A2. ALGŌLによるプログラムの誤りの発見について

榊原　清（電機試験所）

§1 まえがき

電機試験所計算センターでは，FACŌM230-50のモニタ管理の下で，ALGŌL，FŌRTRAN，CŌBŌLのプログラムをバッチ処理方式により処理しているが，ALGŌLによる使用人口がFŌRTRANの使用人口に徐々に移行する傾向が見えている．（図4,5,6参照，電機試験所計算センタ公報第1号所載の図番のまま）

それは，ALGŌLがFŌRTRANに比し

i）エラ発見に時間がかかる．

ii）プログラム完成までのコンパイル回数が多くなる傾向がある．

iii）FŌRTRANの方が現在の所，共通語として便利である．

という事実による．

私は計算センタ員として，エラ発見に協力することが屢々あり，エラの追求の作業を種々経験するが，以上の2点はALGŌLが持つている種々な性質に原因をもつていると思われる．そこで，ここではエラ原因を追求する作業中に得た経験の中から，プログラムエラがある場合のALGŌLが持つ種々な問題点について，次の3点を中心に説明する．

○エラの発見困難な実例（§2）

○ALGŌLのプログラムエラの特徴（§3），（FŌRTRANとの比較）

○対　策（§4）

なお，我々が用いているALGŌLの金物表現を参考のため文末附表1に示す．

又，ここで問題としているのは，モニタ管理の下でバッチ処理したときの問題であるから，ALGŌLを会語言語として用いる場合，その処理単位ごとにエラチェックがされてプログラムが進められることになるから，エラ発見の困難性については大分状態が変る．その問題については，ここでは触れない．

§2 エラ発見の困難な実例

以下の数例は何れも実際に出会つたプログラムを多少変更して，プログラム上の誤りが判然とするように小規模にしたもので，従つてプログラム上の意味はない．原因が判つてしまえばつまらぬエラであるが，発見の困難性は意外に大きい．しかもプログラムによつては重大な結果を惹き起すエラである．

