（第８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）

{ READ文
WRITE文
補助入出力文
FORMAT文 }

大型計算機センター　　磯　本　征　雄

1. はじめに

入出力文は，入出力装置を使って外部記憶媒体と主記憶装置の間のデータの交換をおこなうためのフォートラン文です。これは，入力のためのREAD文，出力のためのWRITE文とこれらの入出力の対象となるレコードの記憶媒体上での位置を定めるための補助入出力文から成っています。また，READ文とWRITE文には書式つきと書式なしの２種類に分けられます。書式つきREAD文・WRITE文のために，さらにFORMAT文が必要です。ここでは，バッグ及びデバッグを中心とした解説をおこなうことにしていますので，文法の詳細は他の文法専門書を参照していただくことにいたします。[1)]

入出力文がフォートラン・プログラムにおけるアルゴリズムの中心的役割りをはたすことはあまりありません。その役割りを大別すれば

- 初期設定データの入力
- 処理結果の出力
- 処理の中間過程でのデータの１時的退避と再入力
- そ　の　他

等になります。変数値そのものを再定義するために使われる場合もあるし，COMMON文のように単に副プログラム間でのデータの受け渡しにのみ使われる場合もあります。

このように，入出力文はその使われ方によっては代入文と同様の特徴をもち，あるいはCOMMON文と同じ特徴をもつ場合があります。プログラム・デバッギングの面からこれらを見ると，これら２つの側面，すなわちデータの再定義という面とデータの転送という両面から見なければならないということです。したがって入出力文のデバッグは，考慮すべき内容の多さの点で他のステートメントに比べて複雑といえます。

また，入出力文は単にフォートランの文法の範囲にとどまらず，その命令の具体的対象となるハードウェアの知識がなければ，その機能を十分に発揮できません。さらに，デバッギ

ングに至っては，ジョブ制御言語についてもある程度の知識をもつ必要があります。ジョブ制御言語については，センター利用の手引[2)]等としてセンターより別に出版されていますのでそれらをお読み下さい。ここでは，これら多面的複雑さを避けて，フォートラン文法に密接に関連した部分のみについてのデバッギングに関する解説をすることにいたします。

2. 入出力装置と入出力文

まずハードウェァの面から見ることにしましょう。当大型計算機センターで現在サービスの対象になっている入出力装置及びそれらのシンボリック・ユニット名，ロジカルユニット番号は表2.1のとうりです。これらが各々ここで話題として取り上げているフォートランの

表　2.1　EORTRAN 700で使用可能な周辺機器とその内容

ファイル参照番号	シンボリックユニット名	ファイルテーブルサブプログラム名		
1	MR 2	NFSAUNIT 1	磁気テープ	2400フィート
2	MR 3	NFSAUNIT 2	同　　上	〃
3	MR 4	NFSAUNIT 3	同　　上	〃
4	MR 5	NFSAUNIT 4	同　　上	〃
5	SIU	NFSANCHOR	カード読取装置	800ch/REC
6	SPR	NFSANCHOR	高速製表印字装置	132ch/REC
7	SPU	NFSANCHOR	カード穿孔装置	80ch/REC
8	MR 6	NFSAUNIT 8	磁気テープ	2400フィート
9	MR 7	NFSAUNIT 9	同　　上	〃
		OSAKAUNIT9	磁気ディスク	20000枚/カード

入出力文の対象となるハードウェァです。ユニット番号は，一応標準としてこのように定められたものであり，必ずしも不変的意味をもつものではありません。また，これら磁気ディスクや磁気テープの使用法に関しては，計算機利用のための運用方法の実状に合せて，使用機器の台数や具体的利用法に対する制限事項が，フォートラン文法とは別に，運用上の制約として定められています。この詳細については，計算機システムの利用の手引き[2)]として詳しく解説されていますので，そちらを参照して下さい。理解を容易にするために，図2.1に計算機とその入出装置に関する模式図を示しておきましょう。

