新しいプログラミング言語 PL/I の出現とその特徴*

竹　下　　亨**

はじめに

1966 年は，プログラミング言語発展の歴史において，ある意味をもった年のように思える．その理由は本年は世界最初のコンパイラ(IBM 704 FORTRAN)が出てから10年目に当たっており，また ALGOL や COBOL の最初のコンパイラが作られてから5年目でもあり，そして新しいプログラミング言語 PL/I の第1号コンパイラ（IBM オペレーティング・システム/360 のデザイン・レベルFコンパイラ）が完成し，実際に使用され始める年である．

PL/I が発表された当初は，分かりやすい解説書がなく，変更が次々と加えられたことと，使いなれた既存の言語があるので新しいものを敬遠する向きもあり，またコンパイラができる時期が遅いという理由から，急速には一般に受け入れられなかった．しかし，最近になりそのよさが次第に理解され，またいよいよそのコンパイラが使われ始めるとあって，PL/I に対する関心が高まって来た．そこで，本稿では文法そのものの説明ではなく，それが出現するに至るまでの経過，その特徴，および既存の言語との差異について述べてみたい．

1. FORTRAN から PL/I まで

FORTRAN なる言語が生まれてからひと昔もたっているが，それを発展させていくだけでは飽き足らず，新しい高度のプログラミング言語が開発されることになった．PL/I は FORTRAN のほか ALGOL や COBOL のすぐれたところを集大成しており，それに新しい進んだ機能を数多く導入している．PL/I をよく理解し，それを正しく評価するには，その前身といってもよい FORTRAN, ALGOL, COBOL の三つを一とおり眺めてみる必要がある．

1.1 FORTRAN の出現とその性格

FORTRAN は，最初 IBM 704 電子計算組織に対して，13 名の人々（IBM より J.W. Backus ほか9名，マサチューセッツ工科大学，カリフォルニア大学ローレンス放射線研究所，ユナイテッド航空機より各1名）により 1954 年より2年半の歳月を費して開発された．その後 704 FORTRAN II が 1958 年にでき上がっており，その頃から他の機種のコンピュータ（IBM および他社のもの）にも，次々と FORTRAN が作られた．1962 年始めにはデータ・タイプの宣言や論理演算を含めた FORTRAN IV が発表されている．現在では科学技術計算用および汎用のコンピュータにはほとんどすべて FORTRAN が提供されており，FORTRAN は世界で最も広く使われている言語である．

そのため 1962 年5月より ASA (American Standards Association) で標準化が始まり，ASA Full FORTRAN と ASA Basic FORTRAN の二つが検討され，さらに ISO (International Organization for Standardization) ではこれに ECMA (European Computer Manufacturers Association) FORTRAN を加えたものを推せん規格案としている．文献 1) には次のように FORTRAN 開発の目的と目標が記されている．「FORTRAN の目的は科学計算の問題を 704 で解くまでに要する仕事を大幅に減らすことである．FORTRAN プロジェクトの目標はプログラマが数学の表現法のようにコンサイスな言語を使って数値計算の手続を示し，その手続を遂行する効率的な 704 のプログラムを自動的に作成することである．」

このように，FORTRAN は最初は 704 という特定なコンピュータに対して，一つのメーカにより開発された．当時は，現在のように多種類のコンピュータの出現を予想しなかったので，他の機種との互換性を主目標にしていない．それよりもコンパイルされてでき上ったプログラムの効率を重視している．そのため機械そのものに拘束された（machine dependent の）要素が多少ある．しかし，FORTRAN は文法が簡単であり，一般性の高い言語で，あらゆるコンピュータに適用できるものである．

FORTRAN は文法的には非常に簡単で，数値計算

---

* Debut of New Programming Language, PL/I, and its Features, by Toru Takeshita (IBM Japan, Ltd.)

** 日本アイ・ビー・エム（株）

は関数を含む数式（arithmetic expression）で表現され，その順序や繰り返えしを示すにはコントロール・ステイトメントを使い，入出力ステイトメントでデータの読み書きを行ない，配列の大きさや，データの形やコア・スペースの共用関係などはスペシフィケーション・ステイトメントで指定する．

FORTRAN でプログラムが組まれるものは，主として科学技術計算であり，その時だけの計算（one-shot job）が多く，処理時間も比較的短い．繰り返えし部分を含む複雑な計算があるが，全体としての流れは簡単な場合が多い．入出力データの形が単純であり，入出力の時間が短い．また，データ・ファイルを必要とすることは少ない．2進法，固定長語，浮動小数点のコンピュータにより適している．

1.2 ALGOL の出現とその性格

スイスやドイツなどの大学関係者が世界共通の算法言語を提案し始め，それに米英の研究者が加わって，1958 年にチューリッヒで暫定報告書（文献 7)）が出された．これに種々の改良を加え 1960 年に 1 月のパリにおける会議の後 ALGOL 60（文法）が出た．これを基準にして欧州を中心にコンパイラが開発され，それと同時に ALGOL で書いたプログラムの交換が盛んになった．1962 年 4 月にはローマにて会議が開かれ Revised ALGOL 60 が発表された．このほか ECMALGOL，ECMALGOL+recursivity，IFIP subset がある．また，入出力の手続については，Full I/O（ACM）と Subset I/O（IFIP）なる二つが提案された．