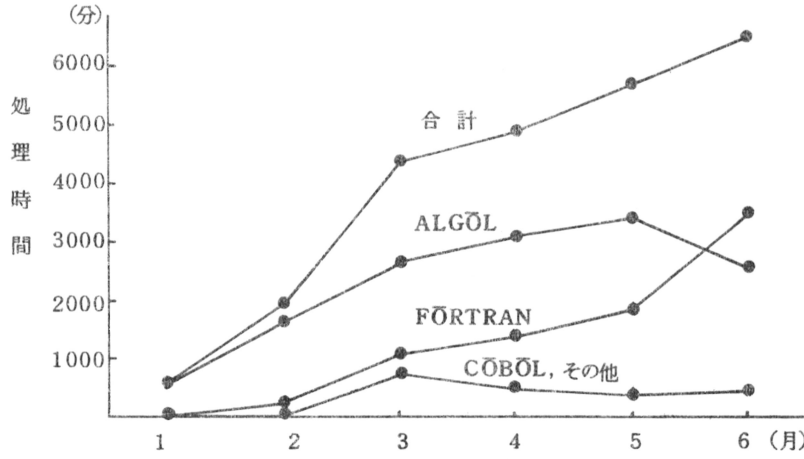


図4 月別処理時間（42. 1～6）

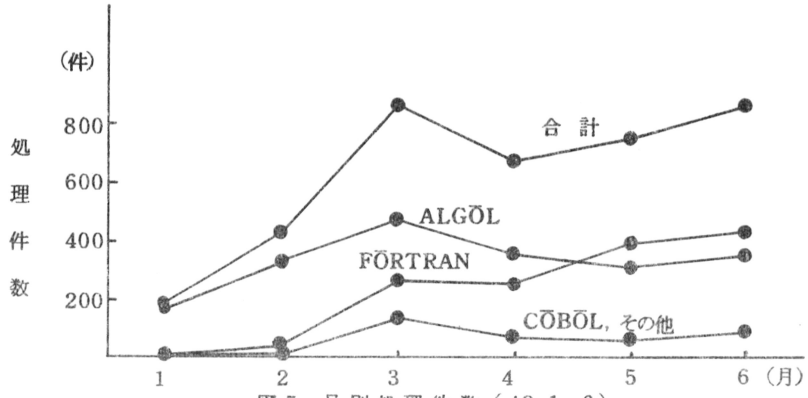


図5 月別処理件数（42. 1～6）

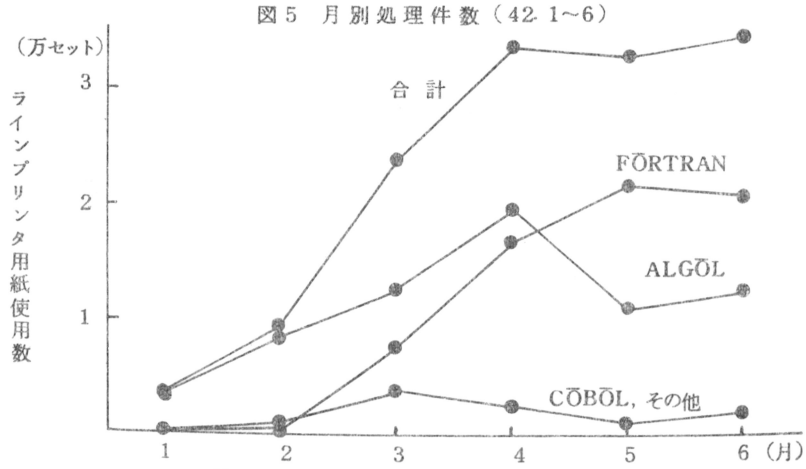


図6 月別ラインプリンタ用紙使用数（42. 1～6）

2.1 'END の後のセミコロンに原因するもの.

'END の後のセミコロンについての文法は，ALGOLのプログラムにおいて種々の困つた現象を惹き起す.

2.1.1 ブロックの構成が崩れる.（1）

```
'BEGIN
'REAL S,D,P,Q,,
S=P*Q,,D=P/Q,,
'BEGIN
'REAL S1,D1,P1,Q1,,
S1=P1*Q1,,D1=P1/Q1,,
'END
'BEGIN
'REAL A,,
A=S+D,,
ER1001      NOT DECLARED      A
'END
```

図2－1

図2－1において，上から7行目の 'END にセミコロンがないため，'BEGIN REALA.,がコメント化してしまつたため「Aが未宣言」となつたエラである.

このためブロックの構成が崩れているのだが，文法上，「どの'BEGINと'ENDが対応せねばならぬ」という規制はないから，最後の 'END は一つ余つてしまう.しかし，この例の場合は……'END'ENDとなつて終るプログラムなので，コンパイラは最後の'END の一つ前の 'END が終りの'ENDと判断する外なく，'END は一つ余されたままエラを出し得ない.従つて「未宣言」というエラからブロックの崩れを診断しなければならない.

2.1.2 ブロックの構成が崩れる.（2）

図2－2において，初めのIFステートメントの中の'END にセミコロンがない.従つてこの場合は'IF T' LS 0.36' THEN'BEGIN K(18)=K(16)+1.,までがコメント化してしまつている.所で前例と相違する点は，前例では'END'ENDと end が続いていたので最後の'END が無視されても表面上は何の影響もなかつたが，この例では，最後の'ENDとその前の'END の間にL6というレーベルが立つているプログラムなので，L6というレーベルは定義されない.

このエラも，プログラムが途中で終つてしまつたことから，'BEGINが足りないか，又

```
'BEGIN'COMMENT K.SAKAKIBARA 'IF ST.,
'ARRAY K(1..20).,
'REAL T.,
T=1.0.,
'IF T 'LS 0.25 'THEN 'BEGIN
K(17)=K(17)+1.,'GO TO L6 'END
'IF T 'LS 0.36 'THEN 'BEGIN
K(18)=K(16)+1.,'GO TO L6'END
ER1002      UNDEFINED LABEL   L6
ER1002      UNDEFINED LABEL   L6
```

図２－２

は 'END が多く入りすぎたかのどちらかの場合であることに気付き，重点的に 'BEGIN と 'END の部分をチェックすると，コメント化に気付く．

このエラも初めて出会つたときには，何故途中でコンパイルが打切られるか，原因が分らずに相当の時間を費した．

以上の２例は，一見何の関係もないエラメッセージから，エラを発見せねばならず，又，普通，上の２例の如き単純なプログラムであることは少ないから，非常に発見困難である．このようなエラを出す原因は，むしろコンパィラ作成者やプログラマに負わせるべきではなく，'BEGIN のようなプログラムの骨組を形成する要素をコメントとして書いてもよいという文法に責任があると思われる．

