## 談　話　室

# 言語とコンパイラの比較

小　島　　惇

### 1. はじめに

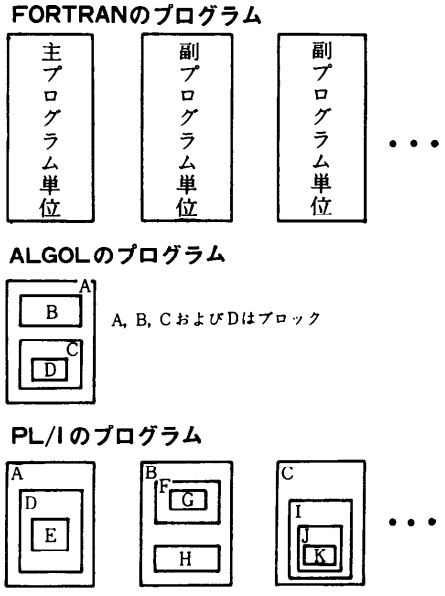
言語とコンパイラの比較という題で談話室に何か書けという話があったとき，言語とコンパイラを比較できるものだろうかと戸まどいを感じた．編集者のお話によれば，かって PL/I を NPL と呼んでいたころ，私が NPL，ALGOL および FORTRAN の機能の比較を試みたことを覚えておられて，そのようなことをこの特集号向きに変形して書けということであった．それにしても，5，6 年前に較べて，これらの言語もずい分変ったものである．JIS FORTRAN が制定されたころには，すでに水準 7000 をサブセットとするような FORTRAN が現われていたが，現在ちよっとした FORTRAN は，ほとんど水準 7000 以上の機能を持っていて，水準 8000 を作る必要を感じさせるほどである．一方，ALGOL にも FORTRAN と同じような傾向があるとともに，ALGOL 系の新しい言語として，ALGOL 68 や ALGOL N が発表されている．PL/I にいたっては，まだ標準化されていないから無理もないが，NPL から現在の PL/I への化け方は大変なものであるし，IBM の language specification も改版の度に相当に変わり，コンパイラ設計に携わっている人々を惑わせている．また，PL/I と称する一見 PL/I らしきコンパイラが多く開発されている[1]．ともかく，この談話室では，JIS FORTRAN 水準 7000，JIS ALGOL 水準 7000 および，70，ならびに，PL/I language specification (IBM FORM Y-33-6003-1) を参考にしながら，コンパイラ的立場から，思いつくままに比較を行なって見よう．

### 2. プログラムの定義

構文上，ALGOL では，

＜プログラム＞::=＜ブロック＞|＜複合文＞，

FORTRAN では主プログラム単位といくつかの（0でもよい）副プログラム単位，PL/I では一つ以上の external procedure からなる．このうちの一つは主プログラム単位に当るものでなければならず，IBM では OPTIONS (MAIN) と指定する．external procedure は，それぞれブロック構造になっているから，大まかにいえば，PL/I は ALGOL と FORTRAN を合併したようなものである（**第1図**）．

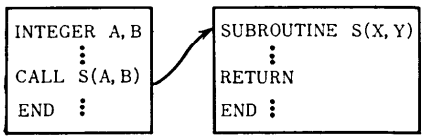



第 1 図

普通各プログラム単位または external procedure は独立にコンパイルされ，オブジェクト・モジュールを結合するのはリンケージ・エジタの仕事であるから，未定義の名前や未確定の性質が残っている．たとえば，FORTRAN の外部手続き名の場合，引数の個

* 日本ビジネスオートメーション(株)第2プログラム部

数や型や順序について，正しいかどうかはコンパイル中は確かめられない（**例 1**）．PL/I では，呼ぶ側のブロックで entry name および引数の属性を宣言することによって，個々の external name を呼ぶ場所でのチェックや実引数の型の自動変換の準備をコンパイル中に行なうようになっている（**例 2**）．