ISO は 1964 年 5 月ニューヨークの会議で標準 ALGOL をつくることを決議している．ついで，翌年 1965 年 10 月に開かれた ISO 東京大会で，上記の四つのレベルの言語と二つのレベルの入出力手続が，三種類の FORTRAN と共に推せん規格案となった．

ALGOL は計算機で解こうとする数値計算を記述する算法言語として提案されたもので，当初は機械による翻訳をあまり考えてない．どのコンピュータにも通用する共通語とすることが主目標であって特定の機種にかたよった要素を排除している．その書き方は日常の英語と常用の数式表現に近いものを採用しており，ステイトメントの長さや添字部（subscript）にも制限がなく，できるだけ一般性をたもっている．その言語で書いたプログラムはいわゆるブロック構造をとり，begin で始まり end で終る一連の宣言と一連のステイトメントがブロックを構成する．宣言はいずれも通常そのブロック内だけに意味を持つ．

ステイトメントの種類としては，assignment, go to, dummy, conditional, for, procedure の 6 種類である．入出力の手続については，始めは何らの規定がなく，個々に決められていたが，1964 年になって先に述べた二つの入出力のレベルが提案された．

対象となるアプリケーションとコンピュータは，FORTRAN の場合とほぼ同じである．

1.3 COBOL の出現とその性格

数多い種類のコンピュータが出てきたので，一つの企業内で異種のコンピュータを使用するようになり，その間の互換性が大きな問題となってきた．1959 年 5 月に米国防省で事務計算用の共通語の必要性と可能性を検討する会議が開かれ，翌月メーカ側 6 名，政府側 3 名よりなる Short Range Committee が構成され，同年 12 月に Executive Committee に報告書を提出している．1960 年 4 月に COBOL-60 の文法書が発行され，米国の数社がその開発を開始した．次いで 1961 年半ばに COBOL-61 が，1963 年に COBOL-61 Extended が出され，1965 年 7 月には COBOL, Edition 1965 が発表されている．

COBOL の標準化の検討は 1963 年 1 月に始まり，現在 ASA, ECMA, ISO で審議を重ねている．

COBOL は COmmon Business Oriented Language の略であり，事務計算用の共通言語として異種のコンピュータ間の最大の互換性を目標としている．そして，できるだけ英語の表現法に近いもので処理手続を表現し，短時間に多くのプログラムを作成することが可能であり，プログラムの修正，保守と新しい機種への切換えを容易にすることもねらいとしている．

各プログラムは，Identification, Environment, Data, Procedure の四つの Division より構成される．一つの手続（procedure）はセクション，パラグラフ，センテンス，ステイトメントの順に小さい単位に分かれる．

COBOL が対象としているアプリケーションは事務計算であるので，継続業務（routine work）で処理時間が長いものが多い．プログラムは繰り返えし部分が少なく計算そのものは簡単であるが，全体の流れは複雑である．通常（テープやディスクの）データ・ファイルを必要とし，入出力データの形が技術計算の場合のように単純ではない．また入出力データの量が多く，処理時間の大部分は入出力操作の時間である．10 進法，可変長語，固定小数点のシステムに適している．

2. PL/I の誕生

FORTRAN, ALGOL や COBOL が広く普及しているにもかかわらず，なぜ新しい言語が開発されることになったのか，そしてそのねらいは何かを考えてみたい．

2.1 新しい言語の必要性

最初の FORTRAN が考えられてから約 10 年を経過し，その間にコンピュータは演算処理装置や主および補助記憶装置が急速に進歩し，それと平行して新しい種々の入出力装置が次々と開発された．さらに，データ伝送の発達にともないオンライン・システムが出現し，また，割込み処理機構，メモリー保護，インターバル・タイマなどが備えられて，マルチ・プログラミングが可能となった．かくて，より大規模で複雑かつ高度の問題が，現実的に処理できるようになった．

一方，ソフト・ウェアの面でも，従来の個別の (stand-alone) コンパイラやアセンブラのみでなく，オペレータの判断や手作業の多くを引き受けて自動的に仕事から仕事を継続進行せしめるモニタやオペレーティング・システムが提供されるようになり，計算機室のオペレーションのやり方も大幅に変わってきている．他方，アプリケーションも，従来とは次元の高いものが考えられるようになり，事務計算と技術計算が入り混ったものや，データ伝送処理，タイム・シェアリングなど高度の応用が考えられるようになってきた．

このような状況にあって，技術計算，事務計算，両者の混合，リアルタイム処理などのあらゆるアプリケーションに適用でき，しかもモニタや OS の機能をフルに利用するとともに，マシン・コンディション（オーバーフロ，エンド・オブ・ファイルなど）もテストでき，従来機械語に頼らねばならなかったことも組めるような言語が要求されるのは当然である．

プログラマの中には，専門家や素人，アプリケーション・プログラマとシステムズ・プログラマ，あるいはクローズド・ショップとオープン・ショップ，事務計算屋や技術計算屋といった区別ができるが，それぞれ必要とするレベルの機能（facility）に応じた範囲 (subset) が選択でき，修得が容易で，使いやすいものが望ましい．