'BEGIN がコメントの中にも書けるという利益と，一寸とした不注意のためにプログラムの構成全体が崩れてしまうという不利益を考えれば得失は明らかであろう．

2.1.3　実行ステートメントが無効になる．

図２－３のプログラムの計算結果が図２－４である．

図２－３のプログラムは文法上の誤りは発見されないので計算が実行され，答が出た．

しかし，変数Aの値は３．０となるべき答が“０”となつた．これは 'END　A=3.0.,としたからである．

図２－５の例は前例より少々性質の悪い例であるが

{ ○外側のブロックで変数Aは 'INTEGER とした．
○内側のブロックで変数Aは 'REALとした．しかしこの 'REAL宣言は 'END 'REAL A.,としたため無効．

従つてB=A+Aの計算結果は２．０となつた．

```
'BEGIN
  'REAL A,B,C,S,D;
  'BEGIN
  'REAL S1,D1,A1,B1;
  A1=1.0;B1=2.0;
  S1=A1+B1;
  D1=A1-B1;
  PRINT(A1);PRINT(B1);CRLF;
  PRINT(S1);PRINT(D1); CRLF;CRLF;
  'END
  A=3.0;B=5.0;
  S=A+B;
  D=A-B;
  PRINT(A);PRINT(B);CRLF;
  PRINT(S);PRINT(D);
'END
```

図２－３

```
.100000'+01 .200000'+01
.300000'+01-.100000'+01

.000000'-77 .500000'+01
.500000'+01-.500000'+01
```

図２－４

```
'BEGIN 'COMMENT K.SAKAKIBARA INVALID BLOCK;
  'INTEGER A;'REAL B;
  'BEGIN
   PRINTSTRING(''INVALID BLOCK'');
  'END
  'BEGIN
  'REAL A;
  A=1.4;B=A+A;
  PRINT(B);
  'END
```

図２－５

この2例は，プログラムを非常に単純化したのと，答が分つているのでよいが，これが巨大なプログラムで，且，答が予測されない場合，又，直接Aの値が印刷されず他の計算に用いられる場合など，完全に見逃がしてしまうであろう．

```
INVALID BLOCK  200000'+01
```

図２－６

なお，２，3例あげれば

'END'IFD'GQ0'THEN A=B.,A=C.,

の如きプログラムでは，A＝Cが必ず実行されるので，A＝BとA＝Cの計算結果があまり数値の上で相違がないと予想しているとき（実際は相違するかも知れぬ）屢々気づかずに長期間そのままのプログラムで計算が続行されてしまうかも知れぬ．

以上のように，実行命令がコメント化するとき，エラメッセージは出ず，しかも屢々重大なエラを惹き起す．

2.2 string quoteによるエラ（金物表現は‘も’も ''である）

```
'BEGIN 'COMMENT STRING NO BALANCE.;
    PRINTSTRING(''ABC'DE'').;
  PRINTSTRING(''FG').;
  PRINTSTRING(''HI'').;
ER0701    MISSING DELIMITER
ER0701    MISSING DELIMITER
 'END
$  DEVICE
ER0007    $ CARD COMMING
```

図２－７

図２－７において''FG''とすべきを'を一個落したためstringは''FG'PRINTSTRING（''となり，次のstring''H''がH'' となつて意味不明となつたためエラが出ている．これはたまたま次の命令がPRINTSTRING 命令であつたので，エラの判定が簡単なのであつて，１つのPRINTSTRING命令と次のPRINTSTRING 命令とが仮に数100～1000 ステップ離れていれば，今のようなエラで，その間数1000 ステップの全部がstring と化し，表面上，エラメッセージは，

```
PRINTSTRING(''HI'').,
FR0701 MISSING DELIMITER
```

と出る．このエラメッセージのすぐ上のステートメントにエラは考えられず，これから数1000ステップ前の'の落しを発見するのは困難であろう．

string quoteの落しの例は，次のような場合もある．

PRINTSTRING(''Y=')とあつてこれからプログラムの最後までstringがないとき，我々のコンパィラの場合，「コントロールカードが来た」というエラメッセージが出る．この場合なども種々のエラ経験を積まない限り，何故コントロールカードが来るか分らぬため発見困難である．

2.3 不正確なエラメッセージの洪水によるもの．

ALGŌLでは不正確なエラメッセージの洪水の中に溺れてしまつて正確にエラを検出し得ないことが多い．

図2－8に示す如く，コンパィラがプログラム上のエラを全部捜し出そうとすると，エラの洪水が出来易い．

しかも，これらは必ずしも正確にエラを指摘しているとは云い難い．

FŌRTRAN でもALGŌLでもエラが多量に出るのは，宣言命令に関係する．しかし，ALGŌLでは宣言中には，「手続き」の宣言も許されており，又，'BEGIN～'ENDでくくれば，プログラム上どこに宣言を置いてもよい．