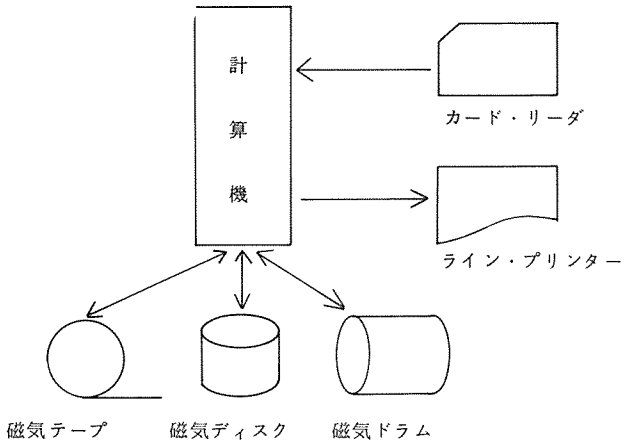

図　2.1　入出力文の対象となる主要入出力装置

ここで，フォートラン・ステートメントにおける入出力文とは，例えば図2.1に示したカードリーダ，ラインプリンター，磁気ディスク装置，磁気テープ装置等を働らかせこれらを利用するためのものです。当然，個々のハードウェアに対して使われる入出力文には自ら固有の制約があります。これらの関係を表2.2に簡単に示しておきました。

表 2.2　ＲＥＡＤ文とＷＲＩＴＥ文及びそれらに対応する使用可能な入出力装置

×印は，使用不可能

○印は，使用可能であることを示す。

		磁気ディスク	磁気テープ	カードせん孔装置	カード読取装置	高速製表印字装置
直列形入出力文	書式つき READ文	○	○	×	○	×
	書式なし READ文	○	○	×	×	×
	書式つき WRITE文	○	○	○	×	○
	書式なし WRITE文	○	○	×	×	×
ダイレクトアクセス入出力文	書式つき READ文	○	×	×	×	×
	書式なし READ文	○	×	×	×	×
	書式つき WRITE文	○	×	×	×	×
	書式なし WRITE文	○	×	×	×	×

さて，入出力文には，つぎの３種類があります。

- ○直列形入出力文
- ○ダイレクト・アクセス入出力文

（ ○ENCODE 文と DECOCD 文 ）

ここで，第３番目のENCODE 文と DECODE 文は，いずれも内部記憶装置内でデータの転送を行なうための文ですが，使用されることが比較的少ないのでここでは省略することにします。

入出力文は，さらにその中を詳しくみると次のような構造になっています。

直列形入出力文		ダイレクト・アクセス入出力文
○ＲＥＡＤ文		○ＲＥＡＤ文
○ＷＲＩＴＥ文		○ＷＲＩＴＥ文
○ＢＡＣＫＳＰＡＣＥ文	補助入出力文	○ＤＥＦＩＮＥ　ＦＩＬＥ文
○ＲＥＷＩＮＤ文		○ＦＩＮＤ文
○ＥＮＤＦＩＬＥ文		

ＲＥＡＤ文とＷＲＩＴＥ文については，さらにＦＯＲＭＡＴ文による書式のついているものと書式のついていないものの２種類があります。ＲＥＡＤ文及びＷＲＩＴＥ文に関して，各々書式つきか否かは入出力文の対象となっている入出力装置によって自ら制約を受けます。
また，直列形入出力文かダイレクト・アクセス入出力文かのいずれであるかということについても，やはり入出力装置との関連により制約を受けます（表2.2参照）。

ここに示したように，入出力文は他のフォートラン・ステートメントにくらべて非常に多様な面があります。上記のことと重複しますが，入出力文をプログラム中で見る時に，単に形式的なステートメントとしてだけでなく，同時にその背影をも考えなければなりません。
例えば，

① 入力文か　出力文か？
② 直列形か　ダイレクト・アクセス形か？
③ 書式つきか　書式なしか？
④ 対象となっている入出力装置は何か？
⑤ システム運用上のジョブ・コントロール・カードとの関係は？
⑥ システム運用上，許可されているか？
⑦ 入出力装置の特徴から見て余りに非常識な使用法になっていないか？

等の様々の側面から見る必要があります。特に，プログラム・デバッギングにおいては，これらの内のいずれかを見落してしまうと，応々にして重大なバッグを見のがしてしまうことになります。

プログラム・デバッギングの立場から入出力文を見ると，単なる文法規則よりはむしろ具体的使用対象となっている計算機及びそれらの運用形態のほうがむしろ大切ともいえます。そこで今回の解説は，紙数の関係もあって文法規則についての詳細については省略することにします。詳しくは，文献 3），4）を参照して下さい。

3. 入出力文に関するコンパイル時のバッグ

入出力文は，FORTRAN文法として決められた，計算機システムに直接依存しない部分と，具体的計算機システム及びそれらの運用形態に直接関連した部分の両面の内容をもっています。コンパイル時のバッグは，この内の計算機システム及びその運用に直接関係しない部分についてのチェックであるという点に十分注意しておくべきでしょう。ただし，ダイレクトアクセス型入出力文については，ＪＩＳ－FORTRANにはその規定がありませんので，その文法規則自体が計算機システムに依存しているともいえないこともありませんが，この点は無視することにします。

