10)

⋮　　⋮

配列宣言においてARRAY (20,20,10) とすべき部分で，コンマについて誤りがある。

414	CONFLICTING USE OF DECLARATOR NAME "name" ○宣言子名としてNAMELIST名か手続き名と同じ名前が使われている。この配列宣言子はないものとしてコンパイルされるが，この配列宣言子を含むプログラム単位はGOファイル上に出力されない。

(例)　内部文番号　　フォートラン文

0001　　DIMENSION A (10)

0002　　EXTERNAL A

⋮

外部手続き名Aに対して配列宣言がなされている。配列宣言は配列に対してのみ有効である。

415	ARRAY "name" IS DOUBLY DECLARED WITH REDUNDANT OR CONFLICTING DIMENSIONS ○同一の宣言子名をもつ配列宣言子が2回以上現われた。2番目以降に現われた配列宣言子はないものとしてコンパイルされるが，このプログラム単位はＧＯ ファイル上に出力されない。

(例)　内部文番号　　フォートラン文

0001　　DIMENSION A(10, 10), A(100)

⋮　　⋮

同じ配列名について2度配列宣言がなされている。配列宣言は1つの主プログラム又は副プログラムに1回のみでなければならない。

**416	SIZE OF ARRAY "name" IS TOO LARGE ○配列宣言子の添字の値が大きすぎる。許される最大の大きさは整数型，実数型，論理型と文字型の場合は$2^{18}-1$，倍精度実数型と複素数型の場合は$2^{17}-1$，倍精度複素数型の場合は$2^{16}-1$である。この配列には許される最大の語数が割付けられてコンパイルされるが，この配列宣言子を含むプログラム単位はＧＯ ファイル上に出力されない。

(例)　内部文番号　　フォートラン文

0001　　DIMENSION A(1008100)

⋮　　⋮

配列の大きさA(1008000)が大きすぎる。但し上の例はA(100,100)をミス・パンチにより誤ったものでありＥＬ一コードでは同じ鍵盤上に「,」と「8」があることから時々生ずる誤りであるげ。

421	ILLEGAL DIMENSION STATEMENT SYNTAX ○DIMENSION文に文法違反がある。適当な仮定のもとにコンパイルされるが，いずれの場合においてもこのDIMENSION文を含むプログラム単位はＧＯ ファイル上に出力されない。

(例) DI MENSION文についての表現上の誤りである。

(1) DIMENSION　　は文を削除される。

(2) DIMENSION A(15), 2B(20), C(50) は

DIMENSION A(15), B(20), C(50) と仮定される。

(3) DIMENSION A(15). B(50)は

DIMENSION A(15), B(50)と仮定される。

このようにエラー・番号421は表現形式上の誤りに関するものである。

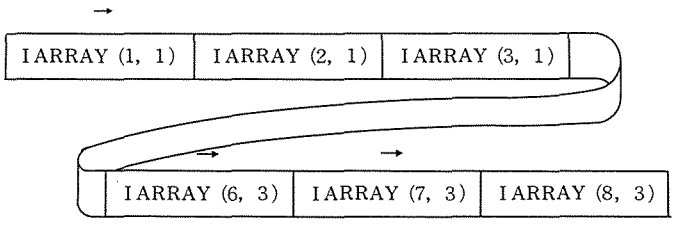
2.3 リンクロード時，実行時における配列宣言の誤り

配列要素の表現形式には2つの点であいまいさがある。第1に文関数や外部関数と同じ形式で表現されていることである。すなわち，英字名の後にカッコがついて，カッコの中に変数（但し整数型）がコンマで区切られて配置されている。第2に配列要素は次元の数と寸法により表現されているにもかかわらず，計算機の主記憶に記憶されているのは一次元的に並べられた(数)値の列であるにすぎない。

プログラミングの手順の上からみて，第1番目の問題点は概ねリンクロード時までに何らかの形でエラー・メッセージが出力されて発見できる。したがってバッグ発見のための困難度は低い。第2番目の問題点は，プログラム作成者自身のプログラムに対する意味付けとの関連が非常に強い。配列が記憶される場合に，要列要素は1列に並べられるにすぎない。したがって配列要素の全体の数が重要であって，寸法のみ又は次元の数のみについては文法上及び計算機の機能上から見て一概に誤りとはいえない。すなわち，配列宣言について問題が生じた場合には，配列要素や配列宣言子の添字の数や寸法がフォートラン文法としてでなく，プログラムで実行したい事項の意味を考えた上で，主記憶上の配列要素の記憶場所との関係において配列要素や変数の記憶や引用が正しく処理されているか否かを確かめる必要がある。特に文法についての誤解や配列要素の記憶されている状況についての誤解は，しばしば“アルゴリズムは正しいけれども最終結果がおかしい”といった種類の誤りになっている。

誤りについて解説に入る前に配列について，各要素の記憶場所のようすを示しておく。文法説明のところでも述べたように例えば配列 IARRAY (8, 3) は図2.1で示された

図2.1 配列宣言子 IARRAY (8, 3) に対する配列要素の記憶



順序で一列に（1本のテープ上に並べられたように）配置される。仮りに配列を IARRAY (24) としたならば，配列要素の値は図2.2に示された順序に1列に配置される。上記図2.1と図2.2に示された2つの手法は値の記憶という点からは全く同じである。

したがって配列宣言子が例え2次元で記述されていても，文法で示された添字の相対的位置が同じであれば，実行文中で1次元配列として用いることができる。但し，この逆は誤りである。

図2.2　配列宣言子 IARRAY (24) に対する配列要素の記憶順序

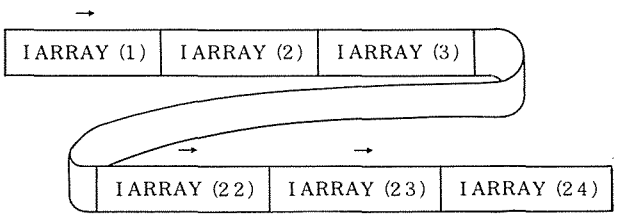


図2.3は配列宣言 IARRAY (8, 3) が2次元でなされた配列を，実行文中で2次元配列として用いた場合と1次元配列として用いた場合の効果が同等である事を示す例である。プログラム中で配列宣言は DIMENSION IARRAY (8, 3) とされている。内部文番号0002～0004 において2次元配列に数値を代入する。内部文番号0005～0006 で IARRAY (I1, I2) として2次元配列のままで出力をおこなう。

次に内部文番号 0008～0011 において配列を IARRAY (I) として1次元配列に見立ててWRITE文で出力した。内部文番号0009～0010は，出力結果が上記2次元配列と同じにするための添字の変換式と考えてよい。図2.3の下部には2つのWRITE文による出力結果を示す。添字について調整することにより全く同じ結果を役ることがわかる。

図2.3　2次元配列と1次元配列の比較

```
0001          DIMENSION IARRAY (8,3)
0002          DO 10 I1=1,8
0003          DO 10 I2=1,3
0004       10 IARRAY (I1,I2)=10*I1+I2
0005          DO 30 I1=1,8
0006       30 WRITE (6,1000)    ((I1,I2,IARRAY (I1,I2)),I2=1,3)
0007      100 FORMAT (1H ,3 (' (',I1,',',I1,')',1X,I2,':',3X))
0008          DO 40 I=1,8
0009          I2=I+8
0010          I3=I+16
0011       40 WRITE (6,200) I,IARRAY (I),I2,IARRAY (I2),I3,IARRAY (I3)
0012      200 FORMAT (1H ,3 ( ( I3,')',1X,I2,':',3X))
0013          STOP
0014          END
```

WRITE文による出力結果

```
(1,1) 11:   (1, 2) 12:   (1,3) 13:
(2,1) 21:   (2,2) 22:   (2,3) 23:
(3,1) 31:   (3,2) 32:   (3,3) 33:
(4,1) 41:   (4,2) 42:   (4,3) 43:
(5,1) 51:   (5,2) 52:   (5,3) 53:
(6,1) 61:   (6,2) 62:   (6,3) 63:
(7,1) 71:   (7,2) 72:   (7,3) 73:
(8,1) 81:   (8,2) 82:   (8,3) 83:
( 1) 11:    ( 9) 12:    ( 17) 13:
( 2) 21:    ( 10) 22:   ( 18) 23:
( 3) 31:    ( 11) 32    ( 19) 33:
( 4) 41:    ( 12) 42:   ( 20) 43:
( 5) 51:    ( 13) 52:   ( 21) 53:
( 6) 61:    ( 14) 62:   ( 22) 63:
( 7) 71:    ( 15) 72:   ( 23) 73:
( 8) 81:    ( 16) 82:   ( 24) 83:
```

以上のことを念頭において，配列に関するバッグについて次の事項を調べる。

1）配列名に対して，その配列宣言を落した場合。

2）配列宣言子において，次元の数を誤った場合。

3）配列宣言子において，添字の寸法を誤った場合。

宣言文はどちらかといえば実行文に対して従属的役割を果たす。したがって配列宣言自体の誤りもさることながら，多くの場合実行文との関係において誤りであるか否かの判定を下すことができる。つまり，上記の誤りは必ずしも宣言文の誤りではないかもしれない。時には実行文が誤っていることもある。しかしここでは実行文は正しく，必ずしも修正する必要のない事を想定して解説する。

配列宣言を落した場合

これは配列宣言子が非常に多いためにうっかりそれらの内の1つを落すことにより犯す誤りである。また時には配列であることを忘れてしまったり，他のデバッグをおこなっているうちに配列宣言の部分を（手違いのため）消してしまう等の比較的素朴な理由で犯すことも多い。

具体的にあらわれる現象としては大きく2つに分けることができる。まず第1に

(1) 配列要素が代入文の右辺にあらわれているにもかかわらず配列宣言がなされていない，場合である。コンパイルは主プログラム又は1つの副プログラムの範囲内でおこなわれ，副プログラム間の関係は見ない。しかも配列要素と関数は代入文の右辺では全く同じ形式で表現される。このために

(2) 配列は関数副プログラムとして処理される。

この結果リンクロードの時点で

(3) リンクロード時エラー・メッセージ

```
LLK17 I*****  UNRESOLVED  ENTRIES  Y
LLK38 I*****  UNRESOLVED  SUBPROGRAM
```

が出力される。

図2.4　配列名Yについての配列宣言を落とした場合のプログラム例。
リンクロード時にエラー・メッセージが出力される。

```
C
      DIMENSION A(10)
      DO 10 I=1, 10
      A(I)=FLOAT(I*(I+1))*Y(I)
   10 CONTINUE
            :
```

リンクロード時に出力されるエラー・メッセージ

```
LLK17 I*****UNRESOLVED  ENTRIES  Y
LLK38 I*****UNRESOLVED  SUBPROGRAM
```