この便利さは一度このプログラムの骨組を崩すようなエラを生ずると，相互の位置関係が崩れて，殆んどのステートメントが文法上許されないことになり易い．図1－8の例は，コンパィラがどのような判断の下にエラメッセージを出しているか理解に苦しむ所であるが，初めの「手続き」の宣言を'INTEGER'PRŌCEDURE TWICE(L).，とすべき所を'INTEGER'FUNCTION TWICE(L).，としたために「手続き」の宣言と解釈されず，NCTIONTWICE(L).，なる「手続き」のCALL命令が入つたと解釈され，（このような解釈が正当であるかどうかは別とする）既に主プログラムが始まつていると解釈したので以下の宣言から全部文法違反とされている．このように一部にエラがあつても他の部分の解釈が正確に出来にくくなる性質をALGŌLは持つている．（§2でもこの点に触れる．）従つて不正確なエラメッセージの洪水を生じ易く，又，そのために本当のエラがその不正確なエラメッセージの洪水の中に埋まつて発見困難になる．

我々はALGŌLコンパィラ作成者に対し，出来るだけエラをプログラムの最後まで検出して欲しいと注文を出したが，そのエラの種類によつてはこの注文は有効であつたが，余りにエラの多いものは結局は，他のエラに影響されて出た不正確なエラメッセージであることが多いので，一番最初に指摘されたエラを修正してから後のエラについては考えるということになつているのが現実である．従つてこのような事実がALGŌLにおけるコンパィル回数の

```
'BEGIN

  'INTEGER 'FUNCTION TWICE(L)..

ER0704      SYLLABLE ERROR IN DELIMITER
ER2022      INVALID DELIMITER COMING IN MAIN DIAGRAMMER
ER1001      NOT DECLARED      NCTIONTWICE
  'INTEGER L..

ER2003      INVALID DELIMITER COMING IN MAIN DIAGRAMMER
  TWICE=2*L..

ER1001      NOT DECLARED      TWICE
ER3101      ARITHMETIC EXPRESSION EXPECTED
  'PROCEDURE WRITE(FUNC,J)..

ER2006      INVALID DELIMITER COMING IN MAIN DIAGRAMMER
ER1001      NOT DECLARED      WRITE
  'INTEGER FUNCTION FUNC..

ER2003      INVALID DELIMITER COMING IN MAIN DIAGRAMMER
  'INTEGER J..

ER2003      INVALID DELIMITER COMING IN MAIN DIAGRAMMER
  'BEGIN 'INTEGER K..

  K=FUNC(J)..

ER3005      MUST BE ARRAY OR PROCEDURE IDENTIFIER
ER3001      MUST BE VARIABLE
ER3101      ARITHMETIC EXPRESSION EXPECTED
  PRINT(K)..

  'END..

  'FOR I=1 'STEP 1 'UNTIL 100 'DO

ER1001      NOT DECLARED      I
ER3101      ARITHMETIC EXPRESSION EXPECTED
  WRITE(TWICE,I)..

ER1001      NOT DECLARED      WRITE
  'END

ER1002      UNDEFINED LABEL   I
ER1002      UNDEFINED LABEL   TWICE
ER1002      UNDEFINED LABEL   J
ER1002      UNDEFINED LABEL   FUNC
ER1002      UNDEFINED LABEL   L
```

第２－８

増大の原因である．

その他，発見困難なエラをその原因のみあげてみると，

i) 指定したarray の上下限を越えて用いる．

ii) dynamic arrayでその上下限を表わす式の値が決まつていない．例えば，

'ARRAY A(1…N) でNの値が決まつていない．

iii）コンパィル上の種々の棚のあふれ.

等である.

§3 ALGOLのプログラムエラの特徴

§2でのべたエラは，ALGOL文法の一部の欠陥から来るものと考えられるが，ここではALGOLの一般的な性質から来るエラの特徴についてのべる.

3.1 'BEGIN～'ENDの構造

ALGOLはこの'BEGIN～'END による所謂「Phrasestructure」をもつて構成されている.従つて，もしこの'BEGIN,'END が1つでも欠けると全体のプログラム構成が崩れてしまう.（§1.1 の例もその1例）

プログラム全体の構成が，骨格に相当するものと，小骨に相当するものと区別なく，1つでも欠けると全体が崩れてしまう構造は非常に困つた性質である.

