# 竹下氏の PL/I の解説について*

COBOL 研究会**



* 文献[1]の章節番号

さきほど新しいプログラミング言語 PL/I の適切な紹介があった[1]．われわれがそれにもとづいて討論した結果を以下に報告したい．もちろん討論は PL/I と COBOL との接触点を中心になされたので，PL/I そのものの全体的な評価は考慮しなかった．

もとよりこのような評価は主観的なものにならざるをえない．まして PL/I についても COBOL[2] についても明細にわたる紹介がいまだなされていないので，第三者としての立場からの批評をうけがたいのであるが，できるかぎり例を引くことによってそれを期待したい．竹下さんはじめ諸氏のご意見をうかがえればさいわいである．

なお本稿の執筆は，幹事西村恕彦が担当した．

## 1. 総　　括

**設問**：PL/I の特徴としてあげられた項目が，COBOL においてどのくらい達成されているか．

**答**：かなりよく達成されている．ただし，仕様決定においてとる基準のちがうものがある．

**設問**：事務用共通言語としてのねらいを COBOL は十分に達成しているか．いないとすれば PL/I はどうか．

**答**：COBOL の仕様はほぼ必要十分なものである．

**設問**：事務用共通言語としての COBOL の守備範囲内の仕事を，PL/I によってよりたやすく，能率よく記述処理できるか．

**答**：共通部分については差はない．

**設問**：汎用の万能言語として COBOL を用いることができるか．

**答**：COBOL のねらいではないが，ほぼ可能である．ただし，関数および外部割り込みに関する仕様が欠如している．

なお，さきの紹介記事は COBOL に関するかぎり，かなり不当なもので，しかもあきらかにまちがいと思われる記述の含まれた，やや粗雑なものであるという印象をうけた．それを下敷にして，弁護的に COBOL を評価紹介してゆこう．

## 2. 書きやすさと文書化

COBOL は制限が非常に少ないので，初心者でも容易にコーディングができる．プログラムを書くことがらくであると同時に，できあがったものが自然に文書化（documentation）されているようにきめてある．特注しなければ標準のやりかたが解釈される部分が多い．初心者でも動くプログラムを書けるうえに，習熟すれば金物の機能まで十分に活用できるみちがいろいろ用意してある．

COBOL の各種の機能は，部分集合や機能単位に分割できるものと考えられるが，低いレベルと高いレベル（たとえば報告書機能）とでは，プログラムの書きやすさと文書化以外の点では，実行用プログラムの能率にかわりがないとみなされている．このことは逆にいうと機械語プログラムをあえて利用する必要がないことになる．すなわち翻訳ルーチン言語であるための

* A Comment on the Introduction to the PL/I made by Toru Takeshita, by COBOL Committee of IPSJ
** 情報処理学会

冗長な手順はできるだけ回避されている．

### 2.1 制限および予約語

COBOL のプログラムはカードの欄にいくらか拘束されている．そのためにある程度の文書化は自動的に保証される．また大きいプログラムは穿孔の能率が高い．端末からの送信に不便かどうかはなんともいえない．

```
003110 YOMIKOMI.
003120     READ    M-FILE     AT  END
003125                        GO  TO    OWARI.
003130     ADD     1          TO        COUNTER.
003140     IF      BANGO      LESS      BANGO-W
003150                        GO  TO    YOMIKOMI.
003160     WRITE   N-RECORD   FROM      M-RECORD.
003170                        GO  TO    YOMIKOMI.
```

この例にみるように，見出し (label) はカードの第 11 欄より左から書きはじめ，本文は第 12 欄より右側に書く．これがカードの欄による拘束のほとんど唯一のものである．

COBOL には 284 語におよぶ予約語 (reserved word) があって，それぞれ固有の機能をもってその意味でしか使ってはいけない．同じ綴りの語をデータ名や手続き名に使ってはならない．この制限は実際のプログラミング作業にあたってひどく不便ではないが，どんな綴りでも使えるにこしたことはない．

PL/I にはいちおう予約語はないと考えられ，任意の綴りの名前を使うことができる．そのかわりに，区切り記号に関する複雑，煩瑣な約束をおぼえなければならない．すなわち，左括弧，右括弧，コンマ，セミコロン，引用符そして間隔までが自由自在に区切り記号として活用させられていて，初心者にはなかなか使いわけられない．そして固有の機能をもった語と，名前との区別はかならずつくとはいえ，ときにより場合により区切り記号により，はなはだ面倒である．たとえば多くの場合に，名前は括弧で囲まれている．しかし常にというわけではない．