図2.4にその具体例を示した。この誤りは配列宣言子がCOMMON文に現われるはずである場合にしばしば経験するものである。

第2に考えられるのは

(4) 配列要素が代入文の左辺または入出力文にあらわれているにもかかわらず，配列宣言がなされていない，

場合である。この時にはコンパイル時エラー・メッセージが

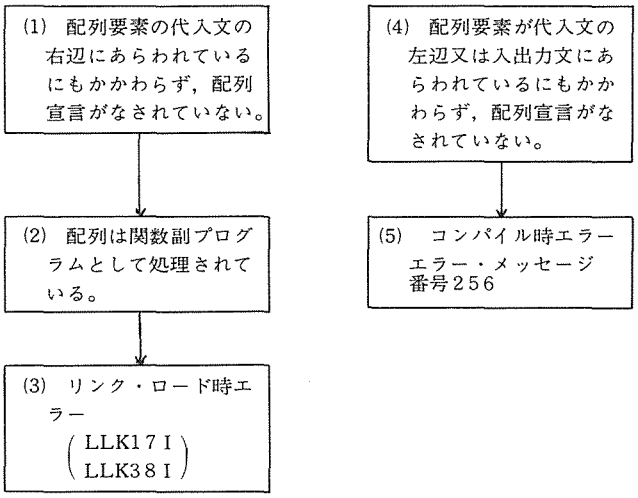
(5) コンパイル時エラー

256 VARIABLE NAME “name” FOLLOWED BY LEFT PAREN PARENTHESES

で出力される。

以上を図式化すれば図2.5の流れ図になる。

図2.5 配列宣言を落した場合に生ずる誤りの症状



配列宣言子の次元の数を誤った場合

誤りについて解説する前に配列の次元の数と配列要素の対応関係について触れておく。文法説明でも示されたように配列要素はすべて（次元の数に関係なく）1次元的に1列に並べて記憶されている。3頁の表に見るように配列要素の添字の値を見ることにより，配列の最初のものから数えて何番目にあるかを知ることができる。しかも，配列宣言において次元の高い配列はより次元の低い配列として実行文中で使う事が可能である。

図2.4には配列宣言子 IARRAY (8, 3) に対して実行文中で IARRAY (I, J) なる2次元配列と IARRAY (I) なる1次元配列の2例について使いわけた例を示した。2次元配列として使った場合については宣言文との関係が明解なので説明を省略する。

図2.4のプログラムでは，2次元配列において配列要素を1列に並べた時の各々の相対的位

置を知っておればそれはいつでも1次元配列でおきかえることが可能であることを示した。

このように配列宣言よりも低い次元の数で実行文中で配列要素を用いることが可能であるために，次の誤りを生ずることがある。

(6) 配列宣言において配列宣言子の次元の数を誤って多くとりすぎた。

もし正しい次元の数のところまでの寸法が誤っていなければ，上記のことは実行文へ直接影響しないかもしれない。ところがCOMMON文やEQUIVALENCE文等により変数や配列の間に結合がもたらされ，この点が重要な意味をもつ場合にはやはり(6)の誤りは重大である。

図2.6には配列IA (3, 3) とIB (3) をEQUIVALENCE文により配列IC (12) に結合させた時の例を示した。WRITE文による出力結果で見るように，配列A (3, 3)，B (3) は配列C (12) と同じ値をもっている。

図2.6 配列宣言文及びEQUIVALENCE文が使われたプログラムの例

```
C       MAIN PROGRAM
        DIMENSION IA(3,3),IB(3),IC(12),ID(10)
        EQUIVALENCE (IA(1,1),IC(1)),(IB(1),IC(10))
        DATA (ID(I),I=1,10)/41,42,43,44,45,46,47,48,49,50/
        DO 10 I=1,3
        IB(I)=I
        DO 10 J=1,3
   10   IA(I)=10*I+J
        DO 20 I=13,22
   20   IA(I)=I-9
        WRITE(6,100) (IA(I,J),I=1,3),J=1,3),(IB(I),I=1,3)
                     (IC(I),I=1,12),(ID(I),I=1,10)
  100   FORMAT(1H1,▽IA(I)=▽,8(I2,▽,▽),I2,▽.:△IB(I)=▽,2(I2,▽.▽)
     1  I2,▽.▽/1X,▽IC(I)=▽,9(I2,▽,▽),8X,2(I2,▽,▽),I2,▽.▽
     2          /1X,▽ID(I)=▽,9(I2,▽,▽),I2,▽.▽)
        STOP
        END
```

WRITE文による出力結果

```
IA(I)=11, 21, 31, 12, 22, 32, 33, 23, 33, :IB(I)=1,2,3.
IC(I)=11, 21, 31, 12, 22, 32, 13, 23, 33,        1,2,3.
ID(I)= 4,  5,  6,  7,  8,  9, 10, 11, 12, 13.
```

図2.6のプログラム例では，配列ID (10) は配列要素IB (3) 又はIC (12) の直後の番地に配列要素ID (1) がおかれ，以後にID (2)，ID (3)…ID (10) とつづいている。したがって

```
        DO 20 I=13,22
   20   IA(I)=I-9
```

を強引に実行すれば，これら右辺の数値はＩＤ(1)，ＩＤ(2)，…，ＩＤ(10)へ代入される。(このプログラム例は，フォートラン７００では必ずしも誤りとはならない。逆に特殊な手法ではあるが，有効な場合もある。)

図2.6がプログラム作成者の意図どうりに正しくつくられたプログラムであるとの仮定のもとで次の例を考える。

プログラム図2.6において配列宣言子が

ＩＡ(3, 3) → ＩＡ(3, 3, 3)

でおきかえられた例を図2.7に示した。配列宣言文以外は図2.6と全く同じである。ＷＲＩＴＥ文による出力結果は図中下部に示した。配列ＩＤ(10)の要素の値が初期のままである点が図2.6と異なる。

図2.7　プログラム図2.6において，配列宣言子を
A(3, 3) → A(3, 3, 3)
でおきかえた場合の例

```
⋮                          ⋮
DIMENSION        IA(3, 3, 3), IB(3), IC(12), ID(10)
⋮                          ⋮
STOP
END
```

ＷＲＩＴＥ文による出力結果

```
IA(I) =11, 21, 31, 12, 22, 32, 13, 23, 33. : IB(I) =1, 2, 3.
IC(I) =11, 21, 31, 12, 22, 32, 13, 23, 33,          1, 2, 3.
ID(I) =41, 42, 43, 44, 45, 46, 47, 48, 49, 50
```

図2.6と図2.7は，配列宣言子Ａ(3, 3)をＡ(3, 3, 3)に変えてしまったために，配列間の対応関係が変ってしまった。

図2.8では，これら両プログラムにおける対応関係の比較を示した。このように，配列宣言子の次元の誤りは，(ＩＢに対するように)全く表面にあらわれない場合もあるが，多くの場合ＩＤに対するように何らかの形で副作用をおよぼす。

さて，以上の症状をもとに整理してみれば次のようになる。まず，エラー(6)と密接な関係にある誤りとして次のものがある。

(7)　COMMON文やEQUIVALENCE文による配列の結合に誤りがある。

この誤りの結果として次のことがおこる。

(8)　配列要素の（数）値が正しく入っていない。

図2.8　プログラム図2.6及びプログラム図2.7における配列の対応関係とWRITE文直前の数値

プログラム図2.6

対応	数値
IB (1, 1) — IC (1)	11
IA (2, 2) — IC (2)	21
IA (3, 1) — IC (3)	31
IA (1, 2) — IC (4)	12
IA (2, 2) — IC (5)	22
IA (3, 2) — IC (6)	32
IA (1, 3) — IC (7)	13
IA (2, 3) — IC (8)	23
IA (3, 3) — IC (8)	33
IB (1) — IC (10)	1
IB (2) — IC (11)	2
IB (3) — IC (12)	3
ID (1)	4
ID (2)	5
ID (3)	6
⋮	⋮
ID (10)	13

プログラム図2.7

対応	数値
IA (1, 1, 1) — IC (1)	11
IA (2, 1, 1) — IC (2)	21
IA (3, 1, 1) — IC (3)	31
IA (1, 2, 1) — IC (4)	12
IA (2, 2, 1) — IC (5)	22
IA (3, 2, 1) — IC (6)	32
IA (1, 3, 1) — IC (7)	13
IA (2, 3, 1) — IC (8)	23
IA (3, 3, 1) — IC (9)	33
IA (1, 1, 2) — IC (10) — IB (1)	1
IA (2, 1, 2) — IC (11) — IB (2)	2
IA (3, 1, 2) — IC (12) — IB (3)	3
IA (1, 1, 3)	4
IA (2, 1, 3)	5
IA (3, 1, 3)	6
⋮	⋮
IA (3, 3, 3)	0
ID (1)	41
⋮	⋮
ID (10)	50

次に考えられるのは，誤り（6）とは逆に

(9)　配列宣言において配列の次元の数を誤って少なくとった，

場合である。この結果，実行文中での配列要素の次元の数が宣言文と一致しなくなり，

(10)　実行文中において，配列宣言より多い次元の数の配列を用いた，

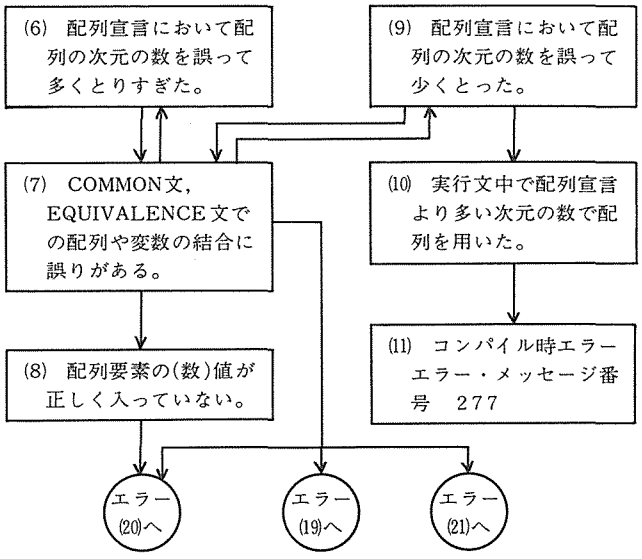
結果になる。但しこの場合には

(11)　コンパイル時エラー・メッセージが番号277で出力，

されるので発見は早い時点でなされる。

一方，エラー(9)にもかかわらず実行文中で用いられた配列要素の次元がなおかつ宣言文より少いこともあり得る。この時にはエラー(7)へつながる。以上の結果を流れ図で図式化したのが図2.9である。

図２.９　配列宣言において配列の次元の数を誤った場合に生ずる症状



配列宣言子の添字の寸法を誤った場合

寸法の誤りは記憶場所を占有する状況の面から見た場合には次元の数の誤りと同じである。したがってエラー(7)へ関連する場合もある。

一方，寸法の誤り方としては“大きすぎる”場合と“小さすぎる”場合が考えられる。

(12) 配列宣言において配列の寸法を誤って大きくとりすぎた，

場合には，もし

(13) 配列の寸法が $2^{18}-1$ を越えている，

ならば

(14) コンパイル時にエラー・メッセージが番号４１６で出力される。

大型計算機といえども記憶できる語数は有限である。したがって配列が大きすぎれば，全体として

(15) プログラムのサイズが大きすぎる。

ために

(16) リンクロード時又は実行時にエラー・メッセージが出力される。

```
LLK131***** CORE EXCEEDED ；リンクロード時
MRMOOI UEP INSUF MEMORY    ；実行開始時
```