1964 年 4 月に発表された IBM のシステム/360 やこれと似た RCA スペクトラ 70（HITAC 8000 シリーズ）は 2 進法固定長語の機械と 10 進法可変長語を一緒にし，それにマルチ・プログラミングやリアルタイムの機能を入れたもので，技術計算と事務計算が 1 台で同様の効率で処理できるようになっているが，これにマッチしたプログラミング言語がまさに PL/I である．この出現により，これまでのように FORTRAN（または ALGOL）と COBOL といった二つの言語を習得・保守をしなくともよく，異種のプログラマも一つの言語でコミュニケーションすることになる．

2.2 新しい言語の開発

1963 年 10 月に SHARE と IBM の 10 名（後に 2 名追加）の委員よりなる Advanced Language Development Committee が設立され，新しい言語を決めることになった．

翌年 3 月に PL/I（当時 NPL と呼ばれた）報告書第 1 版をまとめ，これに対して寄せられた批判や提案を取り入れて 6 月に第 2 版の報告書を出している．その後の変更，修正，改良が加えられた第 3 版について，IBM ではシステム/360 の PL/I を開発している．

オペレーティング・システムにはデザイン・レベル 44 K のが，ディスク・オペレーティング・システムとテープ・オペレーティング・システムには 10 K のコンパイラがあり，そのうち 44 K のものは 1966 年 8 月中に完成することになっている．なお，システム/360 のモデル 67 に対しては，端末機より送信できる対話方式の PL/I が開発されつつある．

2.3 新しい言語のねらい

新しい言語のねらいとするところを一言でいうと，現代およびかなり先の将来までのコンピュータ使用上の必要性を充すことである．

まず，あらゆる分野の進歩したデータ処理に対しても適用可能で，初心者から熟練者まで，誰でも使いこなすことができ，特定の機種に拘束されずに，しかも金物および OS の全機能の利用ができることである．

つぎに，新しい金物や OS と同じように積木方式 (modular system) をとっており，教育用または初心者用，技術計算用，事務計算用といった部分 (subset) をとり出してその範囲だけで使うことができ，簡潔にしてわかりやすいプログラミング言語であって，理論的な算法言語ではない．

PL/I の究極の目的は，コンピュータの処理能力と応用範囲の飛躍的発展に応じて，それを十二分に活用するため，（高度の内容を持つ）プログラムの多数を短時間で作成できるようにすること，換言すればプログラマひいてはコンピュータの生産性 (productivity)

を高めることである.

3. PL/I の書きやすさ

PL/I はプログラマがいろいろな制限を気にしなくてもよいように，また一寸した違反がエラーを生じないように従来の言語より，制限を非常に少なくしている．また書く労力や誤りを減らすために，できるだけ簡潔にコーディングができるようになっている．プログラマが指定をしないときは，コンパイラの方で判定して決めてくれるので，標準の場合は一々指定しなくともよい．したがって，COBOL などよりソース・プログラムがずっと短かくなる.

3.1 制限が少ない

プログラムはカードの欄（column）に拘束されずにどこから書いてもよく，遠隔地の端末機から送信するにも便利である.

COBOL のように制限された語（reserved word）がなく，何をラベルやデータの名前に使ってもよく，コンパイラは文脈により単語の使用目的を判断するようになっている.

データの名前の長さは，31 字まで許される.

計算部分を書くときに，同一の式（assignment statement）に異種のデータを混入しても差支えない.

配列の添字（subscript）の下限が指定でき，それが負の整数でもよい．また，添字の数に制限がなく，添字部はどのような算術式でもよい.

説明に書き添える注（remark）は，いくら長くともよくステイトメントのどこに挿入してもかまわない．したがってドキュメンテーションとしてもよいものができる.

例.

```
Y=A/*COEFFICIENT*/*X+B/*
                    CONSTANT*/;
```

3.2 簡潔に書ける

データの種類，特性，次元（dimension），初期値などが一つのステイトメントで宣言できる.

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

宣言しなければ，**指定がない場合の**（default）**ルール**が適用され，標準形とみなして処理される．数値でその特性が宣言されてないときは，名前の最初が I から N までの間の字なら，FIXED REAL BINARY とみなされ，それ以外は FLOAT REAL DECIMAL とされる．精度も指定しなければコンパイラが標準のものをとる.

いくつかの変数に同じ値を与えるときは，一々置換式を書かぬともよい.

例. A, B, C, D=0;

基本語（key word）の大部分（合計 28）に，略語が使える.

例. 略さぬ場合	略した場合
PROCEDURE	PROC
DECLARE	DCL
DECIMAL	DEC
CHARACTER	CHAR
PICTURE	PIC

上記の例のような正式に定められた略語のほかに，ユーザが任意の略語を使いたいときは，コンパイル時に指定する．すなわち，その略語をコンパイル時の変数とし，その値は対応する正式の基本語としておいて，プログラムの本文では略語を使えばよい.