従つてコンパィルは或判断限界を生じて，それ以上翻訳すれば§2.3でのべた如く不正確なエラを出すことになる. FORTRAN にはこのように全体のプログラムの組織を1ステートメントで崩すようなものはない.

しかも，1ステートメントづつ独立し，プログラム単位ごとに独立している.従つてプログラム全体は先ずプログラム単位ごとに完成し，（プログラム全体の中の或プログラム単位にエラがあつても，他のプログラム単位には影響を与えない）又，1ステートメントづつ完成する.

ALGOLはこれに比し，上にのべたような性質から，エラがあれば，途中でコンパィル中止か，不正確なエラメッセージの出現覚悟の上でコンパィルすることになる.いずれにしてもコンパィル回数を増さなければ，全プログラムのエラは発見出来にくい.

3.2 セミコロンの区切り

ALGOLが紙テープをベースに考えられたため，セミコロンが重要な役目をもたせられたものと考えられるが，セミコロンのみに文の区切りの役をさせているため，これを何等かの原因で落すとき，次のステートメント，又はブロック等と互いに影響し合つて，コンパィラは処理の判断が出来なくなり易い.

次に種々の場合について考えると

3.2.1 ステートメントの区切りに用いられるとき.

この区切りを落すと，次からのステートメントと互いに影響し合つて，コンパィラは処理停止，又は不正確な，又は分りにくいエラメッセージを出す.

3.2.2 ブロックの終止のための 'END の後に用いられるとき.

この区切りを落したときのエラについては§2にのべた.

3.2.3 「手続き」の区切りに用いられるとき．

「手続き」の本体が1命令であるとき，「'BEGIN,'END は省略してよい」という規則は，'BEGIN,'END の代りを一つのセミコロンが代理するということと考えられる．例えば，図3－1のように，'PROCEDURE MATOUT(DELF,TW,N)．，とその本体の間にEJECT．，という実行命令がたまたま入り込んでしまうと，そこで「手続き」は終りとなつて，次の'BEGIN からのこの「手続き」の本体は主プログラムと解釈されることになつてしまう．このようなエラは，従つて，上の3.2.1，3.2.2 の場合と相違して，セミコロンの落しではなく，セミコロンが誤つて重要な役目を演じてしまつた例である．

次のような場合も考えられる．

……'END'REAL'PROCEDURE SUM1．,SUM1=A1+B1．,
'REAL'PROCEDURE SUM2．,SUM2=A2+B2．,

この場合，SUM1=A1+B1 は主プログラムと解され，従つて「手続き」SUM2 の宣言は無効になる．これは，'END の後のセミコロンと「手続き」の本体をセミコロンのみで区切つてよいという2つのことから発生したエラで，不正確なエラメッセージしか出し得ない．

FORTRAN ならば，1ステートメントづつの区切りはCONTINUATION記号がなければ，区切りと見做すので，区切りを忘れて他のステートメントに影響を与えることはない．又，FORTRAN のように1ステートメントが短い言語にとつてステートメントが続くということはそれ程多くはないのでCONTINUATION記号の忘れはそれ程多くないし，エラが直接指示されるので発見は易しい．

以上，結論的に云えることはALGOLはFORTRAN に比し「そのエラが他の部分に影響を与え易い」ことであり，それが又，エラ発見の困難な一つの原因である．

これはALGOL の句構造に大きな原因をもつと考えられるが，今一つ大きな原因は「プログラムの冗長度が少ないこと」によつて他から影響を受け易い原因を持つていると云えよう．

冗長度について種々考えてみると，

イ) FORTRAN ならば，副プログラムの表示は，

SUBROUTINE（又はFUNCTION）
RETURN
END

の3つで構成されるがALGOL では

'PROCEDURE（又は<TYPE><'PROCEDURE>）

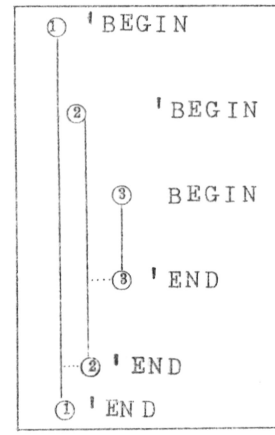
のみであつて他は，'PROCEDURE 独自の表現がないため，他のプログラム部分と混同され易い．（図2－8，図3－1）