配列の寸法を誤ってもそれがただちにエラー・メッセージに結びつくとは限らない。むしろ多くの場合，数値上の異常として検出される。

図2.10　配列宣言子 IA (4, 3) と IA (3, 3) の記憶される状態の比較

配列宣言子 IA (4, 3) の主記憶上での配列要素の順序の順序	配列宣言子 IA (3, 3) の主記憶上での配列要素の順序
IA (1, 1)	IA (1, 1)
IA (2, 1)	IA (2, 1)
IA (3, 1)	IA (3, 1)
IA (4, 1)	IA (1, 2)
IA (1, 2)	IA (2, 2)
IA (2, 2)	IA (3, 2)
IA (3, 2)	IA (1, 3)
IA (4, 2)	IA (2, 3)
IA (1, 3)	IA (3, 3)
IA (2, 3)	
⋮	

図2.10 には配列宣言子 IA (4, 3) 又は IA (3, 3) が主記憶上に記憶される場合の各配列要素の位置関係を示したものである。

図でもわかるように，配列宣言子が違った場合には，実行文で

K=IA (I, J)

を実行する時に，Kの値が違ってくる。すなわち，

(17) 配列宣言において配列宣言子の寸法を誤って小さくとりすぎた

場合においても，またエラー(12)の場合においてもいずれも

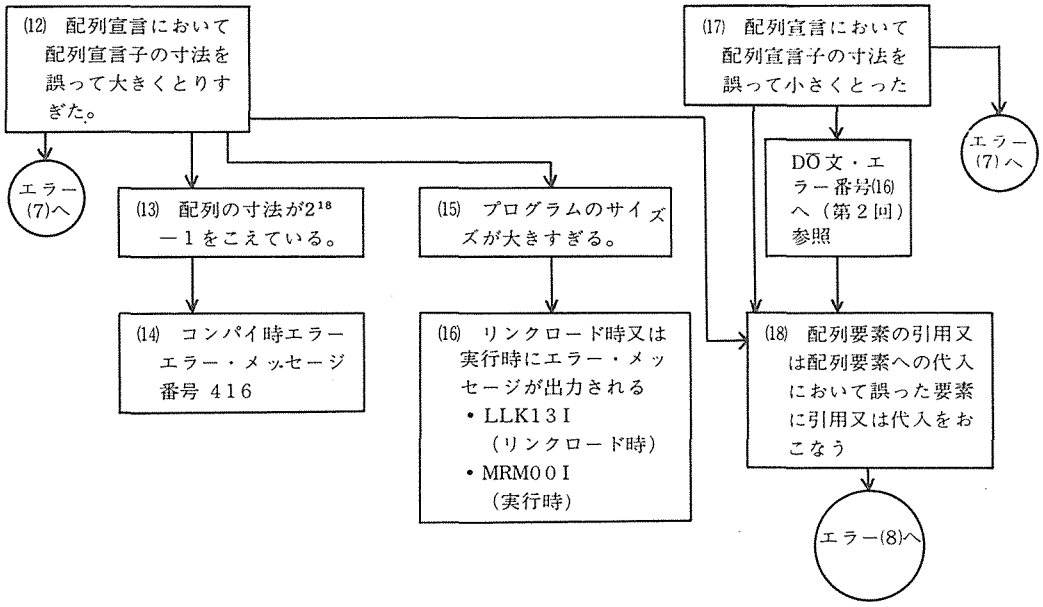
(18) 配列要素の引用又は配列要素への代入において，誤った配列要素の引用又は代入をおこなっている。

また，配列宣言子の次元の数と寸法は主記憶上では混みにして取扱われていることから，計算機としては区別できない。したがってエラー(17)はエラー(7)へ結びつくこともある。

また，配列はDOループの中でしばしば用いられるため，DO文エラー番号(16)（第2回参照）へ結びつくこともある。

以上を流れ図で示せば図2.11になる。

図2.11　配列宣言において添字の寸法を誤った場合に生ずる症状



3. COMMON文

COMMON文は，いくつかの副プログラムと主プログラムからなるプログラムに対して，これらの間に共通な変数や配列を定め，共通の場所に記憶しておくための宣言文である。このほかに配列宣言の機能ももつ。

3.1　COMMON文に関する文法

COMMON文の表現形式は次のように規定される。

(2) **COMMON文**　COMMON文(COMMON statement)は，つぎの形とする。

COMMON/x_1 / a_1 / … / x_n / a_n

ここで，a_1, …, a_nは，いずれも変数名や配列名や配列宣言子の並びとし，空ではなく，その中に仮引数を含まないものとする。x_1, …, x_nはいずれも英字名かまたは空とし，英字名の場合にはブロック名(block name)という。x_1が空の場合には最初の2個の斜線はなくてもよい。ブロック名は，これと同じ名前を持つ変数や配列がどこにあっても，たとえ同じプログラム単位内にあっても，それとは何の関係もない。

例：COMMON文

```
COMMON/G/H, I, B (5, 13) //K
COMMON  L (10) /G/C
```

この例で，H, I, B, Cはこの順で共通ブロックGの中にある。K, Lはこの順で無名共通ブロックの中にある。

このように定められたCOMMON文において，斜線及び英字名はそれぞれの位置により次のように意味づけられる。

COMMON文において，ブロック名xの直後の斜線からつぎの斜線までの間の，また直後の斜線のあとに斜線がない場合には，その文の終りまでの，並びの要素は共通ブロック(common block) xの中にあると宣言されるものとする。COMMON文の最初からブロッ

ク名が現われるまでの，またはブロック名が現われない場合には，文の中のすべての，並びの要素は無名共通ブロック(blank commonまたはunlabeled common) の中にあると宣言されるものとする。さらに，2個の斜線の間にブロック名がないものが現われた場合には，その斜線に続く並びの要素も無名共通ブロックの中にあると宣言されるものとする。無名共通ブロックに対して，ブロック名を持つ共通ブロックを名前付共通ブロック(labeled common block) という。

特定の共通ブロック名が一つのCOMMON文あるいは一つのプログラム単位内に何度現われてもよい。処理系は，それらに続く並びの要素のすべてを，現われる順序につなぎ合わせて，一つの共通ブロックにするものとする。無名共通ブロックについても同様とする。共通ブロックは，変数や配列要素の列で構成され，配列の中の配列要素の並び方は，配列要素関数によるものとする。

プログラム単位内の共通ブロックの大きさは，COMMON文およびEQUIVALENCE 文で持ち込まれた要素が必要とする記憶単位全体の個数とする。一つの実行可能プログラム内のいくつかのプログラム単位に同一のブロック名を持つ名前付共通ブロックがある場合には，それらのブロックの大きさは同じでなければならない。無名共通ブロックの大きさは，プログラム単位ごとに異なってもよい。

共通ブロックはいくつかの副プログラムに対して配列及び変数の共通の記憶場所を指定するものである。したがって副プログラム間で相互につじつまの合う共通ブロックの表現（宣言）をおこなわなければならない。それらについては次のように定められている。

共通ブロック間の対応：いくつかのプログラム単位に同一のブロック名をもつ名前付共通ブロックがある場合には，それらは互いに対応している(correspond) といい，同一の記憶場所を共有する。また，いくつかのプログラム単位に無名共通ブロックがある場合には，それらは互に対応しているといい，それらの大きさが同じならば同一の記憶場所を共有し，大きさが違うならば，小さい方は大きい方の記憶場所の一部を共有する。

共通ブロック内では整数型や実数型や論理型の変数や配列要素は一つの記憶単位を占め，倍精度実数型や複素数型の変数や配列要素は連続した二つの記憶単位を占め，それらは共通ブロック内での並び方に従って記憶場所に割当てられるものとする。対応する共通ブロックでは，それらの先頭の要素の占める記憶場所は同一とする。

同一の記憶場所を共有する二つ以上の変数や配列要素は，互いに結合されていると

例：共通ブロック間の対応

使用例：

```
SUBROUTINE SUB1
COMMON A,B,C/BL/D, E
  ⋮
END
SUBROUTINE SUB2
COMMON X,Y(3)/BL/Z(2)
  ⋮
END
```

上の例で，A, B, C, D, E, X, Y, Z がすべて実数型であれば，AとX, BとY(1)，CとY(2)，DとZ(1)，EとZ(2)は互いに結合されている。

上の例で，Xだけが複素数型で他は実数型であれば，A, BとX, CとY(1) が結合されている。

ここで，AはXの実数部と，BはXの虚数部と，同じ記憶場所を占める。

上の例でDが倍精度実数型で，E, Z が実数型であれば，副プログラムSUB1で

いう。互いに結合されている二つの要素において，両方の型が同じならば，一方が確定のときは他方も確定とし，一方が不定のときは他方も不定とする。また型が違えば一方が確定のときは他方は不定とする。	の共通ブロックBLの大きさは3記憶単位で，SUB2でのそれは2記憶単位であるから誤りである。

COMMON文は，それのもつ機能の点でもまた宣言された内容の点でも共に広範囲に影響をおよぼすので，デバッグにおいては広い範囲にわたって調べなければならない。上記文法については，宣言の形式よりもその効果の大きさに注目するのが良いと思われる。

3.2 COMMON文のコンパイル時エラー・メッセージ

3.2.1 WARNING

002	WARNING: ADJUSTABLE DIMENSION "name" IN COMMON ○整合寸法を示す整数型の変数名がCOMMON文中に含まれている。 （プログラム　コード　-00_8）

（例）

内部文番号	フォートラン文
0001	SUBROUTINE SUB (A)
0002	COMMON B (10) , I
0003	DIMENSION A (I)

内部文番号0003における配列Aの整合寸法がCOMMON文中に含まれている。整合寸法を示す変数は仮引数にする事が原則であるためWARNINGになった。

3.2.2 FATAL ERROR

431	ILLEGAL COMMON STATEMENT SYNTAX ○COMMON文中に文法違反がある。適当な仮定のもとにコンパイルされるが，このCOMMON文を含むプログラム単位はGO ファイル上に出力されない。

（例）

内部文番号	フォートラン文
0002	COMMON
0003	COMMON/A/B,C/3P/Q,R

第1行目はCOMMON文中に変数及び配列を含まない。この誤りの文はないものとしてコンパイルはつづけられる。

第2行目はLABELED COMMON /3PQ, ,Rの名前が誤りである。名前付き共通ブロックの名前は英字かははじめなければならない。この文はCOMMONら

/A/B,Cとしてコンパイルされる。

432	"name" IS A DUPLICATE COMMON ELEMENT OR IS DECLARED IN MORE THAN ONE COMMON BLOCK ○COMMON文において同一の要素が2つ以上のブロックに含まれているか，あるいは同一のブロックに2回以上にわたって含まれている。これらのブロックはないものとしてコンパイルされるが，このCOMMON文を含むプログラム単位はGO ファイル上に出力されない。

(例)　内部文番号　　フォートラン文

0002　　COMMON U,V,U

COMMON文中に同じ英字名Uが2回あらわれているため誤りである。同一プログラム単位内で，1つの英字名についての宣言は1度でよい。

433	MORE THAN 47 LABELED COMMON BLOCKS ARE DECLARED ○名前付き共通ブロックの個数が47を越えた。48個目以降に現われた名前付き共通ブロックはないものとしてコンパイルされるが，このプログラム単位はGO ファイル上に出力されない。