4. PL/I で効率的なプログラムの作成

コンピュータの構成部分（中央演算処理装置，コアスペース，入出力チャンネル，入出力装置，プログラムなどの facility）を効率的に使用するため，PL/I でプログラムを書く場合にコンパイラに対してプログラマが種々の指示を与えられるようになっている．それは単に命令の数を一つ二つ減らすのではなく，もっと次元の高いやり方で効率を高める.

4.1 コア・スペースの弾力的利用

PL/I では変数の名称の通用範囲として EXTERNAL と INTERNAL があり，後者の場合はそのブロック（PROCEDURE または BEGIN で始まって END で終る一連のステイトメント）内に限定される.

コア・スペースの配分には STATIC, AUTOMATIC と CONTROLLED の 3 種類ある．STATIC のときはプログラムをコアに入れるときに固定される——FORTRAN のやり方．AUTOMATIC のときは，そのブロックに入ると自動的にスペースが割り当てられ，ブロックを出れば消失する——ALGOL のやり方.

しかるに，CONTROLLED の場合には，割り当て（とその取り消し）は ALLOCATE（と FREE）のステイトメントで行なうことができる．また CONTROLLED のデータにスペースが配分されているか否かを見るのに，ALLOCATION (X)（X は変数名）なる組込み関数（値は 1 か 0）を使う.

宣言がないときの配分は，EXTERNAL データは STATIC, INTERNAL は AUTOMATIC とみなされる．

```
例. DECLARE X EXTERNAL, Y FIXED,
      Z CONTROLLED, I STATIC INITIAL(0);
                ⋮
      IF I=0 THEN DO; ALLOCATE Z; I=1;
        END;
```

コア・スペースを多く要する配列の場合は，あらかじめ固定したスペースをとらなくとも，計算の実行時になってから次元（dimension）が決められる．

```
例. DECLARE MATRIX (*, *);
```

4.2 ダイレクト・アクセス・ストーレジの利用

多数個のファイルをディスクやドラムに入れて，それぞれアプリケーションに応じた効率的方法（シークェンシャル，インデックスド・シークェンシャルやダイレクト・アクセス）で読み書きが可能である．

```
例.
    DECLARE MASTER FILE
       KEYED (10) INTERNAL;
            ⋮
       READ FILE (MASTER) INTO
            (MASTER-RECORD)
            KEYED (MASTER-KEY);
```

4.3 命令の数を節約できる形のサブルーチン

PL/I の手続（PROCEDURE）には頭の入口のほかに，途中の入口（ENTRY で指定）をいくつでもつけることができるので，一つのサブルーチンでありながら入口を違えることにより，一部分異なった機能や計算を行なうことができる．

```
例. A: PROCEDURE;
            ⋮
    X: ENTRY;
            ⋮
    END A;
```

また，PROCEDURE のあとに RECURSIVE と書くと自分で自分を呼び出す再帰的サブルーチンを作り出すようコンパイラに指示することができる．このようにプログラマが難しい操作を考えなくとも，特殊な形態のサブルーチンが作り出される．

4.4 アシンクロナス・オペレーション (Asynchronus Operation)

PL/I ではある仕事（task）が進行中に別の仕事を始めて，両者を平行させて遂行すること（multi-tasking）ができる．前者があるところまで進んだところで，後者が終了していなければその先へ進めない場合は，後者が終るまで待たせる（synchronize する）ことが可能である．なお，後者の仕事を始める際に優先順位（priority）をつけることもできる．

こうすることによって，たとえば入出力操作を含む部分と計算処理のみの部分を平行して実行させ，CPU の時間を効率的に使うように，プログラムを組むことが可能である．

```
例.
  P1: PROCEDURE;
            ⋮
      CALL PRINT (A, B) TASK (T2)
      EVENT (ET2) PRIORITY (−2);
            ⋮
      WAIT (ET2);
            ⋮
        END;
            ⋮
      PRINT: PROCEDURE(ALPHA, BETA);
            ⋮
        END;
```

この例では，P1 なる手続が進行中に，PRINT という手続を呼び出して，これを T2 なる仕事（task）で ET2 なる事象とし，優先順位は通常のものより 2 だけ小さいとしている．この時点から両者が平行して遂行されるが，後者が終ってから次に進みたいときには WAIT（仕事の名前）を書く．後者の仕事に与えられた優先順位を知りたいときは，PRIORITY（仕事の名前）なる関数を使う．

TASK ( ), EVENT ( ), PRIORITY ( ) はどれか一つでも，あるいはどのように組み合わせてもよい．TASK ( ) がないと PRIORITY 関数を使っての仕事の優先順位を知ることができないし，EVENT ( ) がないと，WAIT ( ) が使えない．

5. PL/I に含まれる強力な機能

PL/I にはその言語の目標である数多くの強力な機能を包含しているが，それらの主なものを若干考察してみよう．

5.1 ブロック構造 (Block Structure)