したがって PL/I のプログラマは任意の綴りを名前として用いられる利益と引き換えに，多くの機能語をおぼえたうえにさまざまな区切り記号の奇妙な使いわけをおぼえねばならない．

COBOL においては各機能を利用するのに必要な予約語さえおぼえれば，あとは単に語を間隔をあけて羅列すればよく，おぼえるにも，書くにも何ら困難がない．しかもさきの例でわかるように，COBOL の命令は次の形で表現されるものが多い．

機能語〔名前　　〔分離詞　名前〕…〕

これは文書化するのにたいへん都合のよい形で，他の複雑な区切り記号による方式では望めない利点である．区切り記号としては，ただ条件文の終端に終止符が強制されるだけである．

PL/I においては，名前と機能語との区別が区切り記号でつけられるため，かなり奇妙な綴りを強制されることがある．たとえば次のような機能語はまぎらわしいうえに，英語の慣用にあっていない．

ENDFILE ENDPAGE FIXEDOVERFLOW KEYFROM KEYTO LINESIZE ONFILE ONKEY PAGESIZE SUBSCRIPTRANGE ZERODIVIDE

### 2.2 文 書 化

名前の長さは 30 字までで PL/I と実用上かわらない．FORTRAN における 6 字という制限では，よい文書化はまったく望めないことがしられている．

COBOL のプログラムを調べた例では 15 字から 20 字あたりが名前の長さの上限になっている．長い名前は書くときに少し面倒なかわりに文書化がよくなる利点をもっている．

数式のなかに異種のデータを混入してさしつかえない．もともと COBOL のデータは金物の語とは無関係で，桁数も位取りもかってなのだから，同じ型のデータをそろえて演算することは実用的でない．数字の表現に外部表現（たとえば十進）と内部表現（たとえば二進）とがあってもかまわない．このことは入出力の多い作業の場合に有利である．

配列の添字は整数項目名または整定数で下限が 1 と制限されている．任意の数式は書けない．

手続き中には文の形で注記を書くことができるだけで，実行文中には書けない．またデータ宣言部にも書けない．もう少し自由に注を入れたい希望はあるようだが，名前のつけかたなどで工夫するしかない．

### 2.3 簡 潔 さ

データの指定はひとまとめに書ける．

PL/I:

```
DECLARE A STATIC DECIMAL
FIXED (6,2) INITIAL (1234.56);
```

COBOL:

```
77 A PICTURE 9999V99 VALUE 1234.56.
```

すべてのデータ項目はきちんと宣言しないといけない．翻訳ルーチンがかってに想定してくれるデータ記述というものはない．名前のつけかたはまったく自由で，IJKLMN の規則を考慮しないでよい．

いくつかの項目に同じ値を転記する場合は，ひとつの命令で書ける．もちろん各の項目の設計はそれぞれちがっていてかまわない．

```
MOVE ZERO    TO A B C D.
COMPUTE X Y ROUNDED Z = R * S + T.
```

ごくわずかの予約語には略記法が用意されている．

```
CONTROL FOOTING → CF
```

データの名前を略記したいときには RENAMES（再命名）で指定できる．

```
66 GAKU RENAMES KYUYO-SOSIKYUGAKU.
```

手続き名については略記する必要はないと思われる．すなわちその名前の場所に何も働きをもたないEXIT手続きをおけばよい．

ある種の動詞については DEFINE（プログラマ・マクロを作る）命令で略記法を作りだせる．ただしPL/I とはやりかたがちがうから制約がある．

```
DEFINE call WITH FORMAT call sub.
a. PERFORM sub.

DEFINE do WITH FORMAT
do proc i = j, k, l.
b. PERFORM proc VARYING i
FROM j BY 1 UNTIL i > k.
```

## 3. プログラムの構造

### 3.1 割り付け

COBOL には ALGOL や PL/I におけるようなブロック構造の概念がなく，記憶場所の動的割り付けはおこなわない．どの名前でもプログラムのどこからでも参照できる．同じ名前があってもよく，修飾で区別される．