例 1

```
A: PROCEDURE;
   DECLARE B ENTRY,
     SUB ENTRY (ENTRY, FLOAT, LABEL);
   CALL SUB (B, SQRT(R), L1);
L1:
   END A;
```

例 2

回帰的呼び出しについては，FORTRAN では全く許されず，ALGOL ではすべての手続きを回帰的に呼び出せることになっている．したがって，ALGOL の手続きの本体における記憶場所の割付けは，動的に手続きが呼び出されるたびに前の状態を保存したまま新たに行ない，戻るたびに現在の割付けを解除して，直前のそれを復活するという方法を取らなければならない．このアルゴリズムは，処理系により，特に計算機の特徴によって異るけれども，普通は回帰的呼び出しを必要としない場合には効率を著しく低下させる原因となる．PL/I では，回帰的呼び出しを必要とする手続きについてだけ，これを指定するようになっている（**例 3**）．

プログラムのオプチマイズ，併行処理，エラーによる割込み処理，バッファリングなどについては，JIS の FORTRAN や ALGOL では，言語のそとの問題として触れていないが，PL/I では言語の中にこれらの機能を組み入れている．たとえば，ブロックごとにオプチマイズの度合を指定するオプション ORDER, REORDER がある．ORDER を指定したときは，共

```
A: PROCEDURE (X, Y) RECURSIVE;
   DECLARE (X, Y, P, Q) FLOAT BINARY;
   CALL A (Y, Q);
   END A;
```

例 3

通式 (common expression) の処理（ある条件の下で，式の共通部分を一度だけ計算し，他はその値を利用すること）を除いては，プログラムの順序に従って実行する．REORDER を指定したときは，ある条件の下では高度のオプチマイズ（たとえば，DO グループの自動的な書き変え）を行なう（**例 4**）．このようなオプチマイズは，FORTRAN コンパイラでもよく行なっているが，その指定は言語のそとの問題として取扱っている．

```
Y: PROCEDURE REORDER;
   DO I=1 TO 10;
   DO J=1 TO 10;
   A(I, J)=B(1+K, J)*C(1+K)
     +D/I+D/K;
   END;
   END;
END Y;
```

⇒

```
Y: PROCEDURE;
   T1=D/K;
   DO I=1 TO 10;
   T2=D/I;
   I1=I+K;
   DO J=1 TO 10;
   A(I, J)=B(I1, K)*
   C(I1)+T2+T1;
   END;
   END;
END Y;
```

例 4

## 3. 変数と型

ALGOL や FORTRAN で型と称していた概念は，PL/I では拡張されて属性 (attribute) と呼ばれ，base, scale, mode, precision, PICTURE の 5 種の属性で算術データの型を，BIT, CHARACTER, length, VARYING, PICTURE の 5 種の属性で string データの型を指定する．**第 1 表**は，FORTRAN の型の指定が他の言語でどのように表わせるかを示したものである．

配列の宣言は，FORTRAN の場合，宣言文の名前の直後に配列宣言子（添字）を書くことによってなされる．特に，DIMENSION 文を用いる必要はない．ALGOL では必らず〈type〉**array** で始まる宣言を用いなければならない．一方，FORTRAN の配列の下限は必らず 1，上限は整合配列を別とすれば，整定

第 1 表

| FORTRAN | ALGOL | PL/I (**) |
|---|---|---|
| INTEGER | **integer** | BINARY FIXED |
| REAL | **real** | BINARY FLOAT |
| DOUBLE | | BINARY FLOAT (*precision*) |
| COMPLEX | | BINARY FLOAT COMPLEX |
| LOGICAL | **Boolean** | BIT (1) |
| (*) | | CHARACTER (*length*) |