(例)　COMMON/COM1/A1,B1/COM2/A2,B2/…

⋮

COMMON/COM46/A46,B46/COM47/A47,B47/COM48/A48,B48/

LABELED COMMON BLOCKが48個あるため誤りである。

434	TOO LARGE COMMON BLOCK "name" ○無名共通ブロックか1つの名前つき共通ブロックの大きさが$2^{18}-1$語を越えた。$2^{18}-1$語までの要素には主記憶装置が割付けられてコンパイルされるが，この文を含むプログラム単位はGO ファイル上に出力されない。EQUIVALENCE文によって持ち込まれた要素によってこのエラーが検出された場合も含む。無名共通ブロックの場合は，"name" の部分は"BLANK—CM" になる。 (プログラム　コード　— 10_8)

このエラーは，計算機システムとして予め共通ブロックだけのために予約された記憶領域があり，この領域の大きさの制限を越えたための誤りである。

537	LABELED COMMON BLOCK INITIALIZED IN NON−BLOCK DATA SUBPROGRAM ○初期値設定副プログラム以外で名前付共通ブロック内の要素に初期値が割付けられている。初期値の割付けられた変数等の含まれる並びとそれに対応する定数等の並びはないものとしてコンパイルされるが，このプログラム単位はGO ファイル上に出力されない。 （プログラム　コード　-10_8）

（例）

内部文番号	フォートラン文
0001	SUBROUTINE　SUB
0002	COMMON　/COM1/A,B,C/
0003	DATA　A,B,C/1.2,2.0,3.0/

初期値設定副プログラム以外で名前付共通ブロック内の要素に初期値の割付をおこなってはいけない。もし初期値の割付けが必要ならば代入文ですると よい。

538	“name” IN BLANK COMMON INITIALIZED ○DATA文あるいは型宣言文により，無名共通ブロック内の要素に初期値を設定している。その要素には初期値は入れられないものとしてコンパイルされるが，そのプログラム単位はGO ファイル上に出力されない。 （プログラム　コード　-10_8）

（例）

内部文番号	フォートラン文
0002	COMMON　A,B,C
0003	DATA　A,B,C/1.0,2.0,3.0/

無名共通ブロック内の変数A,B,Cに DATA文で初期値設定をやってはいけない。もし初期値設定が必要ならば代入文でおこなう。

576	DUMMY ARGUMENT “name” IS IN COMMON ○仮引数として使用されている英字名が，COMMON文中にも現われている。そのままコンパイルされるが，このCOMMON文を含むプログラム単位はGO ファイル上に出力されない。 （プログラム　コード　-10_8）

（例）

内部文番号	フォートラン文
0001	SUBROUTINE　SUB1 (C,I)
0002	COMMON　A,B (10) ,I
	⋮

英字名Ｉは仮引数であると同時にCOMMON文中にもあらわれたため誤りである。COMMON文又は仮引数のいずれか１方のみとする。

3.3　リンクロード時，実行時におけるCOMMON文の誤り

COMMON文は配列や変数の記憶場所をいくつかの副プログラム間で共有することを宣言する文である。したがって単純な誤りはコンパイル時に発見されるが，プログラムの意味にかかわる重大な影響をもつ誤りは概ねリンクロード時以後に発見される。

無名共通ブロックと名前付き共通ブロックのいずれにおいても，各々の副プログラム間で変数や配列の記憶場所が相互に矛盾することなく宣言されていなければならない。また変数や配列の値については，それらの記憶されている場所（主記憶上の番地）が重要であり，英字名はプログラム単位内でのみ意味をもつ副次的役割をもつのみである。但し，型はすべて統一されていなければならない。これらは，COMMON文の欠落やCOMMON文中の変数名・配列名の欠落のみでは文法の面から見て必ずしも誤りとは断定できないことになるため，デバッグの際の困難さの重大な一因になる。

以上のことからCOMMON文に関して誤りを犯すとしたら，次の３つの場合が考えられる。

(19)　COMMON文中である変数名又は配列名を落した。

(20)　COMMON文中で配列宣言子の次元の数又は寸法をまちがえた。

(21)　１つのプログラム単位内でCOMMON文全体を落した。

すでに述べたように，上記の(19)～(21)がプログラムとして誤りであるか否かは１概に判定できず，それはプログラム作成者の判断によるしかない。つまり，作成意図通りに処理されたことをもって正しいと言う以外にないことに注意していただきたい。

以下の具体例を見ることによりCOMMON文の振舞いを示す。但し，以下でバッグ（誤り）と言う表現は，常に正しいといっている部分と対にして見ることが大切である。

COMMON文における変数名又は配列名の欠落

共通ブロックは無名と名前付きのいずれも，その共通ブロックを参照した主プログラム又は副プログラムの内で最初に主記憶にロードされるものによってサイブが定められる。一度サイズの定められた共通ブロックは，次に参照される時にはその大きさを越えてはならない。逆に，一度定められた共通ブロックが別の副プログラムで参照される時に，すでに定められた大きさより小さければ文法上は誤りとならない。

図３.１は主プログラムに対して副プログラムの共通ブロックが小さくとられた例である。この場合には，エラー・メッセージの出力はなかった。但し変数名と配列名において，主プログラムと副プログラム間に次の対応関係がある。

図3.1　サイズの異なる共通ブロックの例
（エラー・メッセージの出力なし）

```
C   MAIN
    COMMON /COM1/A,B,C(10),D(10)
    CALL SUB
    STOP
    END
C
    SUBROUTINE SUB
    COMMON /COM1/A,B,D(10)
    RETURN
    END
```

エラー・メッセージなし

図3.2　サイズの誤なる共通ブロックの例
（エラー・メッセージが出力された）

```
C   MAIN
    COMMON /COM1/A.B,D(10)
    CALL SUB
    STOP
    END
C
    SUBROUTINE SUB
    COMMON/COM1/A,B,C(10),D(10)
    RETURN
    END
```

リンクロード時エラー・メッセージ
LLK67I

主プログラム側 共通ブロック		副プログラム側 共通ブロック
A	⟷	A
B	⟷	B
C(10)	⟷	D(10)
D(10)	⟷	なし

図3.2は主プログラムに対して副プログラムの共通ブロックが大きくとられた場合の例である。この場合には，主プログラムの共通ブロックのサイズが優先され（すなわち先にロードされるため）副プログラム側の誤りとなる。この時，リンクロード時に

LLK67I****LABELD COMMON SIZE ERROR　ICOM1

が出力される。

図3.1のプログラムにおいては，共通ブロックを通して共有される変数名又は配列名の値が，プログラム単位間で正しく対応関係が守られているか否かの点で，プログラムの意味上の問題点を含む。

共通ブロックによって変数又は配列要素の値が共有されているようすを見るために，図3.3にテスト用プログラムの例を示した。副プログラムSUBへ主プログラムから値が移るようすは，WRITE文による出力結果を見ることによりわかる。

さて，共通ブロックによって値が共有されるようすをもっとくわしく見るために図3.3において副プログラムのCOMMON文を変えてみる。

① 変数Bを落した場合。

COMMON/COM1/A,C(10),D(10)

図3.3　共通ブロック内の変数と配列要素の値が副プログラム間で共有されていることを示すためのテスト用プログラム

```
C          MAIN
           COMMON/COM1/A,B,C(10), D(10)
           A=1.0
           B=2.0
           DO 10 I=1,10
           C(I)=FLOAT(I)
  10       D(I)=FLOAT(I)+10.0
C
           CALL SUB
           STOP
           END
C
           SUBROUTINE SUB
           COMMON/COM1/A,B,C(10), D(10)
           WRITE(6,100) A,B, (C(I), I=1,10), (D(I), I=1, 10)
  100      FORMAT(1H3,▼OUT−PUT IN SUB▼/
       1          ▼A=▼,F3.1,▼,△△B=▼,F3.1,▼.▼/
       2          ▼C(I)=▼,9(F4.1,▼,▼), F4.1,▼.▼/
       3          ▼D(I)=▼,9(F4.1,▼,▼), F4.1,▼.▼)
           RETURN
           END
```

WRITE文による出力結果

```
OUT−PUT IN SUB
A=1.0,   B=2.0.
C(I)= 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0,10.0.
D(I)=11.0,12.0,13.0,14.0,15.0,16.0,17.0,18.0,19.0,20.0.
```

もし実行文において変数Bがあらわれていれば

```
025 WARNING: UNDEFINED VARIABLE B
```

が出力される。またWRITE文による出力結果は図3.4のとうりである。

図3.4　プログラム図3.3において副プログラムSUBのCOMMON文を
COMMON/COM1/A,C(10), D(10)
としたときの出力結果

```
OUT−PUT IN SUB
A=1.0,   B=0.0
C(I)= 2.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0.
D(I)=10.0,11.0,12.0,13.0,14.0,15.0,16.0,17.0,18.0,19.0.
```

② 配列C(10)を落した場合

```
COMMON/COM1/A,B,D(10)
DIMENSION C(10)
```

DIMENSION文において配列C(10)を定義した理由は，テスト用のWRITE文の所でコンパイル時にエラー・メッセージ

178 SYNTAX ERROR IN I/O LIST

256 VARIABLE NAME "C" FOLLOWED BY LEFT PARENTHESES

にひっかからないためである。もし実行文中で配列名C(10)が1度も現われなかったならば，エラー・メッセージの出力はない。

さて，このように変えられたCOMMON文に対して，WRITE文による出力結果は図3.5のとおりである。

図3.5 プログラム図3.3において副プログラムSUBのCOMMON文を
COMMON/COM1/A,B,D(10)
DIMENSION C(10)
にした時のWRITE文による出力結果

```
OUT-PUT IN SUB
A=1.0,   B=2.0
C(I) = 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,  0.0.
D(I) = 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0.
```

上記，場合①（図3.4）と場合②（図3.5）を図3.3に示されたプログラムの出力結果と比較することにより次のような結論を得る。

すなわち，

(19) COMMON文中で，ある変数名又は配列名が落ちた，

場合には，もしそれらが実行文で使われていれば，コンパイル時に

(22) 変数名に対して

025 WARNING: UNDEFINED VARIABLE "name"

が出力される。また

(23) 配列要素名に対してFATAL ERRORとなり

256 VARIABLE NAME "name" FOLLOWED BY LEFT PARENTHESES

が出力される。

一方，COMMON文中から落ちた変数名又は配列名が実行文中で使われることもない場合には，エラー・メッセージはない。しかしプログラムの意味の上から誤りであった場合には，数値に異常を生ずる。COMMON文中で変数名又は配列名が落ちた場合には，英字名には無関係にCOMMON文中で左づめに，共通ブロック領域に配置された値がそれぞれの位置に応じて割りあてられる。