プログラムの手続き部分はプログラマの指定によりセクション単位に区分化（segmentation）され，少ない容量で大きい手続きを処理できる．しかしデータ部分はほとんど静的に割り付けられてしまう．わずかにデータ領域の共用（same area, redefines）が指定できるだけである．

データがたとえば配列などでいちじるしく大きくなることがあるから，データ領域について自動的に区分化してくれる機能があると都合がよい．

### 3.2 データ記述

COBOL におけるデータの指定は一ヵ所にまとめて書く．個人用の作業場所をブロックのなかに設定して自由に使い，よそと無関係にするということはできない．

COBOL のデータは一般にレベル構造を有している．桁のつながりが基本項目で，これが演算の基本単位となる．基本項目のつながり，および集団項目のつながりが集団項目である．配列もまた集団項目として指定される．いちばん上位の集団項目がレコードで，レコード同士のあいだには再定義（redefines）以外には構造的な関係は何も定義されない．

数字基本項目について桁数や位取りのほかに任意の表現を指定できる．COBOL のデータの書き方は，ビットに関するものがない点を除いて，PL/I とほぼ同等である．

PL/I には金物向きのデータ指定がいろいろの組み合わせで書けるようになっているが，COBOL においてもそれは禁止除外されているわけではない．標準的なものはピクチャで指定でき，その他にも USAGE（表現基数）句があっていろいろのデータ表現を管理できる．仕様書には「二進数，十進数，浮動小数点数その他」とあるので，その点で何ら制限をおかない趣旨であることはたしかで，たとえば複素数であってもさしつかえはない．文字データの符号系にも ASCII や BCD などいろいろあってさしつかえない．

COBOL のほうでのみ可能な仕様としては，文字データについても編集ができること，通貨記号のおきかえができること，小数点をコンマで，三桁区切りをピリオドであらわせることなどがある．後2者はヨーロッパ向けの仕様である．

たとえば PICTURE XXBXXBXX と指定した8桁の場所へ，6桁のデータを転記すると，2桁ごとに1桁の間隔があくように編集される．

また PICTURE YY,YYY.99 と指定した場所に数値を転記するとたとえば，Y 365.24 というように編集される．ドイツでいえば，通貨記号に M を用い，小数点にコンマを用いるとすれば PICTURE MM.MMM,99 というような指定になる．これらの書き方は PL/I には含まれていない．

文字項目の長さが可変な場合については，PL/I ではそれは allocate 命令またはブロックへの入口の宣言によって固定される．COBOL においては数字項目，文字項目の長さが可変な場合には次のような書き方ができる．

```
SIZE 整定数 TO 整定数;
```

この場合には，可変長の項目の長さは作成者のきめた標準の方式によって動的に指定される．また次の書

き方もできる.

SIZE 整定数 TO 整定数 DEPENDING ON 整数項目名;

この場合には整数項目名の内容の値が桁数をきめる. いずれも,この項目の桁数は動的に可変であって,しかもゼロ桁（空）でありうる.

### 3.3 集団項目の演算

普通の命令において集団項目を参照すると，それは単に桁のつながりとして処理されるだけで，編集や変換はおこなわれない.

しかし名前によって対応（corresponding）をつけることができ，PL/I の BY NAME とほとんど同じである.

```
MOVE CORRESPONDING DELTA TO MU.
ADD CORRESPONDING MEISAI TO OYA.
```

これによって二つのちがう構造の集団項目のなかから名前の等しい基本項目だけがつきあわされて演算される.

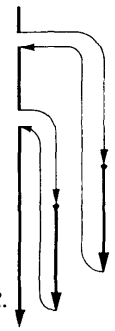
### 3.4 副プログラム

COBOL 言語における大きな欠如は，関数や手続きなどの副プログラムの書き方がないことである. それらのないことが実際に不都合を生ずるか，あるいは事務データ処理ではその必要性がないのか，それはよくわからない. ただパラメータのおきかえを機械語になった実行用プログラムの段階でおこなうことは，COBOL のようなデータではひどく困難で能率がわるいと思われる. 同じ型のデータ指定を有する，つまりレコード中の相対桁位置，桁数，位取り，表現の等しいパラメータのみがおきかえられることにしてもよい. ALGOL や FORTRAN では自明の制限であろうが，それはデータの型が多様でないから実用的におこなえるのであろう. COBOL でも制限つきでよいから使いたいという希望も一部にはあるらしい.