注 (*) 文字型
(**) PL/I には，他に多くの変数の型がある．また，計算機によっては，この表とは異なる対応をする場合もある．

数でなければならないが，ALGOL では上下限とも式でよい．PL/I では，配列宣言子を宣言文の名前の直後に書けばよいことは FORTRAN と似ているが，その ( ) の中の書き方は ALGOL と同じように式を用いることができる (**例 5**)．また，記憶場所への割付け方は FORTRAN がたて形であるのに対し，ALGOL と PL/I はよこ形である．

| FORTRAN | ALGOL | PL/I |
|---|---|---|
| REAL A(10) | **array** A[1 : 10] | DECLARE A(10) BINARY; |
| —— | **array** A[I: J×2] | DECLARE A (I:J*2) BINARY |

例 5　配列の宣言の例

PL/I では，いわゆる型を表わす属性として，構造体 (structure) のほか，AREA, ENTRY, EVENT, TASK, LABEL, POINTER, OFFSET など多くのものがあるが，特に比較の意味で挙げる場合のほか，この談話室では話題にせず，これらが PL/I を多目的の絢爛たる言語として，したがって，また問題の多い言語として特徴づけていることだけを附記して，次へ進む．

## 4. 宣言と有効範囲

FORTRAN では，外部手続き名とブロック名はプログラム全体が有効範囲である[(3)10.2.1,7.3(2)]．その他の名前は，規則[(3)10.]を満足していれば，それが現われているプログラム単位が有効範囲である．ALGOL ではブロック構造によって有効範囲が定められる．しかし，これだけではいくつか不便な点が生ずる．たとえば，**例 6** (FORTRAN) の S2 中では，変数Dの有効範囲はプログラム単位 S2 であり，S1 のDのそれは S1 である．もしDが S1 と S2 に渡る有効範囲を持つような効果を出すためには，S1 のプログラムが

```
SUBROUTINE S1(X, Y)
COMMON A, B, C, D
      ⋮
END
SUBROUTINE S2(P, Q)
COMMON A, B, C, D
      ⋮
END
```

例 6

動かせなければ，たとえ S2 で A, B, C は使用していなくても COMMON 文は S1 のそれと合わせなければならない．PL/I では，**例 7**. のような手段で，この問題を解決できる．すなわち，EXTERNAL と宣言された名前は，プログラム全体を含む仮想的なブロックがあって，そこで宣言されたかのように考えればよい．この例と同じ効果を FORTRAN で出すためには，名前付共通ブロックを用いることができる (**例 8**)．

```
S1: PROCEDURE;
    DECLARE (I, A) EXTERNAL;
      ⋮
    END S1;
```

例 7

```
SUBROUTINE S1(X, Y)
COMMON /P/I/Q/A
      ⋮
END
SUBROUTINE S2(P, Q)
COMMON /P/I/Q/A
      ⋮
END
```

例 8　COMMON の利用法

また，**例 9** のブロックBで使用する SIN は標準関数であって欲しくとも，現在の ALGOL ではその指示をする手段がない．PL/I では BUILTIN という属性で宣言することによって，この目的を果すことができ

```
begin real A;
  procedure SIN (X);
    begin
      ⋮
    end;
      ⋮
M: begin real B;
      ⋮
    B:=SIN (A);
      ⋮
    end;
      ⋮
    end
```

例 9

```
L : BEGIN; DECLARE A;
    SIN: PROCEDURE (X);
          ⋮
         END;
          ⋮
    M : BEGIN;
         DECLARE B, SIN BUILTIN;
          ⋮
         B=SIN (A);
          ⋮
         END M;
    END L;
```

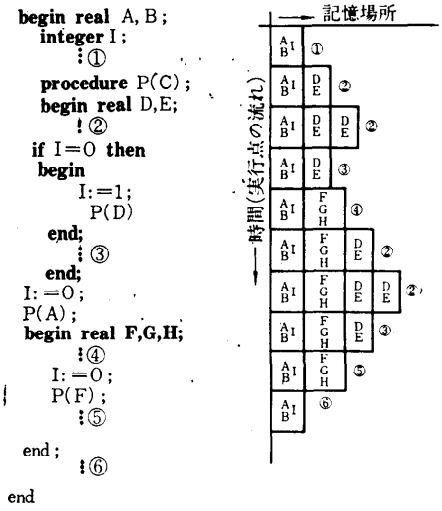
例 10

る（例 10）.