すなわち

(24) COMMON文中で配列又は変数の値が相互に対応しない英字名の所にわりあてられている。

この結果，誤りの症状として表面にあらわれるのは，

(25) COMMON文中に有る（又は有るはずの）配列又は変数の値が異常である。

(26) COMMON文中にあるべき配列又は変数の値がゼロである。

(27) 副プログラム引用の際に正しい値の受け渡しができない。

(28) COMMON文中に有る（又は有るはずの）配列又は変数の値が正しく記憶されていない。

等のさまざまな形態をとる。

最初にも述べたように，COMMON領域の広さは最初にロードされるプログラム単位の中のCOMMON文により決められる。したがってCOMMON文中の変数名又は配列名の落ちたプログラム単位が先きにロードされれば，後にロードされたプログラム単位において矛盾が生ずる。このために

(29) リンクロード時にエラー・メッセージ

LLK67I*****

が出力される。また，COMMON文では配列宣言の機能も備えているので，COMMON文中の配列名の欠落がエラー(1)やエラー(4)につながる場合もある。

COMMON文中の配列宣言子の添字の誤り

共通ブロックとして取られた領域内には，配列と変数の区別はなく，単に値が1列に記憶されていて，それをいくつかの副プログラムで共有しているにすぎない。したがって，配列又は変数として共通ブロックからそれらの値を引用する際には，前から数えて何番目の値を引用するかが重要になる。このために前にも見たように，変数名又は配列名が欠落した時には，落ちた部分は無視してすべて左づめにしてそれらの値が対応させられる。このことは，特に配列名において配列宣言子の添字を誤った場合にも全く同じである。ここで次の例を調べてみよう。

③ 配列宣言子C(10)を誤ってC(5)とした場合

図3.3のプログラムにおいてCOMMON文が次の文でおきかえられる

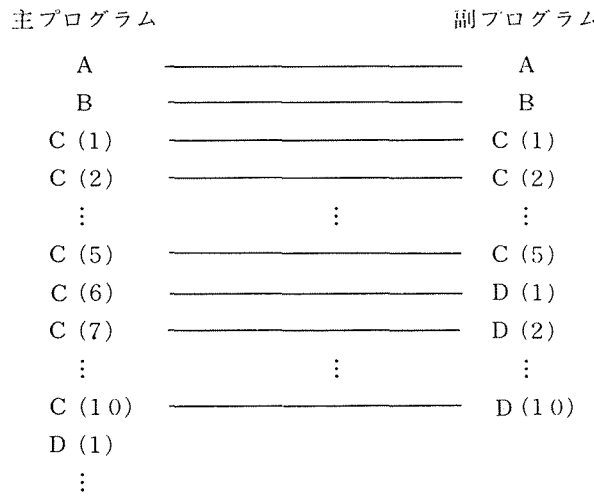
COMMON/COM1/A,B,C(5),D(10)/

この場合には，実行文中で配列要素C(I)に5以上の定数添字が使われていなければ，エラー・メッセージの出力はない。（この例題では副プログラム文中のWRITE文において配列C(5)の出力を(C(I), I=1,5)とした。）

図3.6 プログラム図3.3において副プログラムSUBのCOMMON文を
COMMON/COM1/A,B,C(5), D(10)
とした場合のWRITE文による出力結果

```
OUT-PUT IN SUB
A=1.0, B=2.0.
C(I) = 1.0, 2.0, 3.0, 4.0, 5.0.
D(I) = 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0.
```

主プログラムと副プログラムにおけるCOMMON文中の変数及び配列の値の対応関係は次のようになっている。



このように

(20) COMMON文中の配列宣言子添字を誤った

場合であっても，見掛け上エラー(19)と全く同じ症状を示す。このためにエラー(20)はエラー(24)及び(29)へつながる。

COMMON文全体の欠落

1つの副プログラムにおいてCOMMON文が全く無い場合には，他の副プログラムとの間で変数や配列の授受はおこなわれない。また配列宣言子を含むCOMMON文が欠落した場合には配列に関するエラーがおこる。

すなわち

(21) 1つのプログラム単位内でCOMMON文全体を欠落した

場合には，エラー(26)へつながる。また，COMMON文中にあるべきはずの変数や配列が実行文中にあらわれれば，エラー(22)又はエラー(23)につながる。

図3.7 プログラム図3.3において副プログラムSUBのCOMMON文を完全に落してしまった場合のWRITE文による出力結果

```
OUT-PUT  IN  SUB
A=0.0,  B=0. 0
C(I)=0.0,0.0,0.0,0.0, 0.0,0.0,0.0,0.0,0.0,0.0.
D(I)=0.0, 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
```

参考のため，図3.3において副プログラム中のCOMMON文全体を落した場合の例を図3.7に示した。

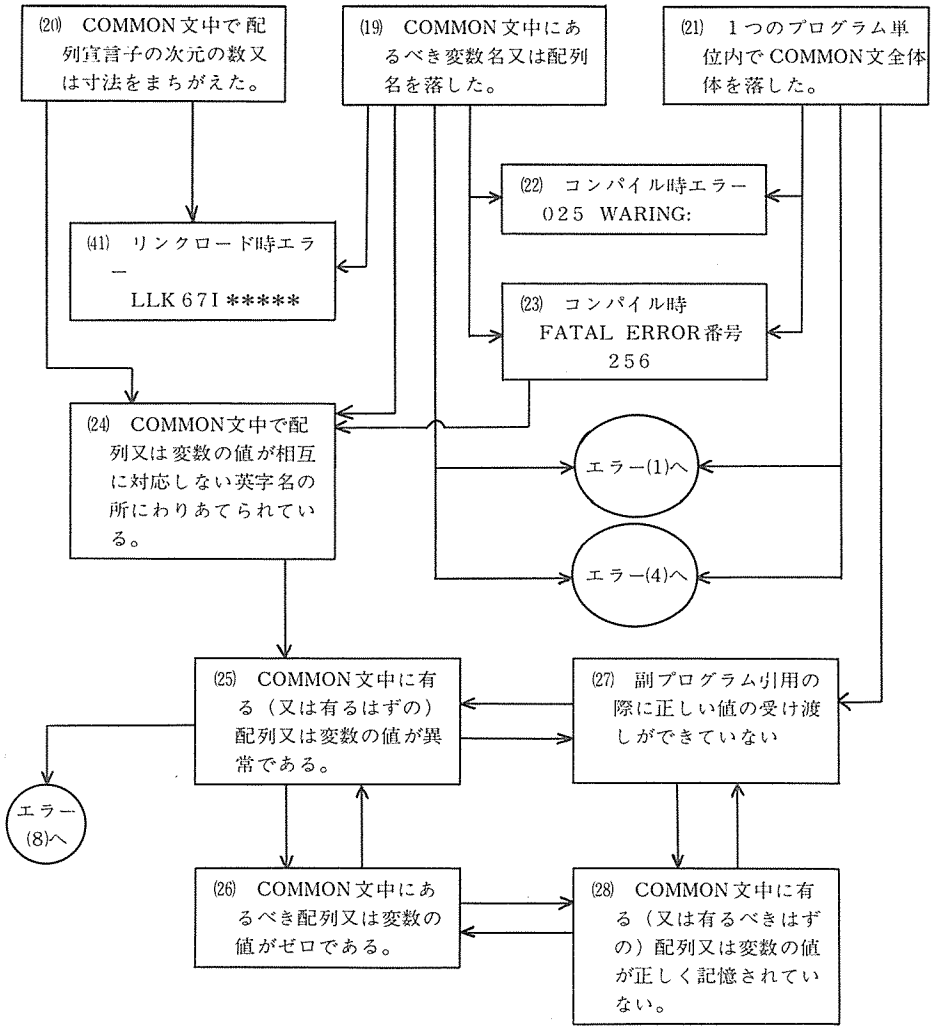
但し，配列名についてはFATAL ERRORを避けるために

DIMENSION C(10), D(10)

を追加した。

以上COMMON文についての誤りの状態の因果関係をまとめたものが図3.8である。

図3.8 COMMON文における誤りの症状



4. EQUIVALENCE文

4.1 EQUIVALENCE文に関する文法

EQUIVALENCE文は違った英字名の変数又は配列に対して，共通の記憶場所を割当てるためのものである。特に配列に関してはEQUIVALENCEで関連ずけられた状態が直接に配列要素の主記憶上での列び方に影響するので充分な注意が必要である。

EQUIVALENCE文は次の形式で表現される。

EQUIVALENCE文　EQUIVALENCE文(EQUIVALENCE statement)は，つぎの形とする。

EQUIVALENCE (k_1), (k_2), …, (k_n)

ここで，k_1, k_2, …, k_nは，いずれもつぎの形とする。

a_1, a_2, …, a_m

ここで，$a_1, a_2, \cdots a_m$は，いずれも変数名か配列要素名で仮引数は許されない。配列要素名の添字として許されるのは定数だけとし，mは2より大きいか等しくなければならない。配列要素名の添字式の数は，配列宣言子の次元の数に等しいかあるいは1でなければならない。後者の場合，添字式の値を添字の値とみなして配列要素を識別する。

このように記述されたEQUIVALENCE文は具体的に変数又は配列の間に次の位置関係を定める。

EQUIVALENCE文は，記憶場所を二つ以上の要素で共有させるのに使う。処理系によって並びの中の各要素は，同じ記憶場所に割当てられるものとする。二つの記憶単位を占める要素が一つの記憶単位を占める要素に対応した場合には，後者は前者の記憶場所のうち最初の記憶単位を共有する。

例：EQUIVALENCE文

使用例：

```
DIMENSION B(5), C(2,3)
EQUIVALENCE (A,B(2),C(5))
```

この例で，A，B，Cがすべて実数型のときはAとB(2)とC(5)とが同じ記憶場所を占める。したがってB(3)とC(2,3)も同じ記憶場所を占める。図示するとつぎのようになる。

```
            C(1,1)
            C(2,1)
            C(1,2)
     B(1)   C(2,2)
A—B(2)—C(1,3)…C(5)
     B(3)   C(2,3)
     B(4)
     B(5)
```

また，使用例のEQUIVALENCEの代わりに

```
EQUIVALENCE (A, B(2)), (B(3), C(2,3))
```

としても同じ結果になる。

使用例でさらに

```
COMMON /X/B
```

となっていてB(1)が共通ブロックXの最初の要素であるときは，上のEQUIVALENCE文によって，共通ブロックXが最初の要素の前に伸びることになるから誤りである。

以上のように定められたEQUIVALENCE文については，さらに次の補足的事項がある。

備考　EQUIVALENCE文は，二つ以上の要素を数学的に等しくするのに使用すべきでない。

配列における配列要素の並び方は，その添字の値（7.2.2参照）によって決められているので，EQUIVALENCE文の並びの要素の中に配列要素名が現われたときには，その配列の他の配列要素の占める記憶場所も自動的に決められているものとする。

備考　そのためEQUIVALENCE文で直接書かれた要素間の対応のほかに間接的な対応が引起こされることがある。EQUIVALENCE文による間接的な対応が共通ブロックの大きさを長くする場合，COMMON文で直接書かれた共通ブロックの最後の要素のあとに伸びることしか許されない。

EQUIVALENCE文を使ったために二つの変数や配列要素が記憶場所を共有している場所には，これらの変数や配列の名前が両方とも同一プログラム単位内にあるCOMMON文の中に現われてはならない。