なお，関数や手続きに類するものが COBOL にまったくないわけではない. すなわちデータ指定にあたってレベル番号 88 で条件名を指定しておくと，それは呼出しパラメータのない（固定された）条件関数としてあつかわれる. 宣言の動詞 DEFINE はいわばおきかえ型のプログラマ・マクロを宣言するもので，開いたサブルーチンとなる. 引用の動詞 INCLUDE はライブラリ中の手続きをパラメータをおきかえ，開いたサブルーチンとしてとりこむ. 引用はソース・プログラムの形で翻訳時におこなわれる.

PERFORM 命令は呼出しパラメータのない，閉じた手続きを呼ぶことにほぼ相当している. しかし呼ばれる手続きの入口も出口も任意でよい. たとえば次のプログラムでのコントロールのうつりかたをみよ.

```
A.  PERFORM  E      THRU G.
B.  READ     D-FILE.
C.  PERFORM  F      THRU H.
D.  MOVE     BAN    TO   W.
E.  ADD      1      TO   K.
F.  ADD      X      TO   S.
G.  MOVE     M      TO   N.
H.  WRITE    E-R    FROM D-R.
```



しかし PERFORM 命令の実行はどのような形ででも再帰的になってはいけない.

### 3.5 翻訳時のおきかえ

PL/I のように翻訳時に実行される命令というものはないから，それほど自由な書き方はできない.

ある程度それに近いのは，INCLUDE と DEFINE との命令である. 前者はライブラリに登録されている COBOL で書いた手続きを引用しておきかえる. その書き方の骨組は次のように定義されている.

```
INCLUDE  登録されている手続き名
  [REPLACING {登録されている語 BY 語}…]
```

ライブラリに登録されているデータ記述は次の形で引用できる.

```
レベル番号  データ名  COPY  データ名
                    FROM  LIBRARY.
```

DEFINE 命令の書き方の例はすでにあげた. いまひとつの例をあげておこう.

```
      DEFINE jump WITH FORMAT
      jump on x to m z p.
a.    IF x IS LESS THAN ZERO
      THEN                      GO TO m
      ELSE IF EQUALS THEN       GO TO z
      ELSE                      GO TO p.
```

## 4. 入 出 力

COBOL にはレコード単位の入出力しかない. ストリーム入出力のいろいろの書き方はいっさいない.

### 4.1 大記憶ファイル

大記憶（mass storage）ファイルについては，順呼出し—順処理，乱呼出し—順処理，乱呼出し—乱処理

を指定できる．乱呼出し（random access）は，レコードの所在番地，または記号のキー（symbolic key）を指定することでおこなわれる．必要ならば記号のキーを所在番地に展開する算法を記述できる．

### 4.2 非同期処理

COBOLにおいては入出力のバッファリングが通常のやりかたで管理されるのは当然であるが，一般の仕事について並行に処理を実行させるような書き方は含まれていない．ただ限定された機能として，大記憶ファイルについて，乱呼出し—乱処理を指定すると非同期処理がおこなわれる．

磁気テープのような逐次ファイルとちがって，ディスクのような大記憶ファイルにおいては，そのファイルにたいしていくつかの読みこみ命令がある順序で発せられても，対応するレコードがその順序でavailableになるとはかぎらない．すなわち，複数個の，別々に動きうるアームがある場合にそれはあきらかであろう．

そこで機能的に書かれた順序とは別の，機械的に各レコードがavailableになった順序で手続きが実行される．その手続きは再入的（reentrant）に作るのが普通であろうが，必須条件ではなく，また言語の問題でもない．

これに関連して待ち合わせの命令HOLDとデータレコードの待ち合わせ領域（saved area）の指定とがある．

COBOLにおける非同期処理の機能はPL/Iにおけるそれよりも限定された，低いレベルのものと考えられる．ただし，PL/Iにおいて大記憶ファイルのレコードを非同期的，即時に処理する書き方がどのようなものであるかはよくわからない．