## 5. データと記憶場所の割付け

変数や配列の記憶場所については，FORTRAN では静的に確定しているし，ALGOL では動的な意味で名前が有効である範囲内では確定している．また，変数の占める記憶場所の大きさは，実質的には PL/I も含めて，単純変数（配列や構造体でないもの）については静的に確定する．すなわち，FORTRAN については，変数の占める記憶場所の大きさは型によって定まっている[3)4.3)]．また，配列の上限は整定数しか書けないから，仮引数の場合を別とすれば，原理的には静的に記憶場所を確定することができる．実引数は，たとえ式であっても，その値を置く場所は静的に定まる．したがって，実行時に実引数と仮引数の結合が行なわれるとき，すなわち，その仮引数を含むプログラム単位の実行前に記憶場所は確定する[3)8.3.2,8.4.2)]．整合配列の場合も，その大きさは対応する実引数の配列の大きさを越えることはない[3)7.2.3)]から，整合配列が新たに領域を占有したり，他の変数を侵すことはない．共通ブロックについては，名前つき共通ブロックは同一の記憶場所と大きさを共有しなければならないから，大きさは静的に定まる．無名共通ブロックは，最大のものはコンパイル中には不明であるが，プログラムの実行前に確定する．すなわち，FORTRAN の場合は，理論的には，プログラム単位を結合したときに，すべての記憶場所の割付けは完了し，変数名と記憶場所の対応づけが，部分的に，動的に行なわれる．

ALGOL の場合は，プログラムは一つのブロック構造体であり，実行点があるブロックの入口に達したとき，そのブロックに局所的な変数や配列の記憶場所の割付けが行なわれる．単純変数や上下限が定数である配列については，ブロックごとに相対的な意味では静的な割付けを行なうことができる．上下限が定数以外の式のときには全く動的な割付けを行なうことになるし，手続きは回帰的に呼ばれるから，記憶場所の確定はすべて動的になる（例 11）．ただし，割付けの働き




例 11

は，ブロックの入口と出口でだけ起り，ブロックの途中で起ることはない．実際の処理系にとっては，ワーキング・メモリの有効な利用法や **go to** でブロックを出るときの記憶場所の管理を含めて，最もむずかしいことの一つとなっている．占有型の単純変数や占有型の定数上下限の配列は，本来の占有型の性質から，記憶場所はブロック構造に関係なく割付ければよい．しかし，上下限が定数でない場合は処理が大変むずかしくなる．すなわち占有型の性質[2)5.2.4)]から，新しい配列のうち，添字の値の共通する部分について，配列の要素の値そのものを移動させる必要が出てくる．たとえば，

**own array** $A[m:n]$

において，$t$ 回目にこの宣言を含むブロックに入ったとき $m=-2$，$n=2$ であり，$t+1$ 回目に入ったとき $m=0$，$n=5$ であったとし，記憶場所の先頭が固定されているとすれば，**第 2 図**で示すようになるはずである．

PL/I では，記憶場所の割付け方を，変数の属性として指定することができる属性は次のものである．

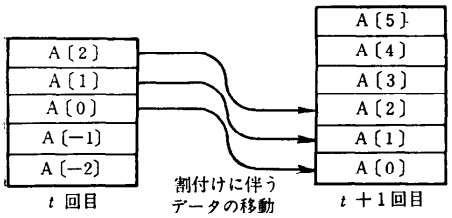
STATIC, AUTOMATIC, CONTROLLED または BASED

ALGOL の静的な占有型の配列や占有型の単純変数の割付けは INTERNAL STATIC と同等であり，

| PL/I | ALGOL | FORTRAN |
|---|---|---|
| DECLARE A STATIC INTERNAL; | own real A | |
| DECLARE A STATIC EXTERNAL; | — | COMMON/C/A |
| DECLARE A AUTOMATIC; | real A | |