PL/I で書かれたプログラムは，ステイトメントが基本となり，それがまとまってグループを作り，さらにグループが一つ，あるいはいくつか集ってブロック

を作る．プログラムは一つまたは複数個のブロックより構成される．

このため，コーディングがしやすく，ドキュメンテーションとしても見やすく，またブロックごとにプログラムのチェックをすればテストが容易である．

ブロックは PROCEDURE または BEGIN で始まり END で終る．それは変数の名前やラベルの通用範囲を自動的に定めることができる．すなわち EXTERNAL でなく INTERNAL な名前はそのブロック内しか適用せず，同一の名前を異なるブロックに使用しても差支えない．一つのプログラムを分割して数人で書き，これを組み合わせるときの支障が少ない．

先に述べたように，コア・スペースの配分が，AUTOMATIC になっているデータ（何も指定しないときはこのようにみなされる）に対しては，そのブロック内にコントロールがあるときにだけコア・スペースが割り当てられ，ブロックの外へ出るとそのスペースは他に与えることができる．このようにダイナミック・アロケーション（dynamic allocation）が可能である．

例.

```
A : PROCEDURE;
    DECLARE Y……;
      ⋮
    O : BEGIN;
        DECLARE X……;
          ⋮
        END;
    S : BEGIN;
        DECLARE X……;
          ⋮
        END;
END A;
```

5.2 種々の複雑なデータの表現

実数，複素数共に 10 進と 2 進の別，固定，浮動の別や，桁数（全桁数と小数点以下の桁数）を指定できる．これら数値のほかに可変長のビットやキャラクタのストリングの取り扱いができる．すなわち，いずれも比較や結合（concatenation）が可能であり，前者にはそのほか論理演算ができる．また，事務計算の場合の複雑な形態のデータが，サブフィールドの集合のそのまた集合といった何段階のレベルを含む構造体（structure）として表現できる．構造体は COBOL のレコード（record）と類似している．そして，個々のデータのフォーマットは非常に複雑であっても，ピクチャ（PICTURE）を使って容易に記述できる．

例 1.

C=A|¬B（A, B 共にビット・ストリング）

C=A||A|B（A, B は共にビットかキャラクタのストリング）

例 2.

```
DECLARE TABLE (10), 1 RAINBOW,
    2 RED, 3 CRIMSON, 3 PINK, 2 BLUE,
    3 NAVY, 3 TEAL, 2 YELLOW;
```

例 3.

```
DECLARE A PICTURE 'XAA9AA';
```

5.3 配列と構造体の計算

ベクトルや行列の加減算およびスカラ量との乗算を行なうときは，他の言語の場合のように対応要素ごとの演算式を書かなくとも，配列の名前を一つの式で示せばよい．またこのことは同一構造をもった構造体の加減算，スカラ量との乗算にも適応される．構造が異なる構造体の対応する要素について演算したいときは，BY NAME を式の後につける．

例 1.

```
C=2*A+B;
```

（A, B, C は共に配列か構造体）

例 2.

```
Z=X+Y, BY NAME;
```

（X, Y, Z は同一構造でないが，同一名称の対応する構成要素をもつ）

5.4 多種類の入出力方式

PL/I の設計目標を最もよく達成しているものの一つが，幅の広い入出力の機能であって，データ・セットをキャラクタの連続した流れと見なすストリーム入出力（STREAM I/O）と不連続のレコードを扱うレコード入出力（RECORD I/O）の 2 種類がある．前者は技術計算に向いており，後者は COBOL の入出力方式と似ている．

ストリーム入出力には次の方式がある．

データ名のリストが与えられた転送（List-Directed Transmission）

データの名前（コアにおける場所）を示すだけで，フォーマットを指定する必要がない．コア・スペースと外部媒介体のデータの位置が一対一対応するときに便利である．

例. RUTLIST ((MAXDAY (I), MINDAY (I)) DOI=1 TO 7);

で次のように印刷できる.

68 45
64 53
55 44
58 44
62 40
51 39
54 37

データ名も一緒に読み書きされる転送 (Data-Directed Transmission)

この方法は，自分の名前そのものを伴ったデータの入出力に使われる.

例. A=38, B=105, D=3; なるデータを

GET DATA (A, B, C);

で読込むことができる.

フォーマットも与えられた転送 (Edit-Directed Transmission)

データの名前のリストとフォーマットのリストを与えて，入出力を行なうときにこの方法を使う.

例. GET EDIT (X, Y, Z(2), CODE)
(F(3), F(6, 2), E(7, 3), A(6));

レコード入出力には次の 10 種の方式がある.

(1) シークェンシャルでバッファーを使う入力
(2) シークェンシャルでバッファーを使う出力
(3) シークェンシャルでバッファーを使う UPDATE
(4) シークェンシャルでバッファーなしの入力
(5) シークェンシャルでバッファーなしの出力
(6) シークェンシャルでバッファーなしの UPDATE
(7) ダイレクト (ランダム) の入力
(8) ダイレクト (ランダム) の出力
(9) ダイレクト (ランダム) の UPDATE
(10) 排他的ダイレクト (ランダム) の UPDATE

5.5 インタラプト・オペレーション

プログラムの実行中にあるコンディションが起きると，通常の進行順序が中断 (interrupt) されて，特別の処置 (action) がとれるようになっている (その後，中断された処理を継続するのが普通だが，そうしないこともある).