### 4.3 割り出しと割り込み

厳密にするために用語を定義する．

プログラムの流れのなかで機能的に書かれたその時点で発生し，枝分かれのおこる条件の検出を割り出しという．

プログラムの外あるいは内で機能的に開始され，その発生および検出の時点がプログラムの流れのなかでは管理困難な条件の検出を割り込みという．

この定義にしたがうならばPL/Iにおいて記述されているinterruptと称するもののほとんどは割り出しに属している．ただしその書き方については特異なものがあって，割り込みにも共通に使える書き方をとっている．

COBOLにおける割り出しの書き方の例をあげる．

計算の過程または答の桁あふれ：

計算命令； ON SIZE ERROR 無条件命令．

入力ファイルの終り：

入力命令； AT END 無条件命令．

レコードを定位できない乱呼出し：

入出力命令； INVALID KEY 無条件命令．

COBOLにおける割り込みの書き方は，プログラムの機能的な順序では管理できない，あるいは書きにくい条件について適用される．そしてプログラムの主流に含まれない宣言部（declaratives）にまとめて書く．

入出力管理システムへの追加：

USE AFTER STANDARD ERROR
PROCEDURE ON ファイル名．

USE { BEFORE / AFTER } STANDARD { BEGINNING / ENDING } { REEL / FILE / UNIT }
LABEL PROCEDURE ON ファイル名．

報告書におけるコントロールやページの切れ目：

USE BEFORE REPORTING 行の名．

これらの書き方はほぼPL/Iにおける計算および入出力における条件と匹敵している．COBOLにあると良いと思われる機能に端末通信管理がある．ひとつの仕事のプログラムが，いくつかの端末によって外部から割り込まれながら進行する機能は含まれていない．

含めるとしたら非同期処理とほとんど同じ書き方で記述できると思われる．すなわち端末の組を大記憶ファイルそのものとみなし，乱呼出し—乱処理をすればよい．

ただし現状においても，たがいに独立で無関係な複数個の仕事のプログラムがひとつの計算組織中で並行的に多重プログラム的に処理されうることを否定するわけではない．それは操作システムの機能に属し，言語の問題ではない．それから，単一の仕事が単一の端末を有する場合にはもちろん順呼出しファイルとしてあつかえよう．

### 4.4 プログラムのデバッグ

デバッグやチェックのための便宜はCOBOL言語のなかには用意されてない．操作システムにまかせるしかない．

### 4.5 作 表

COBOLには報告書機能（report writer）があるの

で，ページの体裁，コントロールの切り方に関しては，他のどの言語よりも優れた書き方ができる．PL/I における作表能力とは比較にならない．

## 5. リスト処理

執筆者の考えでは通常の手続き的な言語（procedural language）をリスト処理可能にするには以下の機能が必要である．

（a） 演算の基本単位を基本項目とする．
（b） 基本項目の集合を集団項目とする．
（c） 集団項目の値を転記できる．
（d） 集団項目を要素とする配列を宣言して，添字によってその要素を参照できる．

あるひとつのリスト中の各要素が均質，等長のときにはうえの仕様は十分であって，リストや木構造を任意に操作処理できる．ただし実行用プログラムの能率が，リストのつなぎになまの番地を使ったほうが高くなる場合には，address 関数と pointer 変数が望ましい．pointer 変数は計算が困難なために，計算を禁止されることが多い．もし計算をみとめるとあいまいさを生ずる．添字についてはあいまいさなく計算ができる．

たとえばある配列中の I 番目の要素 E(I) 中に，そのとなりの要素へのつなぎをおさめてみよう．

添字によるつなぎ：

```
LINK OF E (I) = I + 1;
```


pointer によるつなぎ：

```
E(I).LINK=ADDR(E(I+1));
```

または：

```
ADDR(E(I))→E.LINK=ADDR(E(I+1));
```

IBM 7090 のような計算機では番地の値に 1 を加える（XLOCF(E(I))+1）というような操作はあいまいさなく一意に合意できようが，IBM System/360 のような，語の単位の確定していない計算機では番地にたいする演算（ADDR(E(I))+1）を一意に定義することはむずかしい．

さて，リスト中の各要素が均質等長でないときには，それを等長な要素のリストに再分割するのが常道であるが，そうしないとすると a～d の機能だけでは不十分である．そのときには，based variable が「のりうつる」性質が有効である．「のりうつり」は PL/I のように pointer 変数によってもできるが，また要素がうろこ状に重なり合った配列を利用しても可能である．あるいは関数の記法にしたがってもよい．