例 12

EXTERNAL STATIC は FORTRAN の名前つき共通ブロックを利用した変数に似ている．なお，PL/I の STATIC な配列は上下限が定数でなければならないので，ALGOL の場合のような困難さは存在しない．AUTOMATIC な割付けは ALGOL の非占有型の変数




第 2 図

や配列と同等と考えてよい（**例 12**）．CONTROLLED と BASED は，関連する変数名の有効範囲と記憶場所の割付けの確定とが対応せず，割付けをプログラマが完全に制御するもので，ALGOL や FORTRAN には無い概念である．割付けと解除は ALLOCATE 文または LOCATE 文と FREE 文によって行なわれる．CONTROLLED の場合は，同じ変数を何度もつづけて割付けると，各世代はスタックされ，常に最も新しく割付けられた記憶場所がその変数によって参照される．BASED の場合は，スタックされることなく，関連する locator 変数（たとえば pointer 変数）によって参照される．特に BASED はバッファー・ハンドリングやリスト処理に力を発揮するものであるが，これ以上は説明しない．

## 6. 名前の識別

名前がその宣言以外の所で現われたとき，どのような手順で識別するかを比較する．

ALGOL の場合は，

手順 1. その名前が現われた場所を含む最も内側のブロックを求める．

手順 2. このブロックで同じ名前が宣言されていれば，この宣言がその名前の宣言である（識別おわり）．

手順 3. すぐ外側のブロックを求める．あれば手順 2. へ．なければ手順 4. へ．

手順 4. その名前が標準関数や入力手続きなど宣言なしで使用できる名前と一致すれば，その意味で使われたものとみなす（識別おわり）．一致するものが無ければ誤り．

となる．名前の使用方法が正しいかどうかは，別問題である．FORTRAN の場合は，文字の列が英字名であるか，文を識別するための文字の列であるかの判断や，英字名である場合に FORTRAN が規定する 8 種の類のどれに所属するかを決めるために，宣言文ばかりでなく文脈が問題となる[3)10-1)]．また，型宣言文は，英字名の属性のうち，型だけを宣言するものであって，他の属性は，他の宣言文や文脈上から判断しなければならない．PL/I の場合は，大まかにいえば，次の手順で識別する．

手順 1. その名前が現われた場所を含む最も内側のブロックを求める．

手順 2. このブロックに内部である部分（このブロックに含まれるすべてのブロックを除いた部分）に同じ名前が明示宣言されていれば，この宣言がその名前の宣言である（識別おわり）．

手順 3. すぐ外側のブロックを求める．あれば手順 2. へ．なければ手順 4. へ．

手順 4. 名前の属性が文脈上識別できるかどうか調べる．できれば，文脈宣言が行なわれる（識別おわり）．

手順 5. 暗黙 (implicit) 宣言が行なわれる（識別おわり）．

注意すべきことは，external procedure を呼ぶ場合には，それを必らず呼ぶ方のブロックで entry name として宣言しなければならないこと，および，FORTRAN の基本外部関数に相当するものは，PL/I では組込み関数であることである．組込み関数は entry

```
      ⋮
EXTERNAL A,B     この例では，Bは外部関数であることがわかるが，Aは外部関数かサブルーチンか分類できない．
      ⋮
CALL S (A,B)
      ⋮
X−Y+B (Z)
      ⋮
```

例 13 名前を類に分類できない FORTRAN プログラム

```
begin real A,B; integer C,D;
  procedure P (X,Y);
    begin real E,F;
          ⋮
      E:=F+X;                    E:=F+Xは，Lの
          ⋮                      文でPが呼ばれたと
    end;                         きは E:=F+A
L: P (A,B);                      Mの文で呼ばれたと
M: P (C,D);                      きは E:=F+C
    ⋮                            となる.
  end
```

