論　文

ミニコン用 PL/I サブセット: TPL-40*

大筆　豊** 石田和男*** 田中立二****

Abstract

A PL/I subset, called TPL-40, was developed for the mini-computer. In this paper, TPL-40 language specifications are described. TPL-40 was designed based on the following criteria:

1. TPL-40 is aimed at programming for scientific and process control applications.
2. The dynamic features, except a few for PL/I, are doropped. Because of this, object programs are expected to be very efficient.
3. Some features are added for structured programming, data integrity, debugging, etc.
4. TPL-40 is a multi-level language from which a suitable level can be selected for a particular application.

1. まえがき

PL/I は言語仕様として，あまりに多くの機能を取り込みすぎたため，言語自体に統一した思想を保つことが不可能であった．近年 ECMA[6] 等で言語仕様そのものの見直しがなされ，思想的にも統一した裏づけが保証されている．

しかし多くの機能を包含しているため，コンパイラが大きくなること，あるいはオブジェクト・コードの効率が，実行時，暗黙に使われるライブラリ・ルーチンをも考えると，低くなることは避け得ない．このため本来なら PL/I に吸収されるべき FORTRAN あるいは COBOL 等の使用比率が，PL/I に比べて依然として高いという現実がある．

ミニコン用の言語として PL/I のフルセットを実装することはほとんど不可能であり，たとえ可能としてもミニコンの大型機に対する位置づけを明確にすることができない．

本論文ではミニコン用に PL/I サブセットを選択した理由，および追加機能を付加した理由を述べることにする．

2. ミニコンの制約

ミニコンの言語を考えるとき，その応用分野の大きな比率を占めるプラントなどの制御用目的を無視することができない．オンラインで機器の制御を行うことからも，プログラムの実行時間に厳しい条件がある．また，ミニコンであるがゆえに，メモリ容量に厳しい制約がある．このため依然としてアセンブラによるプログラムの比率が非常に高い．

T: プログラムの実行時間

M: 実行に必要とされるメモリ容量

と定義するとき，プログラムの実行効率のめやすとして，

$$T \times M$$

の値が考えられ，この値をなるべく小さくする必要がある．

計算機システム全体に占める，ソフトウェアの費用は，すでにハードウェアの費用に比べて非常に大きいものになっている．それゆえ，ソフトウェアの生産性合理化を推進するためにも，高級言語によるプログ

---

* A PL/I Subset for the Mini-computer: TPL-40 by Yutaka OFUDE (Research and Development Center, Tokyo Shibaura Electric Co., Ltd.), Kazuo ISHIDA (Fuchu Works, Tokyo Shibaura Electric Co., Ltd.), and Tatsuji TANAKA (Heavy Apparatus Engineering Laboratry, Tokyo Shibaura Electric Co., Ltd.).

** 東京芝浦電気(株)総合研究所
*** 東京芝浦電気(株)府中工場
**** 東京芝浦電気(株)重電技術研究所

ラミングが必要となる．しかし高級言語を使用すると，$T \times M$ の値を大きくする要素を多分にもっている．

これら相矛盾する要素を満足させるようなミニコン用高級言語を開発する必要性があった．高級言語のもつ利点，すなわちプログラムの生産性合理化に役だつこと，および他機種への互換性をも考慮するとき，SP (Structured Programming) の手法を自然に取り込むことからも，PL/I を取り上げることになった．PL/I の持つ利点を生かしつつ，かつミニコン用の言語としての制約をも満足させるような PL/I のサブセットを決め，TPL-40 という名称で世に問うことにした．

3. TPL-40 の設計思想

TPL-40 の言語仕様を決定するにあたり，次のような方針を決めた．

(i) 言語として一貫した思想で統一すること．

(ii) 応用目的として，科学技術計算および応用システム記述用の言語とする．プロセス制御用あるいは他の目的で使用するとき，専用の文の追加，データに対する属性の追加，組み込み関数の追加，あるいは外部手続きにより実現させることとする．

(iii) 言語レベルを次のように定義した (**Fig. 1**).