3.1　文単位のチェックで十分なバッグ

前述のように，入出力文のバッグは入出力文そのものの誤りのほかに，他の宣言文や実行

文の誤りによる派生効果又は他の文との相互関係の取り方のまずさによる部分が少なくありません。ところが，ここで問題にしている入出力文自体の誤りとしては，多くの場合，例えば

```
①  READ（5,100）  A ＊ B
②  WRITE）6,200）  A
          ・
          ・
          ・
```

等のように，当然コンパイラーによるチェックにひっかかり，しかもエラーメッセージが出力されて説明の余地なく“なんだ，こんな間違いか”とか“おや，こんなつまらないミスをして”等とすぐに誤りを発見できるものが多いようです。個々のエラー・メッセージに関する解説は，紙数の関係及びこのような理由から，今回は省略することにします。

3.2　未定義変数の出力又は未定義配列の入出力

ここでは入出力文と他の文との関連の誤りによるバッグあるいは他の文の誤りによる派生効果のためにおこる見掛け上のエラー・メッセージの出力されるような例をまとめてみることにいたします。

WRITE文によって出力されようとする変数は，予め何らかの形で定義されているのが当然でしょう。ところが，現実には，例えば

```
          ・
          ・
          ・
ABCDEF＝XYZ＋UVW
          ・
          ・
          ・
WRITE（6,100）ABCDEE
          ・
          ・
          ・
```

のような形式のプログラム・ミスを犯すことがあります。すなわち，算術代入文で定義されたＡＢＣＤＥＦなる変数をWRITE文によって出力しようとしたのだが誤ってＡＢＣＤＥＥとしてしまったために，WRITE文の所で

025 WARNING：UNDEFINED VARIABLE “ABCDEE”

が出力されることになります。ところが，この誤りは逆に算術代入文におけるＡＢＣＤＥＦが実はＡＢＣＤＥＥでなければならない場合であっても話は全く同じです。特に，この誤りはFATAL ERRORでないのでそのまま実行されて一見奇妙な出力結果を得てしまうことにもなります。

一方，入出力文において例えば

```
READ（5,100）        （A（I），I＝1,10）
WRITE（6,200）       （A（I），I＝1,10）
```

等のＤＯ型並びの入出力文において，もし配列Ａの配列宣言がなされていない場合，これはやはり FATAL ERROR

178 SYNTAX ERROR IN I/O LIST

となります。これは，多くの場合配列宣言が欠落しているようです。このように入出力並びに関する誤りは，その文だけでなくプログラム全体にわたって，その変数の占める役割りを吟味しなおす必要があります。

3.3 READ, WRITE 文と FORMAT 文の関連

書式つき READ, WRITE 文には，必ず FORMAT 文が付随していなければなりません。そして，この両者は文番号により結びつけられます。しかもその対応は１対１でなければなりません。すなわち，同じ文番号が２つ以上あることは許されません。

さて，問題点はもう一つあります。それは，ラインプリンターへの書き出しにおいて，１行当りの文字数が１３３文字以内であるということです。例えば

```
WRITE(6,100)    (X(I), I=1, 10)
100  FORMAT(1H␣, 10E20.15)
```

を実行したとしましょう（但し参照番号６はラインプリンターであるとする）この文による結果は，一行分の出力が不可能であるためにハードウェアから見て誤りとなります。この点については FORTRAN７００ではコンパイル時エラーメッセージは出力されません。ただし，１つの欄記述子が１３３文字以上の Field を占めている場合には

059 WARNING : FIELD WIDTH IS GREATER THAN 133

が出力されます。参考のため入出力装置のシンボリックユニット名と各々に許される１記録の長さを表 3.1 に示しておきました。

表 3.1 シンボリック・ユニット名と許される記録の長さ

シンボリック ユニット名	許される記録の長さ
ＳＩＵ	８０字[(1)]（書式つきでなければならない）
ＳＰＲ	１３３字[(1)]（書式つきでなければならない）
ＳＰＵ	８０字[(1)]（書式つきでなければならない）
ＳＴＵ	６０字[(1)]（書式つきでなければならない）
その他	書式つき：１３３字（１物理レコードの長さ＝１論理レコードの長さ）
	書式なし：３２０字（１物理レコードの長さ）