例 14　型が定まらない ALGOL プログラム

```
        FORTRAN の場合
    REAL FUNCTION F(A)
    REAL A,B,C
        ⋮
      F=B**2-4*A*C
      RETURN
    END
        PL/I の場合
F: PROCEDURE(A)RETURNS (FLOAT
   BINARY)
        ⋮
   RETURN (B**2-4*A*C);
   END;
```

FORTRAN の例では，Fは関数名と変数名との二つの意味で使用されている．ALGOL も同様である．PL/I は，この例で見るとおり，そのようなことはない．

例 15　関数の値の定義

name としての宣言なしに使用でき，その識別は手順 4. の中で行なわれることになっている．PL/I の識別に関する手順は曖昧を残さない点で，FORTRAN よりすぐれている．**例 13** から**例 15** は，すこし具合の悪い場合をとり上げてみたものである．

## 7. 式の評価

オペレータの優先順位比較表を**第 2 表**に示す．1次子の評価の順序はいずれの言語も規定していない．式の評価については，FORTRAN が算術の正常な法則に従う限り，評価の順序は原則として自由であるのに対し，ALGOL は一般に左から右へ向うとし，PL/I は第1優先順位のものは右から左へ，他は左から右へ向うとしている．たとえば，A↑B↑C は $A^{B^C}$ であるが，PL/I の A＊＊B＊＊C は ABC を意味する．また，FORTRAN では＊や＋は結合則も交換則も成り立つとしていると解釈できるが[3)6.4]，PL/I では＊や＋は結合則は成り立たないと決めている[5)p37]．このような解釈の相違はコンパイラの設計，ひいては実行の効率に影響をおよぼすことがある．また，PL/I で共通式(common expression)という概念を，FORTRAN で文節 (basic block) という概念をとり上げているが，ともにコンパイラ設計におけるオプチマイズに関連する重要な要素であることに注意されたい．

第 2 表　オペレータの優先順位

| 順位 | FORTRAN | ALGOL | PL/I |
|---|---|---|---|
| 1 | ＊＊ | ↑ | ¬, ＊＊, prefix+, prefix− |
| 2 | ＊, ／ | ×, ／, ÷ | ＊, ／ |
| 3 | +, − | +, − | infix+, infix− |
| 4 | | | \|\| |
| 5 | .LT., .LE.,⋯,.GE. | <, ≦, ⋯, ≠ | >=, >, ⋯, = |
| 6 | .NOT. | ¬ | ¬ |
| 7 | .AND. | ∧ | & |
| 8 | .OR. | ∨ | \| |
| 9 | | ⊃ | |
| 10 | | ≡ | |

## 8. 入出力

FORTRAN と ALGOL は，基本的に個々のデータの入出力を取扱うことを主体としている．また，外部情報（外部媒体上のデータ）は sequential に並んでいるものと考えている．この考え方は，PL/I では，stream 入出力と呼ばれているものである．PL/I の record 入出力は，むしろ COBOL に似ている．すなわち，入出力はロジカル・レコードを単位として行なわれる．ファイルは，stream 入出力では文字の連続した連なりからなり，record 入出力ではロジカル・レコードの集まりである．このファイルというものは，JIS FORTRAN や JIS ALGOL ではほとんど表面に現われていない．ファイルの属性は，PL/I では基本的に宣言文で与えるが，COBOL では data division の file description entry で指定される．

入出力の働きについて，ALGOL ではできるだけ手続きの機能を用いて記述するように考えられているが，FORTRAN や PL/I は実用的な面を重視していると考えてよい．また，PL/I がオペレーティング・システムの関連する機能をできるだけ言語の中に取入れようとし，特にダイレクト・アクセス・ファイルの取扱いに関する機能を大幅に取入れていることは特筆すべきことであるが，ここでは紙面の都合もあり，詳細な説明や比較は省くことにし，1，2 の例を挙げる．

```
      WRITE (5,100) A,B,C
100   FORMAT (1H1, 3F11.4)
      output 3 (5, '↑,3(−4ZD.4D)', A,B,C)
      PUT EDIT (A,B,C) (3F(11,4)) PAGE;
```