インタラプトを生ぜしめるものとして次の6種類がある.

(1) 計算上のコンディション
CONVERSION
FIXEDOVERFLOW
OVERFLOW
SIZE
UNDERFLOW
ZERODIVIPE
(2) 入出力のコンディション
ENDFILE
ENDPAGE
KEY
NAME
RECORD
TRANSMIT
UNDEFINEDFILE
(3) プログラム・チェック用のコンディション
CHECK (ラベルや変数の名前)
SUBSCRIPTRANGE
(4) リスト・プロセシングのコンディション AREA
(5) プログラマの定めたコンディション
CONDITION (コンディションの名前)
(6) システム・アクションのコンディション
FINISH
ERROR

コンディションの中には，特に抑制しなければインタラプトをどこでも起こすのと (ブロックまたはステイトメントのラベルの前につく)，接頭辞 (prefix) によりブロック内か一つのステイトメントに限定されるものがある.

インタラプトが生じたときは，システムが標準の処置をとることができるが，これとは異なる処置をとらせたいときは，次のような形の ON ステイトメントを与える. それ以後に (しかもそのコンディションによるインタラプトが許された範囲で)，コンディションが起きると，ON ステイトメントで示された処置を行なう.

ON コンディションの名前 [SNAP] とるべき処置

または

ON コンディションの名前 SYSTEM;

SNAP をつけておくとレジスタの内容などプログラム・チェックのための情報を記録する. プログラマ

が定めた処置ではなくて SYSTEM と書くと，それ以後はシステムが標準とする処置がとられる．

例．
```
A : PROCEDURE;
      ⋮
    ON OVERFLOW
            BEGIN;
              ⋮
            END;
      ⋮
    ON OVERFLOW:
      ⋮
    ON OVERFLOW SYSTEM;
      ⋮
    END A;
```

5.6 プログラム・チェックのための便宜

プログラム・チェックのために，変数の変化やステイトメントの遂行を追跡（trace）するには，次のように書けばよい．

ON CHECK（変数やラベル）とるべき処置

たとえば，Aの値が変わるときと，SUB 1 なる手続やST 5 なるステイトメントが実行されるごとに，システム共通の処置（A とその値，SUB 1 と ST 5 の印刷）を行ないたいときには，

ON CHECK (A, SUB 1, ST 5) SYSTEM;

と書く．

変数の添字がその上限，下限を越えたときに，インタラプトさせたいときには，

ON SUBSCRIPTRANGE とるべき処置

を使う．

これらのほかに，プログラマがコンディションを定義して，インタラプト（programmed interrupt）を起こすことが可能である．それには，コンディションの名前（プログラマが勝手に定める）とインタラプトしたときにするべきことを，

ON CONDITION（コンディションの名前）とるべき処置

で与えておいて，コンディションが起きたことを示すには，

SIGNAL CONDITION（コンディションの名前);

と書く．

この方法はプログラム・テストにもかなり使えそうである．

例．
```
    ⋮
ON CONDITION (EXCEPTION 1) CALL
  PROCESS 1;
    ⋮
SIGNAL CONDITION (EXCEPTION 1);
    ⋮
```

5.7 作表のための便宜

処理した結果をストリーム入出力で印刷するのに，縦・横の様式を次のような用語で楽に指定できる．

X(w)……次におく空字の数
COLUMN(w)……次のデータをおく列
SKIP(w)……スキップする行数
LINE(w)……次に印刷する行
PAGE ……ページをあらためる

このほか，OPEN のオプションとして1行の字数と1ページの行数を，LINESIZE（式）と PAGESIZE（式）で指定できる．

5.8 リスト・プロセシング

リストプロセシングは情報検索，メッセージ交換，シミュレーションなどの高度のアプリケーションやコンパイラの制作に使用されている進んだテックニックで，これによりコア・スペースの弾力的利用が可能である．従来はこのための特別な言語やサブルーチンが必要であったが，PL/I では容易にリスト・プロセシングの記述が可能であり，ポインタとベース付きの変数（based variable）によって，配列，構造体やスカラのデータを一つのリストに組立てることができる．次のような構造のリストが取扱える．

一方向につながったリスト
両方向につながったリスト
リング構造
木構造

5.9 総称関数（Generic Function）

PL/I には数多くの組み込み（built-in）関数があるが，その大部分は総称関数の形態をとっている．複数個の関数が一つの名称で呼ばれ，変数の種類に応じてその中から対応する関数が自動的に選び出される．また関数値の性格もそれにより定まる．

たとえば，

FN=SIN (ARG);

と書いたときに，ARG が実数（x）ならそのまま sin x が計算されるし，ARG が複素数（$z=x+iy$）ならば

$$\sin z=\sin x\cosh y+i\cos x\sinh y$$

が計算される.

組み込みの総称関数は下に示すごとく ABS, MAX, MIN など簡単なものから三角関数やストリングの関数を含めて全部で 68 もある.

算術関数……ABS, MAX, MIN, ADD, MULTIPLY など 20 箇

浮動小数点関数……EXP(X), LOG(X) など 30 箇

ストリング関数……BIT, CHAR, HIGH など 10 箇

配列を演算する関数……SUM(X) など 8 箇

組み込み関数のほかに, プログラマが (GENERIC と書くことにより) 総称関数を書くことが可能である.