注：(1) この長さを変更することはできない。ＦＯＲＭＡＴ文あるいは配列内書式により定められた１記録がこれらの長さに満ない場合は，後ろに空白が補われる。

(2) これらの長さは，日本電気より提供される標準のファイル・テーブル・サブプログラム内において定められている長さである。

3.4　入出力文とファィル参照番号

各々の入出力文は,表2.1に示した,ファィルの内のいずれかをその入出力の対象としています。その内のいずれのファィルを参照するかということは，ファィル参照番号により対応づけられます。FORTRAN－700においては，このファィル参照番号として標準を1～16の範囲で定めています。（但し，これ以外のものを使用する手法については文献４）を参照して下さい）。したがって，もし

READ(20, 100)X, Y, Z

等のようにファィル参照番号が16を超えている場合には，エラー・メッセージ

174　ILLEGAL FILE REFERENCE NUMBER IN I/O
　　STATEMENT - SHOULD BE INTEGER LESS THAN 17.

が出力されます。

コンパイル時のエラー・メッセージ出力対象にはなりませんが,READ文あるいはWRITE文の入出力の対象が誤ったファィル参照番号に対応づけられることがあり得ます。例えば標準として

ファィル参照番号	入出力装置
5	カードリーダ
6	ラインプリンター

となっていますが，もしプログラム中で

READ(6,100) X,Y,Z
WRITE(5, 200)X, Y, Z

等があらわれても，これらはコンパイル時のチェックには掛かりません。後述しますが，これらは実行時にはじめてエラーとして検出されます。物理的に見て，ラインプリンターからデータを読み込むことも，カードリーダへデータを書き出すことも出来ないことからこれらは明らかでしょう。

しかし現実には，ミス・パンチの類としてこのような誤りも有り得ますので，コンパイル時の段階でもこの点には十分気を配っておくことが良いでしょう。もっとまずい場合には，磁気テープ等を使用した場合に，うっかりファィル参照番号を誤ったばっかりに，予定外の入出力装置に対して書き込みや読み出しを行ってしまい，しかも外からそれが発見できない場合すらあるからです。

以上，コンパイル時における入出力文に対する主要なチェックポイントのみについてまとめてみましたが，計算機によるチェックはあくまでも文法的側面が主体であって，運用方法等には余り関与していないのだという点を重ねて強調しておきます。

4. 入出力文に関するリンクロード時のバッグ

表2.1に見るように，入出力文あるいはそれに対応する入出力装置には，それぞれ固有のファィルテーブル・サブプログラムがリンクロード時にプログラムの1部分としてリンクされていなければなりません。その具体的手法については文献4）に譲ることにしましょう。ただし，簡易コントロール・カードでシンボリックユニット名（入出力装置）及びファィル参照番号を指定している場合には，これらは計算機内でファィル・テーブル・サブプログラムの結合を自動的にやっていますので必ずしもこの限りではありません。また，カードリーダとラインプリンターに関しても利用者側でファィル・テーブル・サブプログラムの結合を行う必要はありません。

さて，もし必要なファィル・テーブル・サブプログラムが結合されていなかった場合にはどうなるでしょう。その場合には，リンクロード時にエラー・メッセージ

UNRESOLVED ENTRY ファィル・テーブル・プログラム名

が出力されます。

5. 入出力文に関する実行時のバッグ

入出力文のフォートラン・プログラム内における役割りは，大きく2つに分けられる。第1のものは計算機利用者とプログラムとのデータの交換である。具体的にいえば，READ文により変数値の初期設定を行ったり，WRITE文による処理結果の出力がそれです。第2のものは，計算処理の途中において生ずる多量の変数値を外部記憶装置に書き出したり再度これを主記憶装置へ読み込んだりする，いわゆる中間書き込み用ファィルとのデータの交換に使われます。

初期設定用ＲＥＡＤ文

初期設定のためのＲＥＡＤ文の最も常識的なものは，カードリーダからの入力です。カードリーダからの入力のためのＲＥＡＤ文は必ず書式つきでなければなりません。そして

書式つきREAD文は，FORMAT文と組になって働きます。したがって，常にデバッギングの過程においてもFORMAT文と対にしてチェックする必要がある。特に，カード・リーダからの読み込みのためのREADについては，さらにデータ・カード上のイメージとの対応についても調べなければならない。データ・カードからの入力のためのREAD文は，このように