記号処理においては利用される記憶の量が動的にいちじるしく変動することがある．そこで，たとえば空き地リスト（available space list）を管理する手続きが必要になる．それをリスト処理の手法を用いていちいちコーディングすることは可能だが allocate とか free とかの命令があると便利なことはたしかである．

## 6. COBOL と PL/I との比較

おわりに，文献[1]にあげられた COBOL と PL/I との比較の箇条書きをさらってみる．

「PL/I がすぐれている点」について

（1） 手続きとデータの記述を別々にまとめなくともよい

これは選択基準の問題であって，文書化，翻訳能率，デバッグ，書きやすさなど，いちがいに別々にできたほうがよいと断定はできない．**3.2** をみよ．

（2） カードの欄に拘束されぬ

拘束されないほうがよいと断定はできない．**2.1** をみよ．

（3） 注はどこに書いてもよい

COBOL もそうなっているとよい．

（4） ステートメント，特にデータの記述が簡単

これはみかけ上だけ通用する不正確な断定である．COBOL で書いたプログラムを PL/I で書いたときにずっと簡単になるという事実はない．

（5） 略語が数多く使える

COBOL ではほとんどだめである．

（6） 予約語（reserved word）がない

優劣はきめがたい．**2.1** をみよ．

（7） 必要なステートメントの種類が少ない

意味がよくわからない．

（8） ブロック構造をとっている

選択基準の問題であって，優劣はきめがたいと思われる．プログラマの好みもあるし，初心者にはなじみにくいのではないか．デバッグや文書化はどうか．

（9） 手続きに一つ以上の入口が可能

COBOL には他の言語での副プログラム（手続きや関数）に相当する概念があてはまりにくい．PERFORM 命令は複数個の入口，複数個の出口をもっているし，串ざし実行の手続きをよそから呼ぶこともで

きる．**3.4** をみよ．

(10) 再帰的ルーチンが書ける

COBOL には関数の書き方がまったく欠如している．

(11) 浮動小数点の演算

まちがいである．COBOL の仕様書でも除外していないし，実際の翻訳ルーチンの例も多い．

(12) 可変長のビットや文字のストリング

ビットの指定は COBOL にはない．文字および数字の可変長データの管理の仕方は PL/I より優れている．だからこの断定は不正確である．**3.2** をみよ．

(13) ビットや文字のストリングの演算

文字のストリングについての演算としては，転記，比較，編集，桁調べ，変換，連結 (concatenation) が考えられる．連結についてはレベル構造を利用して間接的に実現できる．その他の点では同等または COBOL のほうが優れている．

(14) 添字の下限が指定でき，負の整数でもよい

そのとおりである．

(15) 添字の数に制限がない

COBOL は三次元までで，しかも一般の数式は使えない．

(16) テーブル (行列) の全体や断面が一つの式で

COBOL でも集団項目としてあつかえる制限内ではある程度可能である．しかし Iverson 言語におけるような，配列名を単に参照すると対応する要素間の演算が定義されるやりかたは不可能である．この書き方の有能さは Iverson の著書にみられるとおりであるが，さらにいうならば，二つの演算子を同時に用いる一般積 (generalized product) の書き方があれば行列演算はずっと容易に記述できる．

だがはたして事務データ処理において行列演算をそこまで簡潔に記述せねばならぬ，切迫した必要性があろうか．むしろ数表索引などのほうがより要求されていて，それはまさに COBOL には組み込まれているが，PL/I には入っていないのである．

もし事務データファイルを集合あるいは配列として簡単に参照，処理できる書き方が情報代数や情報操作論のようにあれば，これはもっとも強力な道具となろうが，早急には望めないだろう．

(17) ピクチャがより強力

まったく賛成できない．**3.2** をみよ．

(18) 初期値の与え方が簡単

理解できない．典型的な例でみるかぎりほぼ同等である．**2.3** をみよ．

(19) データを一々記述せぬともよいことがある

COBOL が対象とするような事務データ処理の仕事では，それほど意義がない．文書化とかデバッグとかを考えると利点かどうかうたがわしい．

(20) ラベルを変数としてあつかえる

COBOL ではできない．ただし GO TO 命令の行き先を書きかえてスイッチさせることはできる．

(21) 論理演算が可能

意味がよくわからない．ストリングにたいしてマスクをかけたり，ストリングを結合したりする機能は，レベル構造さえあれば不要であろう．

条件を論理演算詞 NOT, AND, OR でつないで複合条件を作るやりかたは COBOL でもきちんとできる．またさまざまな略記法は COBOL でのみ可能で，より自然な，読みやすい，効率のよい書き方ができる．だから PL/I のほうが優れているとはとうてい信じがたい．