5.10 コンパイル時の便宜

通常のプログラミング言語は, 一つのレベルでしか考えられていないので, データに対する作用しか記述できない. しかるに, PL/I ではプログラムに対する作用をも示すことができる. すなわち, プログラムのステイトメントを修正したり, 作り出すことが可能である.

コンパイル時とはユーザのソース・プログラムが翻訳されて実行可能なオブジェクト・プログラムになる時を意味する. PL/I では, コンパイル時のステイトメントを調べてその指示どおりにソース・プログラムの修正を行なう**プロセサ**の段階と, それから出たものを実行可能のオブジェクト・プログラムにコンパイルする**コンピレーション**の段階の二つに分かれる.

プロセサにより, たとえば次のようなことが, 可能である.

(1) 変数名の変更のためや名前の付け方の便宜のためにソース・プログラムの修正.

(2) ソース・プログラムの節 (section) を条件により選んでコンパイルすること.

(3) ユーザまたはシステムのライブラリよりテキスト (一連のソース・ステイトメント) をソース・プログラムに組み入れること.

プロセサに読み込ませるものは**ソース・テキスト**と呼ばれ, コンパイル時のステイトメントは頭に % をつける. それによって出て来たものは**プログラム・テキスト**と呼ばれる.

変数や手続名が置換えの可能性ありと指定されているときは, それに出会うたびに可否を調べてから, 置換が行なわれる.

例 1.

```
% DECLARE A CHARACTER,
    B FIXED;
% A='B+C';
% B=2;
X=A;
```

と書くと, 次のステイメントが作り出される.

```
X=2+C;
```

コンパイル時のステイトメント, グループおよび手続は次のとおりである.

(1) DECLARE ステイトメント

コンパイル時の変数や手続名を指定し, それが置き換えられる可能性があることを示す.

(2) 置換ステイトメント

コンパイル時の変数=コンパイル時の式の形をとり, 前者を後者の値で置き換える.

(3) ACTIVATE と DEACTIVATE のステイトメント

変数名や手続名が置換可能と不能になることを意味する.

(4) GO TO ステイトメント

プロセサが次に行なうべきステイトメントを示す.

(5) 無 (null) のステイトメント

コンパイル時のラベルを挿入するのに使用する.

(6) IF ステイトメント

コンパイル時の式の値の如何により, プロセサが実行すべきことを示す.

IF コンパイル時の式 % THEN……〔% ELSE……〕;

(7) DO グループ

% 〔ラベル:〕……DO 〔i=m_1 TO m_2 〔BY m_3〕〕;

⋮

% 〔ラベル:〕……END 〔ラベル〕;

なる形をとり, ソース・プログラムの DO の使い方と同じ.

(8) INCLUDE ステイトメント

ライブラリから一連のソース・ステイトメントをプログラム・テキストに組み入れる.

(9) コンパイル時の手続 (procedure)

プロセサの段階で実行する手続 (関数のようなもの) を示す.

例 2.

```
% DECLARE I FIXED;
% I=1;
```

```
% LAB:;
Z(I)=X(I)+Y(I);
% I=I+1;
% IF I<=10 % THEN % GO TO LAB;
% DEACTIVATE I;
```

6. 既存の言語との比較

これまでの説明で，PL/I が全体としてすぐれた言語であることが明らかであるが，既存の言語の一つ一つと具体的に比較してみる．

6・1 FORTRAN との比較

PL/I がすぐれている点

カードの欄に拘束されない
注はどこに書いてもよい
ブロック構造をとっている
ステイトメント・ラベルに英数字
ラベルやデータとファイルの名前は 31 字まで
データ名が修飾できる
関数名に制限がない*
ラベルが変数，しかも配列として取扱える
関数サブプログラムとサブルーチン・サブプログラムの区別がない
サブルーチンの入口は一つ以上でもよい*
再帰的サブルーチンが書ける
数多くの総称関数
固定小数点を整数と限らぬ
複素数の演算*
ビットやキャラクタのストリングの演算
行列の全体や断面が一つの式で書ける
構造体としてのデータが扱える
ピクチャによるデータ・フォーマットの指定
複雑なファイルの記述
初期値の与えた方が簡単*
DIMENSION, INTEGER, DATA, COMMON などが一つのステイトメントで*
添字の下限が指定でき，負の整数でもよい
添字の数に制限がない
添字部の式の書き方が自由
行列の次元が可変*
式の右辺内に異種のデータを混入してもよい*
一つの式で同じ値を多数の変数に
行列式の加減算，スカラ量との乗算が一つの式で
論理演算のステイトメントの中にピリオドが不要
CONTINUE のステイトメントが不要
DO や IF のステイトメントがより強力
入出力データのフォーマットはなくともよい
多数の入出力方式
作表が簡単
コア・スペースの弾力的利用
リスト・プロセシング
インタラプト・オペレーション
プログラム・テスト用の便宜
マルチ・プログラミング
コンパイル時の便宜