ＲＥＡＤ文

ＦＯＲＭＡＴ文

データ・カード

の3者1体にして考えなければなりません。したがって，例えプログラムに誤りがあってもこれら3者の内のいずれが誤りであるかは，プログラムの意図を離れて論ずることはできません。そこで，まず最初に正しいプログラム及びそのデータカードを図5.1であると仮定し

たうえで，一体どのような誤りがあり得るかを考えることにします。

```
          :
     READ(5,100)  I, J, X, Y
100  FORMAT(2I5, 2F10.5)
          :
```

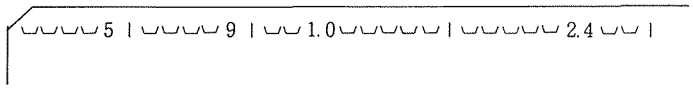

図 5.1　カード入力に対するＲＥＡＤ文及びデータ・カード上のイメージの例

プログラム実行における誤りの原因の１つは，

(1)　データカード上でのカードイメージの誤り

です。READ文中の変数の並びとFORMAT文中の入力書式が定まれば，データ・カード上のパンチの状態は自ら決まっているはずです。もし，入力書式に違反したイメージであれば，実行時にエラー・メッセージが出力され

(2)　FORMAT - MISMATCH のエラーとなる

ことになります。ところが，図5.1となるべきはずのデータ・カードを誤って間違ったカラム上にパンチしてしまっても偶然にピリオドの位置が入力書式にひっかからなかったために誤りを見逃がしてしまうことがあります（**図 5.2**）。特に，Ｉ変換のみで成っている入力書式に対しては，入力書式の工夫を凝らさなければこのような誤りを犯す危険はさらに大きくなります。ましてや，５を４にミス・パンチする等の誤りは，プログラマー自らがチェックする以外には調べる方法がありません。

```
          :
     READ(5,100)  I, J, X, Y
100  FORMAT(2I5, 2F10.5)
          :
```

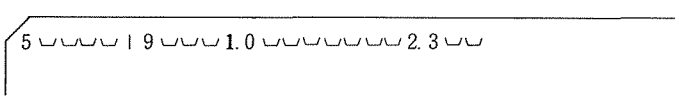

図 5.2　プログラム図5.1に対して誤ってここに示したようなデータ・カード・イメージを入力してもエラー・メッセージは出力されず，処理結果は当然誤りとなる

さて，データカード上のイメージとも関連したことですが，

(3)　READ文において入力並びの順序に誤りがある

場合にも，やはりデータカード上の値が入力並びに正しく対応しないために誤りとなります。

特にプログラムとは別に“プログラム利用説明書”なるものを準備して,それをたよりにデータ・カードをパンチする等の場合にはよくこのような誤りを犯してしまいます。
上記のものよりもさらにやっかいな誤りとしては,

(4) READ文において入力並びの変数名や配列名に誤りがある

場合があります。例えば,図5.3において第2番目の変数Iは,実はJであるべきであったとしてみましょう。この時には,Jの値は未定義のまま放置されることになります。この時,

```
      ⋮
      READ(5,100)  I, I, Y, X
100   FORMAT(2I5, 2F10.5)
      ⋮
```

図5.3 プログラム図5.1に対して,変数名Jを誤ってIとしてしまった場合の例である。この時,JはREAD文で定義されることなく放置される。

FORTRAN-700では,WARNINGが出力されることになります。ただし,万一JがDO文の制御変数や代入文の左辺に使われている場合には,WARNINGは出力されません。

```
      ⋮
      READ(5,100)  I, J, X, Y
100   FORMAT(2(I5, F10.5))
      ⋮
```

図5.4 整数型変数IとJ及び実数型変数XとYのそれぞれに対する入力書式の変換が正しくない。IとJに対してはI変換でありXとYに対してはF変換が各々対応すべきである。

入力並びとFORMAT文の入力書式の間の関係についての直接的な誤りとして

(5) FORMAT文中の入力書式に誤りがある

場合が考えられます。例えば,図5.4のように,整数型変数と実数型変数の入力並びに対して各々

I → I5
J → F10.5
X → I5
Y → F10.5

なる変換がそれぞれ対応している場合,常識的にみてこれは誤りでしょう。図5.4の例では入力並びの数も少く,入力書式も簡単なのでデバッグは簡単でしょう。しかし,入力並びの数が多くしかも入力書式が複雑な場合には決して簡単ではありません。ここで同じ図5.4に対しても,プログラマーによっては,