直接的にせよ，一つの記憶単位に同一の配列の二つ以上の配列要素を割当ててはならない。

4.2　EQUIVALENCE文に関するコンパイル時エラー・メッセージ

4.2.1　FATAL ERROR

281	CONSTANT SUBSCRIPT EXCEEDS RANGE OF ARRAY ‘“name” IN EQUIVALENCE ○EQUIVALENCE文中において，配列要素の添字の値が配列宣言子により定められた大きさを越えている。この配列要素を含む並びはないものとしてコンパイルされるが，このプログラム単位はGOファイル上に出力されない。 （プログラム　コード　-10_8）

（例）

内部文番号	フォートラン文
0005	DIMENSION A(10), B(10)
0006	EQUIVALENCE (A(1), B(11))

配列宣言においてB(10)であるにもかかわらずEQUIVALENCE文においてB(11)が与えられている。

473	UNDEFINED ARRAY REFERENCE “name” ○EQUIVALENCE文において，配列宣言されていない配列要素名が現われた。その配列要素名を含むグループはないものとしてコンパイルされるが，この文を含むプログラム単位はGOファイル上に出力されない。 （プログラム　コード　-10_8）

（例）

内部文番号	フォートラン文
0005	DIMENSION A(10)
0006	EQUIVALENCE (A(1), B(2))
⋮	⋮

配列宣言においてB（10）であるにもかかわらずEQUIVALENCE文においてられている。

473	UNDEFINED ARRAY REFERENCE “name” ○EQUIVALENCE文において，配列宣言されていない配列要素名が現われた。その配列要素名を含むグループはないものとしてコンパイルされるが，この文を含むプログラム単位はGO ファイル上に出力されない。 （プログラム コード -10_8）

（例）

内部文番号	フォートラン文
0005	DIMENSION A（10）
0006	EQUIVALENCE （A（1），B（2））
⋮	⋮

配列Bは配列宣言されていないにもかかわらずEQUIVALENCE文中にあらわれた。

451	ILLEGAL EQUIVALENCE STATEMENT SYNTAX ○EQUIVALENCE文に文法違反がある。適当な仮定のもとにコンパイルされるが，このEQUIVALENCE文を含むプログラム単位はGO ファイル上に出力されない。

このエラー・メッセージは文法に違反した形式の表現をおこなった場合に出力される。誤りのようすによりコンパイル時の処置は違ってくる。例えば，次のような処置がなされる。

(1) EQUIVALENCE（A（1），B（5））， （C（1），D（J））は
EQUIVALENCE（A（1），B（5））と仮定される。

(2) EQUIVALENCE（A（1）/B（5））（C（10），D（30））は
EQUIVALENCE（C（10），D（30））と仮定される。

452	PROCEDURE NAME “name” USED IN EQUIVALENCE ○手続き名とNAMELIST名で使用された名前がEQUIVALENCE文でも使用されている。この名前を含む並びがないものとしてコンパイルされるが，このEQUIVALENCE文を含むプログラム単位はGO ファイル上に出力されない。

(例)

```
内部文番号          フォートラン文
0001                SUBROUTINE  SUB0 (X,Y)
0002                EQUIVALENCE  (SUB0,X)
                        ⋮
```

サブルーチン名 SUB0 が変数Xと EQUIVALENCEで結合されている。文法違反である。

453	DUMMY ARGUMENT“name” USED IN EQUIVALENCE ○EQUIVALENCE文中において仮引数名が使用されている。その名前の含まれる並びがないものとしてコンパイルされるが，このEQUIVALENCE文の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。

(例)

```
内部文番号          フォートラン文
0001                SUBROUTINE  SUB1 (I)
                        ⋮
0004                EQUIVALENCE  (I, K)
                        ⋮
```

仮引数 I がEQUIVALENCE文中にある。仮引数はEQUIVALENCE文にあらわれてはいけない。

454	CONFLICTING EQUIVALENCE DECLARATION FOR “name” ○EQUIVALENCE文に矛盾した定義がある。その名前を含む並びはないものとしてコンパイルされるが，その文を含むプログラム単位はGO ファイル上に出力されない。

(例)

```
内部文番号          フォートラン文
0002                DIMENSION  A (20)
0003                EQUIVALENCE  (A (1), A (15))
```

DIMENSI ON文では配列Aが

A (1), A (2), …, A (20)

の順で主記憶上に記憶されることを示している。EQUIVALENCE文はA (10) とA (15) を主記憶の同じ場所に記憶することを示している。これらは矛盾である。

455	EQUI VALENCE ALTERS THE BASE OF COMMON ○EQUIVALENCE文により，共通ブロックがその最初の要素の前に向って拡張されている。この誤りを引き起した並びがないものとしてコンパイルされるが，このEQUIVALENCE文を含むプログラム単位はGO ファイル上に出力されない。

（例）

内部文番号	フォートラン文
0005	REAL A (10)
0006	COMMON/MYCOM/B,C
0007	EQUIVALENCE (A (10), C)

上記宣言文による配列A及び変数B，Cの記憶場所は次図のとおりである。しかしCOMMON領域は変数Bより上にはとられていない。このため誤りとなる。（EQUIVALENCE文についての文法説明を参照。）

A (1)
A (2)
A (3)
⋮
A (8)
B—A (9)
C—A (10)

4.3 リンクロード時，実行時におけるEQUIVALENCE文の誤り

EQUIVALENCE文は変数名又は配列名について共通の記憶場所を割当てる。また，COMMON文とは違って1つの副プログラム単位内でのみその効果は有効である。しかし，EQUIVALENCE文中にあらわれた変数名又は配列名がCOMMON文中にも現れる場合には，副作用として記憶場所が前後に伸びる効果があるために，他の副プログラムへその効果が派生する場合もあり得る。その他，EQUIVALENCE文では 配列要素に対して記憶場所の対応関係を誤ることから，大きなバッグになることが多い。以下ではEQUIVALENCE文により結合された変数や配列要素の記憶場所についての相対的位置関係を調べる。

図4.1はEQUIVALENCE文を用いたプログラムの例である。

配列A (3,3) ,B (3) と配列C (12) はEQUIVALENCE文により図4.2の如くに結合されている。これらが正しく配置されている事はWRITE文の出力結果を見ればわかる。配列A (3,3) は配列C (1) ～C (9) が対応しており，配列B (3) にはC (10) ～C (12) が対応している。図4.1のプログラムにおいて誤りとなる場合として2つの場合が考えられる。1つはEQUIVALENCE宣言を落した場合であり，もう1つはEQUIVALENCE 宣言中の配列の添字を誤ったために対応関係が壊れる場合である。

図4.1　EQUIVALENCE文を用いたプログラムの例

```
内部文番号                              フォートラン文
0001              SUBROUTINE SUB
0002              DIMENSION IA(3,3), IB(3), IC(12)
0003              EQUIVALENCE (IA(1,1), IC(1)), (IB(1), IC(12))
      C
0004              DO 10 I=1,3
0005              IB(I)=I
0006              DO 10 J=1,3
0007           10 IA(I,J)=10*I+J
      C
0008              WRITE(6,100) ((IA(I,J), J=1,3) ,I=1,3), (IB(I), I=1,3),
                 1 (IC(I), I=1, 12)
0009          100 FORMAT (1H2,3X,▼IA(I)=▼,8(I2,▼,▼), I2,▼.:IB(I)=▼,2(I2,
                 1 ▼,▼) , I2,▼.▼/
0010              2          3X, IC(I)=  ,9(I2,▼,▼), 7X,2(I2, ▼,▼), I2,▼.▼)
0011              RETURN
0012              END
```

WRITE文による出力結果

```
IA(I)=11,12,13,21,22,23,31,32,33.:IB(I)=1,2,3.
IC(I)=11,21,31,12,22,32,13,23,33,          1,2,3.
```

EQUIVALENCE宣言を落した場合

EQUIVALENCE宣言によって結合された変数又は配列においてそれらの内の1つに値が与えられれば，他の結合された変数名又は配列名にも同時に同じ値が与えられる。これらのことを逆に考えれば，次のようになる。

(29)　変数又は配列間の結合に使われたQUIVALENCE宣言が落ちた，

場合には，その変数名又は配列名が代入文の左辺のみにあらわれるか又は右辺のみにあらわれるか，あるいは両辺にあらわれるがによって次の症状を示す。

(30)　変数又は配列のEQUIVALENCE宣言が落ちていて，しかもそれらが代入文の右辺又は入出力文中にのみあらわれている，

場合においては，エラー(22)になる。

しかし，

(31)　変数名又は配列名のEQUIVALENCE宣言が落ちて，しかも宣言されなかった変数又は配列が代入文の左辺のみ，又は左辺と右辺の両方にあらわれる，

場合には，フォートラン文法上は誤りではない。すなわちEQUIVALENCE宣言が落ちているか否かはプログラム作成者の判断によるしかない。

今，ここではEQUIVALENCE宣言が落ちたものと仮定するならば，

図4.2　プログラム図4.1における配列要素の結合の状態

対応			数値
IA (1 , 1)	—	IC (1)	: 1 1
IA (2 , 1)	—	IC (2)	: 2 1
IA (3 , 1)	—	IC (3)	: 3 1
IA (1 , 2)	—	IC (4)	: 1 2
IA (2 , 2)	—	IC (5)	: 2 2
IA (3 , 2)	—	IC (6)	: 3 2
IA (1 , 3)	—	IC (7)	: 1 3
IA (2 , 3)	—	IC (8)	: 2 3
IA (3 , 3)	—	IC (9)	: 3 3
IB (1)	—	IC (10)	: 1
IB (2)	—	IC (11)	: 2
IB (3)	—	IC (12)	: 3

(32)　EQUIVALENCE宣言文中にあるべきはずの変数名又は配列名の値が相互に結合されていない。

結果になる。エラー(32)はさらに具体的には，

(33)　変数又は配列の値が初期値（多くの場合ゼロ）のままで全く変らない。

QUIVALENCE宣言文中の配列要素の対応の誤り

配列名に対してQUIVALENCE宣言をする場合には，配列添字がつけられる。配列名の添字は記憶場所の対応を与えるために重要な役割をもつ。図4.3には，プログラム図4.1のQUUIVALECCE宣言において　EQUIVALENCE

(IB (1), IC (10)) $\xrightarrow[\text{誤き換}]{}$ (IB (1), IC (12))

とおき換えた場合のプログラム例及びWRITE文による出力結果を示してある。図4.1の場合との違いはC (10), C (11), C (12) の値にある。

図4.3のプログラムにおける配列の対応関係及び数値は図4.4に示したとおりである。

図4.3　プログラム図2.6においてEQUIVALENCE宣言を誤って
(IB (1), IC (10)) → (IB (1), IC (12))
とした場合の症状

```
内部文番号              フォートラン文
  ⋮                       ⋮
0003            EQUIVALENCE (IA (1 , 1) , IC (1)), (IB (1), IC (12))
  ⋮                       ⋮

   WRITE文による出力結果
IA (I) =11, 12, 13, 21, 22, 23 , 31 , 32 , 33 . : IB ( I) = 1 , 2 , 3 .
IC (I) =11 , 21 , 31 , 12 , 22 , 32 , 13 , 23 , 33 ,          0 , 0 , 1.
```