レベル 1: PL/I の文法に完全に包含される部分であり，他機種の PL/I との互換性が保証され，かつ実行効率が非常に高い部分

レベル 2: 制御用目的やプログラムの生産性を向上させる目的などから PL/I の文法に追加された部分，あるいは効率を上げるため，文法の構文あるいは意味づけを変更させた部分

レベル 3: ミニコン用言語として機能的に大きくなるが，将来の拡張を考慮して，言語仕様あるいはコンパイラへの準備を考えておく部分

(iv) 実行時，動的に決まるような要素は極力削除しオブジェクト・コードの効率を上げるようにする．

(v) SP の思想を無理なく自然に表現できるような機能を取り入れる．

(vi) プログラムの作成が容易にできるような機能，信頼性を上げるような機能，および汎用性を保つような機能を入れる．

(vii) プロセス制御に適した機能を，専用の文あるいはタスク制御の機構として取り入れる．

(viii) マクロ的な機能，たとえば集合体全体の演算など，他の方法でも実現可能なものは原則として削除する．

(ix) 実行時の補助ルーチンとして暗黙のうちに使われるライブラリのルーチンは極力少なくする．

これらの要素は必ずしも PL/I という言語の思想と一致するものではないが，コンパイラのミニコンでの実装，あるいはミニコンでの実行を考えるとき，適切な基準であると確信する．

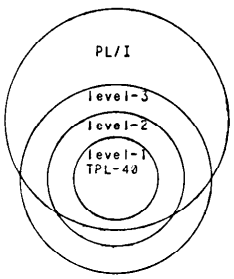

Fig. 1 Hierachy of TPL-40.

4. TPL-40 の位置づけ

TPL-40 は，ミニコン TOSBAC-40 (以下 T-40 と略す) 上でコンパイラを実装し，実行させることを目的として設計された．T-40 の言語体系は **Fig. 2** のような構成となる．ここで PL/40[9),10)] は N. Wirth 氏の PL 360[8)] を原型として作成された言語であり，「構造を持ったアセンブラ」としてとらえられる．ちなみにアセンブラで最適にコーディングされたものと比較するとき，オブジェクト・コードの大きさの冗長度は，PL/40 で約 7 %，TPL-40 あるいは FORTRAN

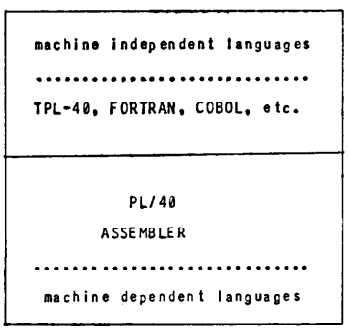

Fig. 2 Language system of TOSBAC-40.

などの高級言語で約 20% と見込まれている.

このことを背景に TPL-40 の位置づけを考えると次のようになる.

(i) アセンブラ: オンラインの割り込み処理, あるいはループの一番内側などで, 速度あるいはプログラム・サイズに非常に厳しい条件を伴うところの記述.

(ii) PL/40: 基本ソフトウェアの開発, あるいはハードウェアの詳細な記述を必要とする応用プログラムの開発.

(iii) TPL-40: 大規模なシステム, 互換性を必要とするプログラム, あるいは少々の冗長度が許されるようなプログラムの開発.

(iv) その他の高級言語: COBOL や RPG のように専用の目的を持った言語以外は, SP 的手法によるプログラムの生産性から考えても, TPL-40 あるいは PL/40 に吸収されると見込まれる.

5. TPL-40 の言語としての特徴

TPL-40 は 3 章で述べたように, その目的とする分野として科学技術計算用および制御用言語をめざしている. また SP 的手法を容易に記述できるような要素, そしてそれに付随して, プログラムの生産性, 信頼性, 保守性を向上させる機能, およびデバッグが容易にできるような機能を取り入れた.

言語の仕様は大別すると次の 3 部分よりなる.

(i) 手続き (アルゴリズム) 記述部

(ii) 入出力処理記述部

(iii) タスク制御記述部

タスク制御記述部は, T-40 のプロセス制御用目的に依存しているが, この部分を切り離して使うとき, 機械独立な PL/I サブセットとしてとらえることができる. 言語の構文については付録を参照されたい.

5.1 SP 的要素の取り入れ

PL/I では, 多重分岐はラベル変数を用いて行うが, 必然的に **goto** 文を仮定することになる. ラベル変数の動的な把握が困難になることから, 非常に危険である. 多重分岐の動的な動きを, 静的にも把握できるように **case** 文を導入した.

またループを実現するのに **do** 文があるが, 増分の正負によって制御変数を増減させ最終値と比較するようになっている. しかしこれらの処理を動的に行わせるのは, オブジェクト・コードの増大, および実行効率の低下をきたすので, 増分は常に正をとるとし,

```
CASE I OF
  /* I=1 */ statement-1;
  /* I=2 */ statement-2;
    ---
  /* I=n */ statement-n;
OTHERWISE
          statement;

DO I=1 TO 100 BY 1;
   ---
END;

DO I=100 DOWNTO 1 BY 1;
   ---
END;
```

Fig. 3 Example of additional control structures.

do 文の増分指定に **to** 以外に **downto**[11] を導入した (**Fig. 3**).