(6) READ文の入力並びとFORMAT文入力書式の対応が正しくない

という誤りであるかも知れません。バッグが(5)であるか(6)であるかは,単にプログラマーの

考え方の差異ともいえますが，できればプログラムの構造から判断するのがよいでしょう。

誤り(1)，(3)，(4)，(5)及び(6)は，プログラムの中での処理として結局

(7) ＲＥＡＤ文による変数・配列要素の初期設定が正しくないことによってバッグであることが判明するでしょう。ところで，初期設定に誤りがある場合には，その後の処理がことごとく誤りとなることは容易に想像されることです。その誤りの１つは

(8) 実行順序に誤りがある

ことです。実行順序の誤りをさらに詳細に見るならば，例えば算術ＩＦ文，算術ＧＯＴＯ文やＤＯ文のパラメター等に対してこれら誤って初期設定された変数が用いられている状況がその具体例としてあげられます。さて，(7)及び(8)はいずれもプログラム処理手順の途中において，

(9) 変数値・配列要素の値に誤りがある

ことにより，バッグの存在を知ることになるのです。

中間書き込み用ＲＥＡＤ／ＷＲＩＴＥ文

中間書き込み用ＲＥＡＤ／ＷＲＩＴＥ文は，その効果が直接外部に出ないためにバッグの発見が困難です。また，多くの場合書式なしで使われるでしょう。そこで一応書式なしREAD／WRITE文であるとして考えることにします。

書式つきであっても，それらは前記のREAD文の説明とほぼ同じになると思われるので，詳細は省略します。さて，

(10) 中間書き込みＲＥＡＤ文，ＷＲＩＴＥ文において入出力並びとＦＯＲＭＡＴ文入出力書式の対応に誤りがある

場合が，しばしば経験されることを注意しておきましょう。

中間書き込みに使われる外部記憶媒体は，通常ディスク・パック又は磁気テープです。したがって，ダイレクト・アリセス形入出力文に対しては

ＤＥＦＩＮＥ　ＦＩＬＥ文

ＦＩＮＤ文

が必要です。これら補助入出力文は，ＦＩＬＥから入出力されるべき記録の位置を定めるのに用いられます。したがって，

(11) 補助入出力文の使いまちがい

は，入出力されるべき変数値を誤ってしまうことになります。次に示したダイレクト・アクセス形入出力文

```
ＦＩＮＤ（８：ＩＤ８）
ＲＥＡＤ（８：ＩＤ８）　Ｘ，Ｙ，Ｚ
```

においてもしID8=5とすべき所を誤ってID8=7としてしまったならば，当然読み込まれたＸ，

Y，Zの値は間違いとなるでしょう。その他直列形入出力文に対しては，REWIND文や，BACKSPACE文が間違った場所や間違った順序で使われた場合もダイレクト・アクセス入出力文におけるFIND文の間違いと同様の誤りとなります。これらは場合によっては，

(12) 補助入出力文に対するエラー・メッセージが出力されている

こともありますが，しかし必ずしもそうとは限らないようです。

これら入出力文の使い間違いがエラー・メッセージの出力に結びつかない場合には，

(13) 中間書き込みREAD文・WRITE文において相互の入出力並びの対応に誤りがある

結果になります(**図 5.5**)。一方，中間書き込み入出力文は，多量のデータの外部記憶装置

```
      ⋮
WRITE(8) X, Y, Z, I, J, U, V

READ(8) I, J, X, Y, Z, U, V
```

図 5.5 対応するWRITE文とREAD文における入出力並びの対応の誤った例。
変数名の順序だけでなく，型についての対応も正しくない。

上への1時退避であるという性質上,WRITE文で書き出す記録の状況と再び主記憶装置上へREAD文により読み込む記録の状況が正確に意図どうりに対応していなければなりません。書式つき入出力文の場合には，入出力書式をも含めて正しく対応していることが必要です。したがって，これらの誤りを犯した場合にもやはりバッグとしては上記(13)となります。これらの誤りは，直接に発見されることは少く，たいてい