ロ) 又，'BEGIN,'END は，どの組合せでも出来てしまう．例えば，次例において

```
        'BEGIN
          'PROCEDURE MATOUT (DELF,TW,N)..
          'INTEGER N..
          'ARRAY DELF..
          'INTEGER'ARRAY TW..
          EJECT..
          'BEGIN
            'INTEGER M,MM,II..
            'ARRAY A(1..5,1..5,1..5)..
            'FOR M=1 'STEP 4 'UNTIL N 'DO
ER1001        NOT DECLARED       N
ER3101        ARITHMETIC EXPRESSION EXPECTED
              MM=M+3..
            'IF MM 'LQ N 'THEN 'GO TO M10.. MM=N..   M10..
ER3101        ARITHMETIC EXPRESSION EXPECTED
ER3001        MUST BE VARIABLE
ER3101        ARITHMETIC EXPRESSION EXPECTED
            'FOR I=M,I+1 'WHILE MM 'GQ N 'DO
ER1001        NOT DECLARED       I
ER1001        NOT DECLARED       I
ER3101        ARITHMETIC EXPRESSION EXPECTED
              OUTPUT(1,200,$4B,4(2B,''F(Y+TE I)'',5B,''F(Y-TE I)'',2B),/$,
                     TW(I))..
ER1001        NOT DECLARED       TW
ER0702        MUST BE IDENTIFIER
ER3101        ARITHMETIC EXPRESSION EXPECTED
            'FOR I=1,I+1 'WHILE N 'GQ 0 'DO
ER3001        MUST BE VARIABLE
ER3101        ARITHMETIC EXPRESSION EXPECTED
ER3101        ARITHMETIC EXPRESSION EXPECTED
            'FOR II=M,II=1 'WHILE MM 'GQ N 'DO 'BEGIN
ER2546        INVALID DELIMITER COMING IN FOR STATEMENT
              B=2*(MM-M+1)..
ER1001        NOT DECLARED
              OUTPUT(3,200,$3D,B(BZ,7D),/$,TW(I),A(I,II,1),A(I,II,2))..

ER0574        INVALID CHARACTER COMBINATION IN FORMAT STRING
ER0534        INVALID CHARACTER COMBINATION IN FORMAT STRING
ER1001        NOT DECLARED       TW
ER0702        MUST BE IDENTIFIER
ER3101        ARITHMETIC EXPRESSION EXPECTED
ER3101        ARITHMETIC EXPRESSION EXPECTED
ER3101        ARITHMETIC EXPRESSION EXPECTED
                                 'END..
              EJECT..
            'END..
            'PROCEDURE MATO1(A,N,L,CRLFF,LABEL1)..
ER2006        INVALID DELIMITER COMING IN MAIN DIAGRAMMER
ER1001        NOT DECLARED       MATO1
            'STRING CRLFF,LABEL1..
ER2020        INVALID DELIMITER COMING IN MAIN DIAGRAMMER
            'ARRAY A..
ER2005        INVALID DELIMITER COMING IN MAIN DIAGRAMMER
            'INTEGER N,L..
ER2003        INVALID DELIMITER COMING IN MAIN DIAGRAMMER
            'BEGIN
            'END
        'END
ER1002      UNDEFINED LABEL    LABEL1
ER1002      UNDEFINED LABEL    LABEL1
ER1002      UNDEFINED LABEL    CRLFF
ER1002      UNDEFINED LABEL    L
ER1002      UNDEFINED LABEL    A
ER1002      UNDEFINED LABEL    I
ER1002      UNDEFINED LABEL    N
```

図 3－1



①－①，②－②，③－③なる'BEGINと'END が対応するのが正常なのだが，③の'BEGINにクオート'を落したり，又は§1.1でのべた原因で無効になると，対応は①－②，②－③となり，最後の'ENDは余されたままとなる．どの'BEGINと'ENDが対応すべきだという積極的な指示がないためである．

ハ) 'END の後のセミコロンにしても，セミコロンのみにプログラム全体の骨組を崩すような重要な責任を負わすべきではなく，他の幾つかの情報を添えてコンパィラの判断を確実にすべきであろう．（カードベースのALGOLであれば特にカードの特性を利用すべきであろう．）

ニ) 「手続き」を呼ぶのにCALLの如きステートメントがないため，例えば図1－8の例にもある如くTWICE(L).，は手続きを呼ぶステートメントと混同されている．

以上のようにプログラムの冗長度を増すことによりコンパィラは一部分の間違いによつて次から判断停止の状態に追い込まれるような事態を避けることが出来る．特にカードベースであればカラムの特徴をもつと生かせばよいと考える．冗長度を増すことにより，ALGOLでも部分的な完成が出来，又，正確なエラ診断の可能性が増すと考えられる．