5.2 動的に決まる要素の大幅な削除

PL/I では実行時, 動的に決まる要素が非常に多い. たとえば, 次のようなものがある.

- ブロック, あるいは手続きの出入口での, メモリの確保, および開放.
- データの型および精度の自動変換.
- 集合体データの適合化や, 集合体データ全体の演算.

TPL-40 では, 特に許すもの (たとえば, 再入可能な手続きを可能にするための, **auto** 属性をもつ記憶域) を除き削除した. ミニコンでの実行を考えるとき, これら実行時に決まる部分は, 暗黙のうちに使われる実行時処理ルーチンの補助, あるいはオブジェクト・コードの増加をまねき, ミニコンにとって負担の大きなものになるからである.

5.3 信頼性を向上させる要素の導入

多人数で大きなシステムを作成するとき, ある共通変数をどの部分が壊わしているかをつきとめるのが, 非常に困難な場合がある. トップダウンでシステムを作成してゆくとき, 参照してもよいが, 変更してはならない共通変数が明確になる. これらの変数を保護するため **protect** 文を導入した. ここで指定された変数は, その有効範囲で, 式の左辺に現われるなど, 変更される可能性のあるとき, コンパイル時に警告が出される (**Fig. 4**).

```
SUB:PROC(A,B,C);
    PROTECT A,B;    /* A,B, and X are protected */
    DCL X,Y,Z;      /* in procedure SUB.        */
    PROTECT X;
       ---
      BEGIN;
       PROTECT Y;   /* Y is protected in this   */
          ---       /* internal block.          */
      END;          /* C and Z are not protect- */
       ---          /* ed in procedure SUB.     */
END;
```

Fig. 4 Example of protection mechanism.

ファイル・システムには，参照のみ可，変更も可といったキーがついており，実行時，動的に保護機能が働く．本方式では，プログラム作成時，あるいはコンパイル時，静的に保護機能を働らかせることになる．また保護機能の管理はプロジェクト・リーダあるいはプログラマ自身にまかされている．

5.4 デバッグを容易に行わせる機能の導入

虫のないプログラムを作成するのは，現在のソフトウェア生産技術では，不可能といえよう．虫とりに必要な情報は，プログラムの動的な制御の流れ，およびある特定の変数にどんな値がセットされるか，ということである．

制御の流れを知るには，どの手続きが呼ばれるか，あるいはどの名札に制御が移るかで把握できる．変数の値が変るのは，式の左辺に現われるとき，あるいは手続きの引数として渡されるときである．いずれにしても虫とりに必要な情報は，名前の形で把握できる．

TPL-40 では，**debug** 文を導入し，そこで指定された名前に特別な属性を与えることにした．名札の場合，そこに制御が移ったという事実を，手続きの場合 **call** 文の前後すなわち呼ぶ前と呼んだ後に，その手続きが呼ばれ，かつ戻ってきたという情報を，変数の場合その値が変化する前後の情報を，実行時出力することにした（**Fig. 5**）．

この効果を取り除くには，**debug** 文を抜き去ること，あるいはコンパイル時，実行時のオプション指定により可能である．

5.5 汎用性を保つ機能の導入

PL/I の場合，ある手続きが呼ばれたとき，その手続きに必要なメモリ領域は，引数の指定などで動的に確保することが可能である．TPL-40 ではこのような動的な機能を削除したため，同様の機能を持たせる必要から，**constant** 属性の意味づけを拡張し，静的な

```
BEGIN;
  DCL X,Y,Z;
  SUB:PROC;
      ---
    END;
  DEBUG X,L;
    ---
  X = --- ;          /* output of value X   */
  BEGIN;
    DEBUG SUB ,Y;
      ---
    X = --- ;        /* output of value X   */
    Y = --- ;        /* output of value Y   */
    CALL SUB;        /* output the fact of  */
    GOTO L;          /* transfer to and re- */
      ---            /* turn from SUB       */
  END;
    ---
L: --- ;             /* output the fact of  */
  ---                /* transfered to L     */
END;
```

Fig. 5 Example of debugging mechanism

```
BEGIN;
   DCL I CONST(5);
   DCL A(I,I);
     ---
   PUT EDIT(A)((I)F(10,5));
     ---
END;
```

Fig. 6 Example of named constant.