図４.４　プログラム図４.３における配列の対応及びそれらの数値

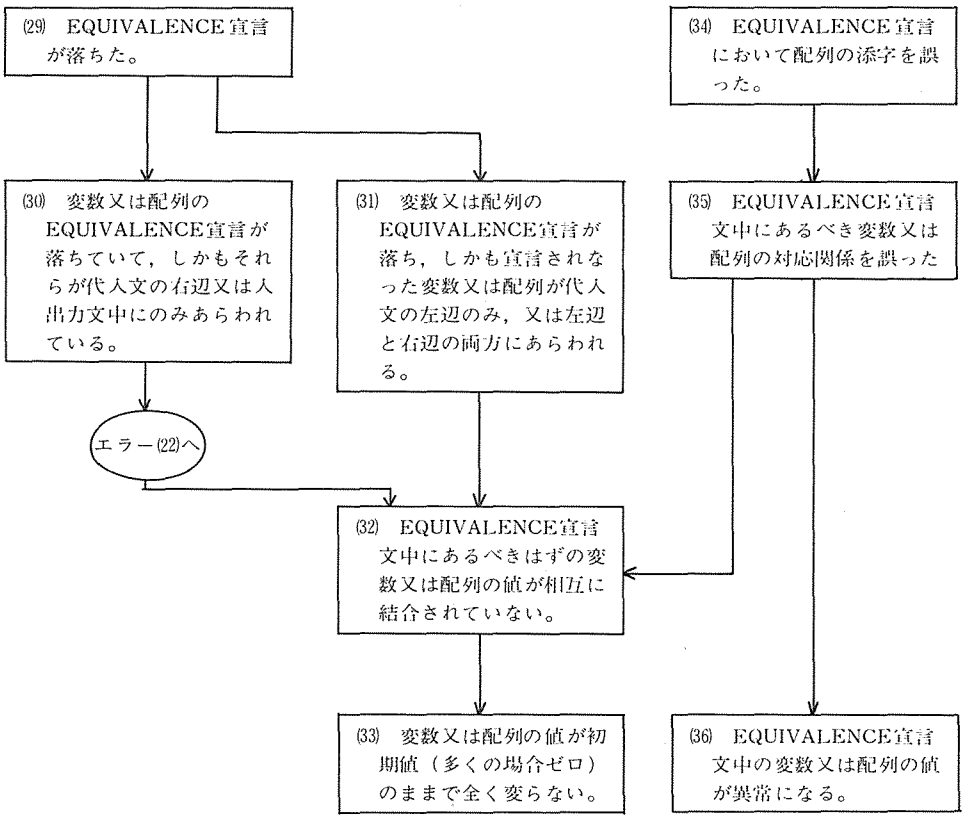
対応		値
A (1 , 1)	— C (1)	1 1
A (2 , 1)	— C (2)	2 1
A (3 , 1)	— C (3)	3 1
A (1 , 2)	— C (4)	1 2
A (2 , 1)	— C (5)	2 2
A (3 , 2)	— C (6)	3 2
A (1 , 3)	— C (7)	1 3
A (2 , 3)	— C (8)	2 3
A (3 , 3)	— C (9)	3 3
	— C (10)	0
	— C (11)	0
B (1)	— C (12)	1
B (2)	—	2
B (3)	—	3

以上の結果から見て 次のことがいえる。

(34)　EQUIVALENCE宣言において，配列の添字を誤った，

場合には

図４.５　EQUIVALENCE宣言における誤りの症状



(35) EQUIVALENCE宣言文中にあるべき変数又は配列の結合関係（対応関係）を誤ったことになり，エラー(33)に帰着するか又は

(36) EQUIVALENCE宣言文中の変数又は配列の値が異常になる。

以上の結果を流れ図にまとめたものが図4.5である。

5. EXTERNAL文

フォートラン・プログラムは次の4種類の手続きからなる。

文関数

組込み関数

外部関数

外部サブルーチン

特に外部関数と外部サブルーチンをまとめて外部手続きという。詳細は以後の回に譲ることにする。EXTERNAL文は外部手続きの引数として外部手続きを使用する時に，その手続きを変数と区別するための宣言文である。

5.1 EXTERNAL文の文法

EXTERNAL文は次の形式で表現される。

EXTERNAL文 EXTERNAL文(EXTERNAL statement)は，つぎの形とする。

EXTERNAL $v_1, v_2, \cdots, v_n$

ここで，$v_1, v_2, \cdots, v_n$は，いずれも外部手続き名とする。

EXTERNAL文の中に現われた名前は，その名前が外部手続き名として宣言されるものとする。外部手続き名を他の外部手続きの引数として使用する場合には，この外部手続き名はその使用されるプログラム単位内のEXTERNAL文の中に現われなければならない。

例：EXTERNAL文

使用例：

```
C MAIN PROGRAM
   EXTERNAL SIN
      ⋮
 10  Y=R+ABLE (SIN,R,T)
      ⋮
     END
     FUNCTION ABLE (XFUN,A,B)
     ABLE=XFUN (A) +B**2
     RETURN
     END
```

主プログラムの中の番号10を持つ文の実行中にABLEという名前を持つ関数副プログラムが呼ばれ，その中の算術代入文

```
ABLE=XFU (A) +B**2
```

は

```
ABLE=SIN (R) +T**2
```

として実行される。

5.2 EXTERNAL文のコンパイル時エラー・メッセージ

5.2.1 WARNING

030	WARNING: REDUNDANT DECLARATION FOR "name" IN EXTERNAL STATEMENT ○1つのEXTERNAL文で同じ英字名が2回以上宣言されているか，または2つ以上のEXTERNAL文で同じ英字名が2回以上宣言されている。そのままコンパイルされる。 （プログラム　コード　-01_8）

（例）　内部文番号　　フォートラン文

003　　EXTERNAL　COS ,SI N ,COS

関数 COSが2度宣言されている。

5.2.2 FATAL ERROR

471	ILLEGAL EXTERNAL STATEMENT SYNTAX ○EXTERNAL文に文法違反がある。適当な仮定のもとにコンパイルされるが，このEXTERNAL文を含むプログラム単位はGO ファイル上に出力されない。

EXTERNAL文における外部手続き名の宣言のしかたが文法に違反する場合に出力されるエラー・メッセージである。誤りは適当な仮定のもとにコンパイルされる。

（例1）　EXTERNAL　A.B　　は

EXTERNAL　A,B　　と仮定される。

（例2）　EXTERNAL　2A,B,3C　　は

EXTERNAL　A,B,C　　と仮定される。

472	DATA NAME "name" USED IN EXTERNAL ○データ名がEXTERNAL文に現われた。その名前はEXTERNAL文にはないものとしてコンパイルされるが，このEXTERNAL文を含むプログラム単位はGO ファイル上に出力されない。

（例）　内部文番号　　フォートラン文

0002　　REAL　A(10)

0003　　EXTERNAL　A

Aは配列宣言子で配列名であると宣言されている。しかしEXTERNAL文で再度，外部手続きの宣言がなされている。これら両者は矛盾する。

5.3 リンクロード時及び実行時におけるEXTERNAL文の誤り

EXTERNAL文は外部手続きを引数として引用するためのものである。したがって，EXTERNAL宣言された外部手続き名が誤りであるか否かはリンクロード以後になるまで確定できない。特にEXTERNAL宣言は外部手続きが最小限2段階以上の引用がなされるものであるため，一般に複雑な様相を示す。

以下，具体例により，誤りの症状を示す。図5.1には，

$$\sin^2 x + \cos^2 x = f$$

を各argument xについて計算するためのプログラムを示した。
（このプログラムは$\sin x$と$\cos x$を混みにした，全体の誤差を見るためのものである。）このプログラムは正しく目的を達成したプログラムである。SIN, COSをEXTERNAL宣言する。そしてDO—ループにおいてargnment xを変えながら関数RADIUSを呼ぶ。関数RADIUSにおいてSIN, COSは外部手続き名として引用される。図5.1のプログラムに対するWRITE文での出力結果は図5.2のとおりであった。

図5.1 EXTERNAL文をテストするためのプログラム。このプログラムは（$\sin^2 x + \cos^2 x$）につい変数xのいくつかの数値について計算するものである。プログラムは意図通りに正しくつくられている。

```
      CCCC  MAIN PROGRAM FOR THE TEST ABOUT EXTERNAL STATEMENT
0001        EXTERNAL SIN,COS
0002        DIMENSION FR(45)
0003        WRITE(6,100)
0004    100 FORMAT(1H3, ▼THE TEST ABOUT EXTERNAL—STATEMENT.▼)
0005        X=2.0
0006        DO 10 N=1,45
0007        FR(N)=RADIUS(SIN,COS,X)
0008     10 X=X+2.0
0009        WRITE(6,200)
0010        WRITE(6,300) (FR(N),FR(N+1),FR(N+2),FR(N+3),FR(N+4),
           1N=1,45,5)
0011    200 FORMAT(1H2,▼      OUT—PUT OF THE RESULT▼)
0012    300 FORMAT(1H ,5E18.10)
0013        STOP
0014        END
      CC
0001        FUNCTION RADIUS(F1,F2,X)
0002        RADIUS=F1(X)**2+F2(X)**2
0003        RETURN
0004        END
```

図5.2　プログラム図5.1における出力結果

```
THE  TEST  ABOUT  EXTERNAL-STATEMENT.
   OUT-PUT  OF  THE  RESUL1
.9999999999E+00   .9999999998E+00   .9999999999E+00   .9999999999E+00   .9999999999E+00
.1000000000E+01   .9999999997E+00   .9999999999E+00   .1000000000 E+01  .9999999999E+00
.9999999998E+00   .1000000000E+01   .9999999992E+00   .9999999998E+00   .9999999999E+00
.9999999999E+00   .9999999999E+00   .9999999999E+00   .9999999990E+00   .1000000001E+01
.9999999988E+00   .9999999998E+00   .1000000001E+01   .9999999999E+00   .9999999999E+00
.9999999998E+00   .1000000000E+01   .1000000003E+01   .9999999998E+00   .9999999982E+00
.9999999999E+00   .9999999999E+00   .9999999998E+00   .1000000000E+01   .9999999970E+900
.9999999998E+00   .1000000001E+01   .9999999999E+00   .1000000002E+01   .9999999999E+00
.9999999999E+00   .9999999999E+00   .1000000000E+01   .99999999998 E+00 .9999999999E+00
```

$(\sin^2 x + \cos^2 x)$ の値は 10^{-10} 程度の誤差をもつことがわかった。

さて，ここで誤りの例として

(37)　EXTERNAL文又はその他の誤りのために，外部手続き名のEXTERNAL宣言が無効になる，

場合について調べてみる。図5.1のプログラムにおいて

```
EXTERNAL  SIN, COS
          ↓
EXTERNAL  SIN  COS
```

のように作為的にSINとCOSの間のコンマを消してみる。この時EXTERNAL　SIN　COSは外部手続き名SINCOSがEXTERNAL宣言されたことになり，このこと自体は誤りでない。しかし外部手続き名SINとCOSは宣言されないことになる。図5.1のプログラムで，EXTERNAL　SIN, COS→EXTERNAL　SIN COSとおきかえた時のプログラム実行結果は，図5.3のとおりである。SINCOSはプログラム中で実際に引用されることがないため，リンクロード時においてもエラーとならなかった。