例 16　FORTRAN, ALGOL, PL/I の出力文の例

にとどめる.

```
      WRITE(5, 200)((A(I, J), I=1, 5), J=1, 5)
  200 FORMAT (1H , 5F 11.4)
      procedure layout;
          format ('/, 5(−4 ZD. 4 D)') ;
      procedure list (item);
        procedure item;
          begin integer i, j;
            for i:=1 step 1 until 5 do
            for j:=1 step 1 until 5 do
                item (A(i, j))
          end;
      outlist (5, layout, list);
      PUT EDIT ((A(I, J) DO I=1 TO 5) DO
                        J=1 TO 5)
                       (SKIP, 5F (11, 4)) ;
```

例 17 FORTRAN, ALGOL, PL/I の出力文の例

## 9. PL/I コンパイラの複雑さ

今まで, FORTRAN, ALGOL, PL/I について, ある意味で共通した点であってコンパイラを考えたとき問題となる点を, 言語の表面的な細かい差にこだわらずに挙げて来たつもりであるが, PL/I には他の言語には存在しない多くの機能があって, コンパイラの設計をむずかしくしている. このような機能または困難さを列挙して見よう.

(1) 略語が沢山あること. DECLARE は DCL, PROCEDURE は PROC など 42 個の keyword に略語が使用できる. これは, 多くの built in function の名前とともに, 大きな索引用のテーブルと処理時間を必要とする. これはまた, parsing の phase を ALGOL とは較べものにならないほど大きくする.

(2) reserved word がない. 文法上の曖昧さはないようであるが, parsing の phase を大きくする原因となっている.

(3) ENTRY 文がゆるされている.

(4) 組込み generic function の作成が想像以上に大変である.

(5) default rule の処理が大変である. 算術データについての default rule は昔の specification より簡単になっているけれども, 特に file についてむずかしいようである.

(6) array および structure expression の処理は, cross section の処理も含めて新しい仕事である.

(7) based variable に関する諸問題 (qualification, allocate, free, area の取扱い等々) は大変多くの困難さを含んでいる. area asignment をとり上げて見ても仲々むずかしい.

(8) 割り込み処理についても, 金物の割込みとの関係も当然発生して, O.S がうまくできていないとむずかしいことになる. また, CHECK prefix は, ブロック構造から規定される名前の有効範囲をはずれたところに, データの名前を書かせることになっている.

(9) event の処理もまた, task と関連してむずかしい問題である.

(10) picture に関わる処理は record 入出力の際には, 必らず必要となるものであるが, 自動変換の効率の問題とあいまって, 強い設計者にとっては泣き所であろう.

(11) Compile time facility の処理は, コンパイラとは別に, preprocessor を作ったほうがよさそう.

(12) 構造体の substructure や基本項目の識別のアルゴリズムも大変である.

(13) 型の自動変換も, 混合演算の自由度が大きいだけに馬鹿にならない.

(14) executor とそれに関連して必要な情報の設計は基本的な問題.

等々数えれば切りがない.

### 参考文献


1) PL/I Bulletin, No. 7, Jan. 1969.
2) JIS 電子計算機プログラム用言語 ALGOL (水準 7000), JIS C 6210, 1967.
3) JIS 電子計算機プログラム用言語 FORTRAN (水準 7000), JIS 6201, 1967.
4) JIS 電子計算機プログラム用言語 ALGOL の入出力 (水準 70), JIS C 6215, 1967.
5) PL/I Language Specification, Form Y 33-6003-1, 1969.
6) IBM System/360 Operating System PL/I(F) Programmer's Guide, Form C 28-6594-3, 1967.


(昭和 45 年 4 月 7 日受付)