その他，ALGOLのプログラムで初心者が侵すエラを参考迄にあげておく．

i) 'END の後のセミコロンの打ち方と間違い．

ii) specificationとdeclaration の意味の混同．

iii) value の使用法の間違い．

iv) read,print 命令以外の入出力命令のformatの間違い．

v) 重複定義（'INTEGERA.,'INTEGER'ARRAY(1.,20).,）
これはFORTRAN使用者に多い．

vi) While形のfor ステートメントを収束せぬような問題に用いる．

§4 対　　策

§3の終りにのべた如く，ALGOLにはプログラマの意志を間違いなくコンパィラに伝達するための情報が欠けている．従つて，何等かの手段でこれを補う必要があり，それには次の2点が考えられる．

i) ALGOL文法を変更する．

ii) 書き方に工夫を加える．

4.1 ALGOL文法を変更する．

コンパィラ言語が共通語という性質をもつている以上文法の変更は容易ではない．その

変更が部分集合になればよいが，方言と化してしまうことは困る．

しかし，ここでは，共通語という性格を一応無視して単に「エラの発見の困難性をなくす」という立場から考えてみる．

4.1.1 'BEGIN～'END に対応番号がつけられるようにする．

例えば'BEGIN1 ～'END1の如くである．

このようにすれば，対応について判然とし，どの'BEGIN，又は'END のバランスが崩れたかは明瞭となり，エラの診断がし易い．

4.1.2 'END の後にコメントを許さず，直後にセミコロンをつける．

セミコロンでない記号は許さない．('ELSE の場合のみを除く，又，もしセミコロンを書き落しても'ELSEが続く他はセミコロンを補つてやつてもよい．)従つて従来の 'END の後のコメントは許さない．

4.1.3 'END の後のコメントは許す場合は，コメントの中にdelmiterを許さない．

このようにすれば'BEGIN のようなプログラムの骨組を形成するような重要な要素をコメント化する心配がなくなる．

4.1.4 「手続き」の本体には必ず'BEGIN，'END をつける．

他のブロックと区別するため'BEGIN P1～'END P1 の如く「手続き」の本体の区切りであることを明確にする．このようにすれば，他のプログラム部分と混同されることを防ぐことが出来，他の部分にエラがあつても，「手続き」の部分だけでも部分的にエラチェックが可能になるであろう．

4.1.5 「手続き」を呼ぶため CALL をつける．

呼び出すステートメント(CALLとは限らない．)をつけて呼ぶことにすることによつて，他との混同を防ぐ．

4.1.6 カードの場合，Stringが2行以上にわたるときは何等かの続き印を書く．

4.1.7 ブロック別，又は「手続き」別のコンパィル．

大きなプログラムは，FŌRTRAN と同じく，分割して別コンパィルとすべきである．別コンパィルにすれば部分的完成が可能となる．但し，その際，FŌRTRAN における CŌMMŌN ステートメントのようなものがないと不便ではないかと思われる．

4.2 書き方に工夫を加える．

ALGŌL はプログラムの書き方によりかなり分り易い，又，従つて間違いのないプログラムを書くことが出来る．

4.2.1 'BEGIN と'ENDにはレーベルをつける．

4.2.2 カードの場合，対応する'BEGIN と'ENDは同じカラムから書く．

4.2.3 'END にはその直後に必ずセミコロンをつける．コメントは'CŌMMENT による．

4.2.4　出来るだけ見易いようにプログラムを書く．

ステートメントを1行に幾つも書くことは間違いの原因となる．

4.2.5　IFステートメントなどもIF～THEN～ELSE を長々とつなげないようにプログラムを工夫する．

4.2.6　ブロックはなるべく単純に整理すること．

§5　まとめ

以上にALGOLの特徴的なエラと実例についてのべ，その原因を次の2点

i)　ALGOL文法の部分的欠陥

ii)　ALGOL言語の性質から来るもの．

から説明し，その対策についても私の考えをのべた．

しかし，その原因を除去しようとすると，共通語としての性格という壁にあたる．ALGOLには言語としての自由度を拡大させるような文法修正の動きは常にある（ALGOL 6X のような）が，実際上の使用経験からする修正意見，むしろ「不便でもよいから規制する」とか「冗長度を増大せよ」というような意見を吸収する努力は今迄にされていないと思われる．

種々の部分集合や方言が発生するのも，一つの原因はここにあるのではないかと考える．使用経験からする修正意見を部分集合，又は方言とせずに良い意見をどしどし文法に吸収してゆく道を作るべきだと考える．