結局，実行の際に関数RADIUSを引用するとプログラムの実行が誤った方法ですすみ，図5.3に示された結果となった。

この現象は

(38)　外部手続き名に対して，必要とされるEXTERNAL宣言が無い，

図5.3　プログラム図5.1においてEXTERNAL文を変えた時の症状

```
                ⋮
0001              EXTERNAL  SIN  COS
                ⋮

   実行結果
  THE  TEST  ABOUT  EXTERNAL-STATEMENT.
*****MRMOOI  UEP  0000000  OP-CODE  ERR  =  04
```

場合にも全く同じである。エラー(37)及び(38)は結局

(39) 実行時エラー

MRMOOI UEP00000000 OP-CODE ERR=04

となる。

一方，上記の誤りに対して

(40) プログラム中に存在しない外部手続き名がEXTERNAL宣言され，しかもCALL文においてその外部手続き名が実引数として引用されている，

場合も誤りである。この時は

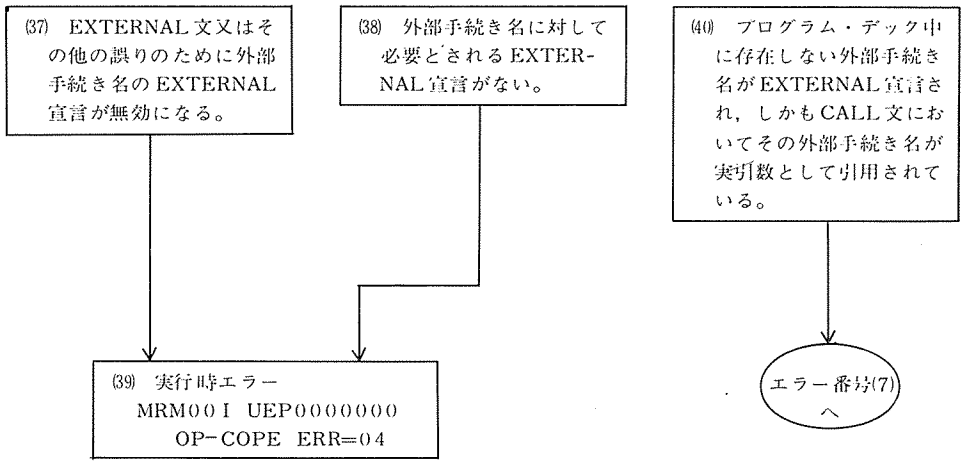
(7) リンクロード時にエラー

LLK17I ,LLK38I

が出力される。

以上を流れ図にしたものが図5.4である。

図5.4 EXTERNAL文に関連する誤りの症状

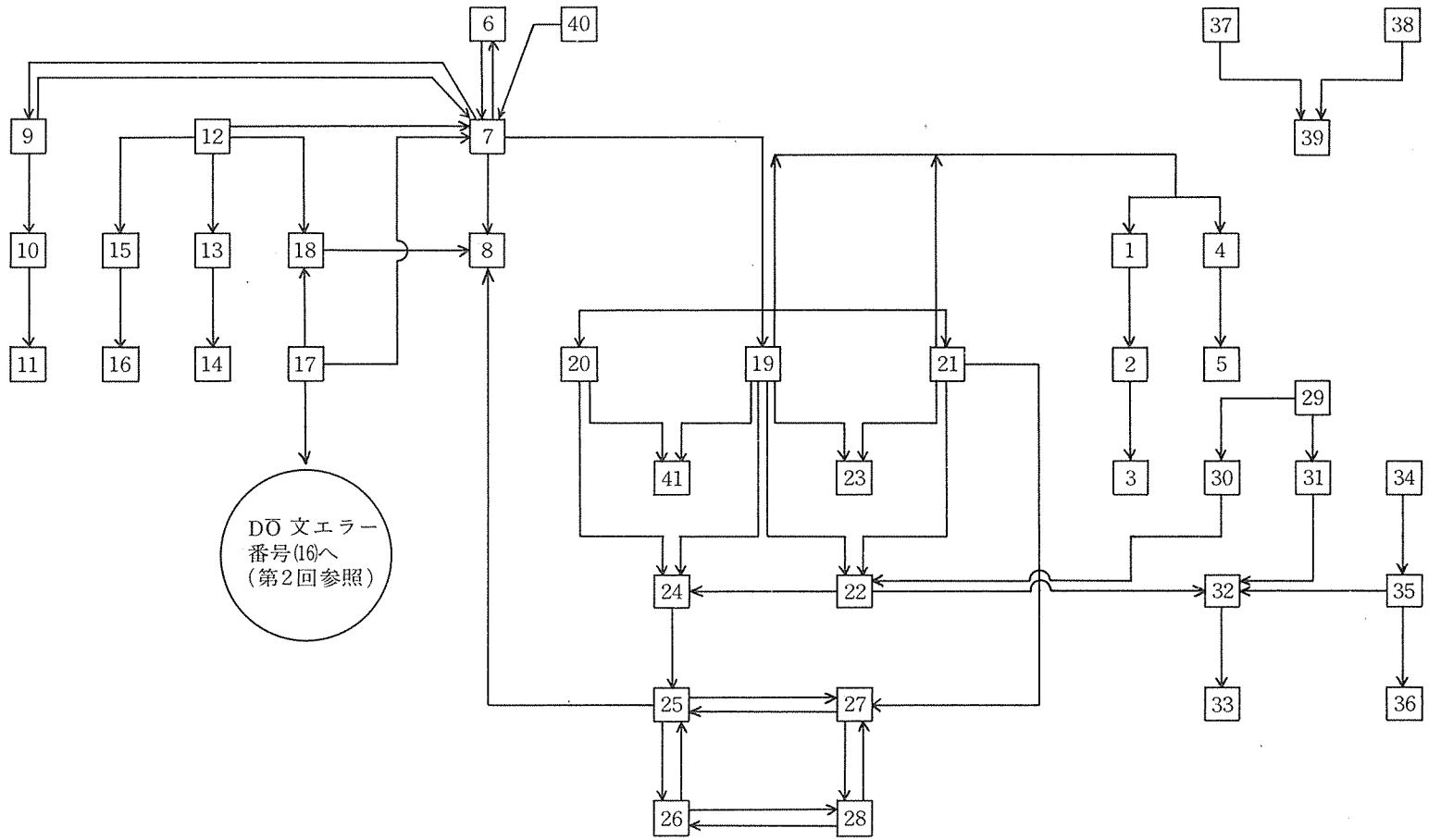


6. まとめ

宣言文に関する誤りの主な特徴は，２つ以上のフォートラン文の間での意味の矛盾することに原因することである。このために，誤りを調べる際には広範囲にわたって宣言文と他の文の間に矛盾のない事を１つ１つ確かめてゆかなければならない。ここに示した誤りの例は，プログラム作成時に意図していたことを，宣言文において逸脱してしまった時の症状を示した。このようなことは大きなプログラムを作成する時には頻繁に起こることであり，それでいて発見困難である。

今後，デバッギングをおこなう際に，「アルゴリズムは正しくしかも各部分で実行された内容は局所的に見る限りは正しいにもかかわらず，プログラムは正しく働かない」場合にはぜひとも宣言文に疑いを持って調べることをお推めしたい。

付録　宣言文誤りの流れ図



(1) 配列要素が代入文の右辺にあらわれているにもかかわらず，配列宣言がなされていない。

(2) 配列名は関数副プログラム名として処理される。

(3) リンクロード時：エラー・メッセージが出力される。

LLK17I *****

LLK38I *****

(4) 配列要素が代入文の左辺又は入出力文にあらわれているにもかかわらず，配列宣言がなされていない。

(5) コンパイル時：エラー・メッセージが出力されている。(番号256)

(6) 配列宣言において，配列の次元の数を誤って多くとりすぎた。

(7) COMMON文，EQUIVALENCE文での配列や変数の結合に誤りがある。

(8) 配列要素の（数）値が正しく入っていない。

(9) 配列宣言において，配列の次元の数を誤って少くとった。

(10) 実行文中で配列宣言より多い次元の数で配列を用いた。

(11) コンパイル時：エラー・メッセージが出力されている。(番号277)

(12) 配列宣言において，配列宣言子の寸法を誤って大きくとりすぎた。

(13) 配列の寸法が$2^{18}-1$をこえている。

(14) コンパイル時：エラー・メッセージが出力されている。(番号416)

(15) プログラムのサイズが大きすぎる。

(16) リンクロード時又は実行時にエラー・メッセージが出力される。

リンクロード時：LLK13I*****

実行時　　：MRM00I*****

(17) 配列宣言において，配列宣言子の寸法を誤って小さくとっている。

(18) 配列要素の引用又は配列要素への代入において誤った要素の引用又は代入をおこなう。

(19) COMMON文中にあるべき変数名又は配列名を落した。

(20) COMMON文中で配列宣言子の次元の数又は寸法をまちがえた。

(21) 1つのプログラム単位内でCOMMON文全体を落した。

(22) コンパイル時：エラー・メッセージが出力されている。(025 WARNING:)

(23) コンパイル時：エラー・メッセージが出力されている。(番号256)

(24) COMMON文中で配列又は変数の値が相互に対応しない英字名の所にわりあてられている。

(25) COMMON文中にある（又はあるはずの）配列又は変数の値が異常である。

(26) COMMON文中にあるべき配列又は変数の値がゼロである。

(27) 副プログラム引用の際に正しい値の受け渡しができていない。

(28) COMMON文中にある（又はあるべきはずの）配列又は変数の値が正しく記憶されていない。

(29) EQUIVALENCE宣言が落ちた。

(30) 変数又は配列のEQUIVALENCE宣言が落ちていて，しかもそれらが代入文の右辺又は入出力文中にのみあらわれている。

(31) 変数又は配列のEQUIVALENCE宣言が落ちて，しかも宣言されなかった変数又は配列が代入文の左辺のみ又は左辺と右辺の両方にあらわれている。

(32) EQUIVALENCE宣言文中にあるべきはずの変数又は配列の値が相互に結合されていない。

(33) 配列又は変数の値が初期値（多くの場合ゼロ）のままで全く変らない。

(34) EQUIVALENCE宣言において配列の添字を誤った。

(35) EQUIVALENCE宣言文中にあるべき変数又は配列の対応関係を誤った。

(36) EQUIVALENCE宣言文中の変数又は配列の値が異常になる。

(37) EXTERNAL文又はその他の誤りのために外部手続き名のEXTERNAL宣言が無効になる。

(38) 外部手続き名に対して，必要とされるEXTERNAL宣言が無い。

(39) 実行時：エラー・メッセージが出力される。

MRMOOI UEP0000000 DP-CODE ERR=0 : 04

(40) プログラムデック中に存在しない外部手続きがEXTERNAL宣言され，しかもCALL文においてその外部手続き名が実引数として引用されている。

(41) リンクロード時：エラー・メッセージが出力される。

LLK67I*******