(22) 同一の値は一つの式で多数の変数に

この項は完全にまちがいである．たとえば例にあげられている A, B, C, D = 0; はまさにその意味で COBOL に通用する．各変数ごとに切捨てか四捨五入かを選択もできるし，割り算の剰余をとる書き方も提案されている．

この仕様が IBM の COBOL にはがいして含まれていないことからきたまちがいであろうが，一般の COBOL にはちゃんと存在するのである．

(23) 行列式の加減算，スカラ量との乗算が一つの式で

そのとおりである．なお第 16 項をみよ．

(24) 強力な DO ステイトメント

賛成できない．PL/I の DO と COBOL の PERFORM とは，ある条件に達するまでの反復と，制御変数による反復とを基礎としている点でほぼ同等であるとみるのが公平であろう．ささいな点では両者それぞれに特長がある．

COBOL としては，相手の手続きがどこにあってもよく，またそれは串ざしの実行もできるとか，ひとつの命令で複数個の制御変数を管理できるとかの特長がある．

(25) コア・スペースの弾力的利用

PL/I のほうが優れている．

(26) インタラプト・オペレーション

組み込みずみのものについては，それほどの差はな

い．しかし PL/I のほうがいくらか融通性，拡張性があると思われる．

(27)　プログラム・チェック用の便宜

COBOL では言語の仕様からは除外されている．

(28)　マルチ・プログラミング

意味がよくわからない．なお第27項および **4.3** をみよ．たがいに無関係な独立の仕事を並行的に処理させるだけなら，それは操作システムの機能である．

(29)　コンパイル時の便宜

翻訳時に実行される命令は書けない．しかし翻訳にさきだって，ライブラリを引用したり，動詞を定義したりはできる．

「COBOL がすぐれている点」について

文献[1]はこの点で，ささいな仕様をわずか3項あげているに過ぎないが，その他に以下の仕様が考えられる．これらによって COBOL がより実用的な言語をねらっていることが了解されよう．なお分類と報告書機能とは 1963 年，表操作は 1964 年以来 COBOL に入っている．

（1）　分類（sort）ができる

これがあればファイルの再分類と多数回の通読とを含む，一連の長い事務データ処理のジョブをひとつのプログラムで統一的に記述できることになる．従来のシステムでは分類だけは別のランにする例が多いが，今後は一貫したプログラムで記述し，ジョブ・モニタや操作員の介入を不要にする方向に進むであろう．

（2）　完全な報告書機能（report writer）

コントロールの切り方とページの体裁との指定が容易に記述できる．この機能の有無は事務データ処理のプログラミング時間を大きく左右する．

（3）　表操作（table handling）

表引き（search）や指標レジスタの効率のよい利用．事務データ処理では料率表やコード・ブックのように単純な算法では記述できない関数がしばしばあらわれるので，表引き機能の手当てはのぞましい．

（4）　キャラクタ・セットが実際的

ホレリスの48字と ＜ ＞；の3字合計 51 字で，しかも後者の3字はなくても十分に読みやすいプログラムを作れる．また区切り記号の使い方がたいへんゆるやかである．

（5）　入出力周辺の指定が明白

PL/I では Environment の指定がすべて言語の外として除外されている．

COBOL では入出力装置，入出力技法，ファイル形式などの書き方が Environment Division と File Description とに含まれている．これらをプログラマは十分に管理することができるし，またこれら周辺的な指定のどこまでを人手の管理下におき，どこまでをシステムの固有の管理にゆだねるかについての明白な標準を提起している．

## 参考文献

1)　竹下　亨：新しいプログラミング言語 PL/I の出現とその特徴，情報処理 **7**-4，1966，202～212

2)　DOD：COBOL, Edition 1965

3)　IBM：PL/I Language Specifications, C 28-6571-3

（昭和41年12月6日受付）