注：*印のものは，類似の機能が FORTRAN Ⅳ には含まれている．

FORTRAN がすぐれている点

各ステイトメントの終りにセミコーロンは不要
ブランクをどこに幾つ置いてもよい
ソース・プログラムに書く字数が少い

6・2 ALGOL との比較

PL/I がすぐれている点

注はどこに書いてもよい
略語が数多く使える
予約語（reserved word）がない
手続に一つ以上の入口が可能
より多くの組み込み総称関数
複素数の演算可能
精度が任意に指定できる
ビットや文字のストリングの演算
行列の全体や断面が一つの式で書ける
構造体としてのデータの取り扱い
データ名が修飾できる
初期値の与え方が簡単
指定しなければ標準とみなしてくれる
行列式の加減算，スカラ量との積が一つの式で
ダイレクト・アクセス・ストーレジの使用
リスト・プロセシング
インタラプト・オペレーション
プログラム・テスト用の便宜
マルチ・プログラミング
コンパイル時の便宜

ALGOL がすぐれている点

式の中に条件節を組み込める（条件付算術式）
小文字や ¬，↑，〔，〕が使える
ブランクをどこにおいてもよい

6.3 COBOL との比較

PL/I がすぐれている点

手続とデータの記述を別々にまとめなくともよい
カードの欄に拘束されぬ
注はどこに書いてもよい
ステイトメント特にデータの記述が簡単
略語が数多く使える
予約語（reserved word）がない
必要なステイトメントの種類が少ない
ブロック構造をとっている
手続に一つ以上の入口が可能
再帰的ルーチンが書ける
浮動小数点の演算
可変長のビットや文字のストリング
ビットや文字のストリングの演算
添字の下限が指定でき，負の整数でもよい
添字の数に制限がない
テーブル（行列）の全体や断面が一つの式で
ピクチャがより強力
初期値の与え方が簡単
データを一々記述せぬともよいことがある
ラベルを変数として扱える
論理演算が可能
同一の値は一つの式で多数の変数に
行列式の加減算，スカラ量との乗算が一つの式で
強力な DO ステイトメント
コア・スペースの弾力的利用
インタラプト・オペレーション
プログラム・チェック用の便宜
マルチ・プログラミング
コンパイル時の便宜

COBOL がすぐれている点

英語そのものに近い
ENTER のステイトメントで機械語が組み込める
EXAMINE が使える

参考文献

1) J.W. Buckus ほか12名: The FORTRAN Automatic Coding System, Proceedings of the 1957 Western Joint Computer Conference, pp. 188～198.
2) Reference Manual 704 FORTRAN Programming System, Form C 28-6106, IBM Corp., New York, USA, 1958.
3) General Information Manual FORTRAN, Form F 28-8014, IBM Corp., New York, USA, 1961.
4) IBM Operating System/360 FORTRAN IV, Form C 28-6515-2, IBM Corp., New York, USA, 1964.
5) W.P. Heising: History and Summary of FORTRAN Standardization Development for ASA, Comm. ACM 7, 10 (1964).
6) D.D. McCracken: A Guide to FORTRAN Programming, John Wileys & Sons, New York, USA. 1962.
7) Preliminary Report—International Algebraic Language, Comm. ACM 1, 12, (1958).
8) A.J. Perlis and K. Samelson: Report on the Algorithmic Language ALGOL by the ACM Committee on Programming Languages and the GAMM Committee on Programming, Num. Math. 1 (1959), pp. 41～60.
9) P. Naur (Ed.): Revised Report on the Algorithmic Language ALGOL 60, Comm. ACM 6, 1 (1963),
10) D.E. Knuth ほか5名: A Proposal for Input-Output Conventions in ALGOL 60, Comm. ACM. 7, 5 (1964).
11) IFIP/WG 2.1: Report on Input-Output Procedures for ALGOL 60, Comm. ACM, 7, 10 (1964).
12) D.D. McCracken: Guide to ALGOL Programming, John Wileys & Sons, New York, USA, 1962.
13) COBOL Report to CODASYL (Initial Specs), Department of Defence, USA, 1960.
14) COBOL Report to CODASYL (Revised Specs.) Department of USA, 1961.
15) General Information Manual COBOL, Form F 28-8053, IBM Corp., New York, USA,
16) CODASYL COBOL Preliminary Edition (1964), CODASYL COBOL Committee, USA, 1964.
17) Draft of COBOL, Edition 65, CODASYL COBOL Committee, 1965.
18) D.D. McCracken: A Guide to COBOL Programming, John Wileys & Sons, New York USA, 1963.
19) Status Report of the Advanced Language Development Committee of the SHARE FORTRAN Project, March 1, 1964.
20) Report II of the Advanced Language Development Committee of the SHARE FORTRAN Project, June, 1964.
21) G. Radin and H.P. Rogoway: NPL: Highlights of A New Programming Language, Comm. ACM, 8, 1 (1965).
22) IBM Operating System/360 PL/I: Language Specifications, Form C 28-6571-2, 1966.
23) 竹下 亨: 新しいプログラミング言語 PL/I, 情報処理学会資料 5, 1965 年8月.

（昭和41年2月10日受付）