(9) 変数値・配列要素の値に誤りがある，

ことを知ることによりわかることが多いため，デバッグは容易ではありません。

ラインプリンター出力用WRITE文

ラインプリンターに出力された結果は，誤りであるか否かを容易に判断することができます。そして，それらを大きく分けると

(18) 出力変数値に関する誤り

(19) 出力書式に関する誤り

より成っています。これが如何なる原因により生じた誤りであるかは，簡単ではありません。まず，出力変数値に関する誤りについて考えてみましょう。

変数値が誤って出力されている場合でも，常に変数値自体についてのみの誤りであるとは限りません。

(14) WRITE文中の出力並びの順序とFORMAT文の書式の対応の誤り

の場合にも見かけ上変数値に関する誤りを見ることになります。例えば実数型変数Xを出力する時に

```
WRITE(6,100) X
100 FORMAT(I5)
```

としてしまうと，Xは整数型で出力されるだけでなく，もし|X|＜1.0である場合には常にゼロとなってしまいます。ましてや，I変換，E変換，F変換とA変換の使い間違い等は，大変奇妙な出力結果を得ることになります。特に複雑なFORMAT文に関しては，不本意なミスのためにこのようなバッグを生ずることがあります。

WRITE文以前の所での計算処理が正しく実行されていても，

(15) WRITE文の出力並びの順序に誤りがある

場合にも，ラインプリンター上の出力変数値が誤っているように見えます。すなわち，プログラマーの考えていた変数値の出力順序と現実のラインプリンター上の変数値の順序が違っているということです。このことは余りにも当然のように思われますが，しかし実際にこのような状況に直面すると，プログラマ自身の先入観が優先してしまってこのことに思いいたるのは必ずしも容易でありません。また，

(16) 出力順序の誤り

がある場合にもやはり，(18)に関連するバッグになりますが，これは(8)の実行順序に誤りがある場合の一つの結論としてとらえることができます。

最後に，(18)に関するバッグの最もたちの悪いものは，

(9) 変数値，配列要素の値に誤りがある，

場合にその原因の端を発する場合です。この時には，これまで前7回までの総てのバッグの可能性までも含めてその原因を調べる必要があります。これについては，前回までの解説を参照して下さい。

さて，WRITE文の内のもう一つの誤り，

(19) 出力書式に関する誤り

を考えてみましょう。この場合は，(18)の場合にくらべ原因は，さほど深くはありません。まず単純な原因としては，

(14) WRITE文中の出力並びの順序とFORMAT文の書式の対応の誤り，

があります。例えば，図5.6がそれです。出力並びの変数は6個あります。ところが出力書式中には5個しかありません。したがって最後の1文字Zは改行の後，I5で出力されることになります。このような誤りは，出力並びの多い時にしばしば生ずることです。

```
      WRITE(6,100) I, J, K, X, Y, Z
100   FORMAT(1H␣, I5, 2X, I5, 2X, I5, 2X, F10.4, 3X, F10.5)
                    ⋮
```

図5.6 出力並びとRORMAT出力書式の対応を間違えたプログラム例。

出力書式に関する誤りのうちで，つぎにしばしばおこるバッグは

(17)　ＦＯＲＭＡＴ文改行指定に関する誤り

です。改行は，

書式中の／に達した時

ＦＯＲＭＡＴ文最後のカッコに達した時

実行が新しいＷＲＩＴＥ文に達した時

のいずれかに行われます。また，改行の行数は，行の最初の文字により決められます。これに関する誤りが時に，非常に乱れた出力結果になることがあるのです。例えば，図５.７の場合，変数Ｉによって改行の量がかわってしまいます。つまり

```
Ｉ＝１１        改頁
Ｉ＝　２        １行改行
Ｉ＝２３        ２　〃
```

⋮

となるのです。これは，記録の第１字が改行指定に使われる結果このようなことになります。これについては文法説明書により各自研究してみて下さい。

```
                    ⋮
      WRITE(6,100)  I, X, Y
100   FORMAT(I2, 2E20.11)
                    ⋮
```