ALGOL文法の作成，又はその修正にあたられる人々が

i)　現在計算機を用いる人々が最早，プログラマと呼ばれるようなプログラムの専門家のみでないこと．

ii)　専門家でない人々にも容易に分るようなエラメッセージが要求されていること．

iii)　プログラムの専門家でない人間は，表現機能の拡大より，簡単な基本的な機能を組合せて種々のプログラムが出来ること，完成迄の時間の早さ，エラ診断の確実さ等を望んでいる．

ということを充分認識されるよう望みたい．

我々の計算センタでも経験する所であるが，計算機をソロバン代りと考える人にとつては，「手続き」の文法さえ面倒で，もし間違つて用いると時間を消費するということから用いたがらぬ人々も居る．これは少々極端ではあるが，ALGOLの表現機能の拡大の努力も必要と思われるが，実用的に工夫を文法に取入れることが現在のALGOLにとつてもつと必要であると考える．

終りに，本文をまとめるにあたり，終始御鞭撻戴いた戸田電気試験所計算センタ長に感謝する．

附表1

FACOM-230-50/60-70 ALGOLの金物の表現

ALGOL symbol	representation
go to	'GOTOまたは 'GO'TO
if	'IF
then	'THEN
else	'ELSE
for	'FOR
do	'DO
step	'STEP
until	'UNTIL
while	'WHILE
comment	'COMMENT
begin	'BEGIN
end	'END
boolean	'BOOLEAN
integer	'INTEGER
real	'REAL
array	'ARRAY
switch	'SWITCH
procedure	'PROCEDURE
string	'STRING
label	'LABEL
value	'VALUE
<	'LS
≦	'LQ
=	'FQL
≠	'NQ
≧	'GQ

ALGOL symbol	representation
>	'GR
≡	'EQV
⊃	'IMP
∧	'AND
∨	'OR
¬	'NOT
+	+
−	−
×	*
/	/
÷	//
↑	**
,	,
.	.
10	'
:	..
;	.,
:=	=
(	(
)	)
[	(/または(
]	/)または)
‘	''
’	''
¥（新設）	¥
␣	
true	'TRUE
false	'FALSE

本 PDF ファイルは 1968 年発行の「第 9 回プログラミング一シンポジウム報告集」をスキャンし、項目ごとに整理して、情報処理学会電子図書館「情報学広場」に掲載するものです。

この出版物は情報処理学会への著作権譲渡がなされていませんが、情報処理学会公式 Web サイトの https://www.ipsj.or.jp/topics/Past_reports.html に下記「過去のプログラミング・シンポジウム報告集の利用許諾について」を掲載して、権利者の捜索をおこないました。そのうえで同意をいただいたもの、お申し出のなかったものを掲載しています。

過去のプログラミング・シンポジウム報告集の利用許諾について

情報処理学会発行の出版物著作権は平成 12 年から情報処理学会著作権規程に従い、学会に帰属することになっています。

プログラミング・シンポジウムの報告集は、情報処理学会と設立の事情が異なるため、この改訂がシンポジウム内部で徹底しておらず、情報処理学会の他の出版物が情報学広場 (=情報処理学会電子図書館) で公開されているにも拘らず、古い報告集には公開されていないものが少からずありました。

プログラミング・シンポジウムは昭和 59 年に情報処理学会の一部門になりましたが、それ以前の報告集も含め、この度学会の他の出版物と同様の扱いにしたいと考えます。過去のすべての報告集の論文について、著作権者（論文を執筆された故人の相続人）を探し出して利用許諾に関する同意を頂くことは困難ですので、一定期間の権利者捜索の努力をしたうえで、著作権者が見つからない場合も論文を情報学広場に掲載させていただきたいと思います。その後、著作権者が発見され、情報学広場への掲載の継続に同意が得られなかった場合には、当該論文については、掲載を停止致します。

この措置にご意見のある方は、プログラミング・シンポジウムの辻尚史運営委員長 (tsuji@math.s.chiba-u.ac.jp) までお申し出ください。

加えて、著作権者について情報をお持ちの方は事務局まで情報をお寄せくださいますようお願い申し上げます。

期間：2020 年 12 月 18 日 〜 2021 年 3 月 19 日
掲載日：2020 年 12 月 18 日

プログラミング・シンポジウム委員会

情報処理学会著作権規程
https://www.ipsj.or.jp/copyright/ronbun/copyright.html