図５.７　改行指定が非常に乱れるプログラムの例

さて，最後の例としての

(16)　出力順序の誤り，

も見かけ上，出力書式の誤りに通じます。これは，ＷＲＩＴＥ文の実行される順序を間違えてしまったために生ずる誤りです。

以上のＲＥＡＤ文とＷＲＩＴＥ文に関する誤りの因果関係を整理すれば図 5.8 のようになります。

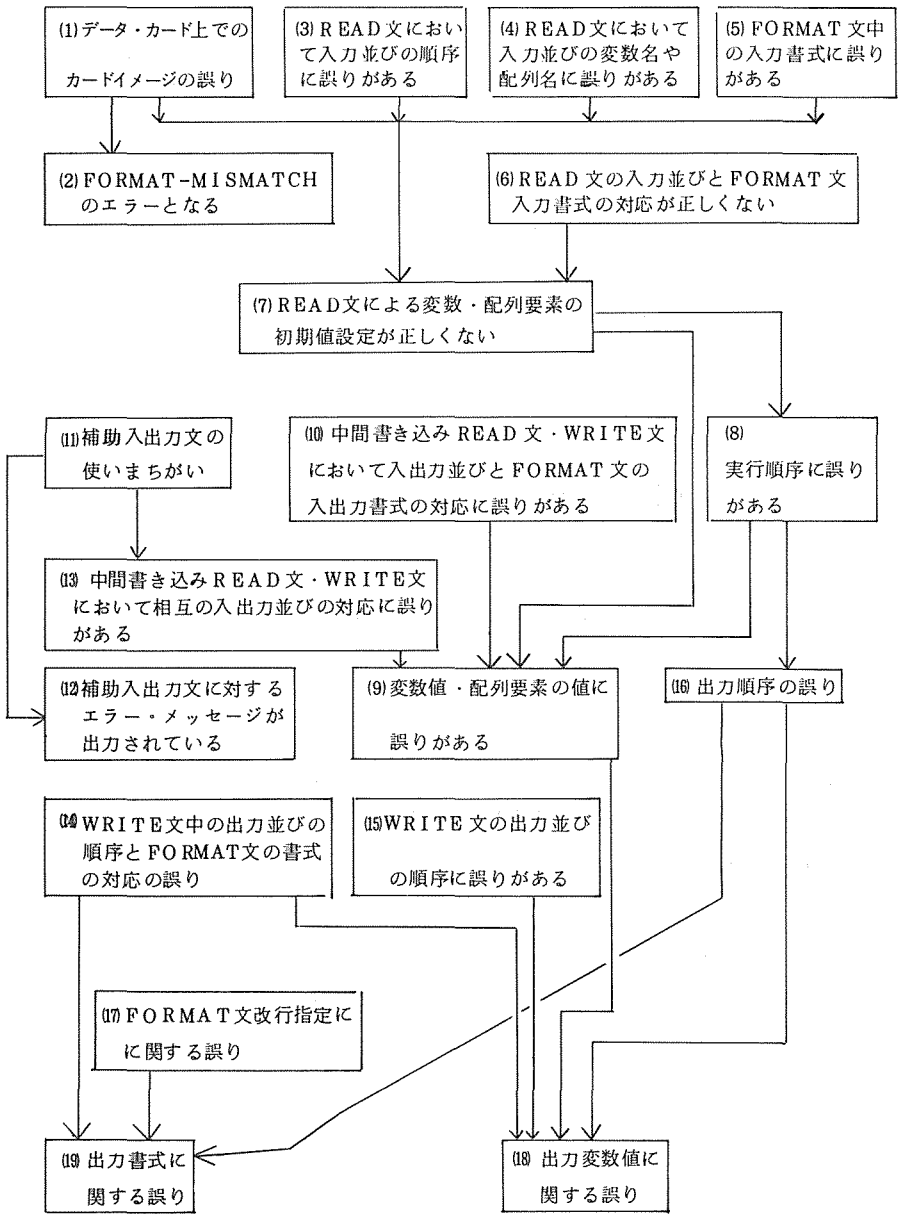

図 5.8　入出力文の誤りの因果関係

6. ま と め

入出力文は具体的システムに依存する所が大きい。したがって，誤りの多くは個々の計算機ごとに異なり，またＦＯＲＴＲＡＮ－７００についても非常に多くのエラーメッセージが出力されるようになっています。しかも，エラー・メッセージの出力されるようなバッグは，たいていその原因を即座に知ることが出来ます。

そのような理由から，本解説では直接にエラー・メッセージの与えられないバッグのみに焦点をしぼって解説しました。必ずしも総ての可能性を尽しているとはいえませんが，これにより入出力文の問題点のいく分かを明らかに出来ていれば幸いです。

引　用　文　献

1） ＪＩＳハンドブック，情報処理，日本規格協会（１９７４）

2） 大阪大学大型計算機センター利用の手引，（１９７５）

3） ＮＥＡＣシリーズ２２００　MODⅣEX/Ⅶ
ＦＯＲＴＲＡＮ７００　文法説明書，ＮＥＣ（日本電気）。

4） ＮＥＡＣシリーズ２２００　MODⅣEX/Ⅶ
ＦＯＲＴＲＡＮ７００　プログラミング説明書，ＮＥＣ（日本電気）