意味での可変性を保証することにした．

すなわち，PL/I ではメモリ領域などの可変性が実行時，動的に可能であるが，TPL-40 では **constant** 文の書き換えによる，コンパイル時の静的な可変性を可能ならしめている（**Fig. 6**）．

5.6 タスク制御機能の導入

TPL-40 は，ミニコン用の汎用高級言語を目的として設計されたが，ミニコンの応用として大きな割合を占めるプロセス制御用の記述言語としての位置づけを無視することができない．特にタスク間の制御，イベントの処理を記述する機能は，必須の条件である．

TPL-40 では，タスク間の制御，イベントの処理などを，セマフォアの概念で統一的に取り扱うことにした．そしてセマフォアの管理を，リソースを要求した方で行うか，要求されたリソース側で暗黙に行うかによって，**task** 文，**event** 文，あるいは **semaphore** 文の区別を行うことにした．

タスク，イベント，あるいはセマフォアを表わす名前はシステム側で保持する必要があり，オペレーティング・システムに依存する．**task** 文，あるいは **event** 文でリソースを使用するとき，並列処理が可能であり，**semaphore** 文の場合指定されたリソースの処理が完了するまで待つことになる．

5.7 再入可能，再帰呼び出し手続きの導入

ミニコンなるがゆえの制約のため，メモリの共有を実現すべく，再入可能な手続きは不可欠の要素である．論理的には，再入可能な手続きを呼び出す各タスクにその原型がコピーされ取り込まれると考えられる．それゆえ，各タスクに必要とされる変数領域の確保のため **auto** 属性を持つ変数を許すことにした．

再帰呼び出し手続きは，システム・プログラムの作成などで，是非必要な要素である．しかしメモリの動的な確保，開放などが要求され，実行効率は低下し必要なメモリの大きさをあらかじめ予想するのが困難である．これらの理由から，再帰呼び出しの手続きを乱用されるのは，特にミニコンでの実装を考えるとき，非常に危険な要素を持っている．TPL-40 では，これらのことを考慮して，制限した形で取り入れることにした．

6. むすび

ミニコン用 PL/I サブセットとしての位置づけからTPL-40 の言語仕様をまとめた．ミニコンでのコンパイラの実装あるいは実行を考え，削除した機能，意味づけを変更した機能，あるいは T-40 のハードウェアやオペレーティング・システムに依存せざるを得ない機能についても付言した．

また SP 的なプログラムの生産性を向上させる機能や，保守および信頼性を向上させる機能，あるいはプログラムのデバッグを容易ならしめる機能を導入した．

著者らは，このバージョンをもとに，さらに TPL-40 のミニコン用言語としての機能拡張に取り組んでゆく予定である．たとえば TPL-40 の言語仕様として，PL/I に完全に含まれる部分，あるいは将来の拡張を考慮して，レベル1，レベル2，およびレベル3の区別をもうけている．

この言語仕様をもとに，レベル2とレベル3の間の規模で，(PL/40 を記述言語とした) T-40 上で動くセルフ・コンパイラ，および (PL/I を記述言語とした) クロス・コンパイラを作成した．

TPL-40 の言語仕様は，ミニコン用の言語としてだけでなく，大型機用のコンパクト PL/I としても十分実用に耐えると確信している．

謝辞：本研究の機会を与えて頂いた，当社府中工場松本吉弘主幹，細田泰雄部長，総合研究所高橋義造研究主幹，そして有益な助言を頂いた京都大学池田克夫助教授に深謝するものである．

参考文献

1) 石田，田中，大筆：ミニコン用 PL/I サブセット—制御用—，情報処理学会第 18 回プログラミングシンポジウム報告集，pp. 101～106 (1977).
2) 梅田，石田，田中，堀，大筆：ミニコン用 PL/I サブセット(制御用)言語の開発思想および仕様，昭和 52 年度電子通信学会総合全国大会予稿集，p. 1318 (1977).
3) 田中，堀，石田，梅田，大筆：ミニコン用 PL/I サブセット (制御用) SP および信頼性面からみた特徴，昭和 52 年度電子通信学会総合全国大会予稿集，p. 1319 (1977).
4) TOSBAC-5600 プログラミング説明書，PL/I 入門書，東京芝浦電気株式会社，No. 56A PR 44A.
5) TOSBAC-5600 プログラミング説明書，PL/I 操作編，東京芝浦電気株式会社，No. 56A RP 46A.
6) BASIS/1-10, ECMA. TC 10/ANSI. X3J1 (1973).
7) PL/I 標準化に関する調査 (BASIS/1-12)，日本電子工業振興協会，51-C-307 (1976).
8) N. Wirth: PL 360 A Programming Language for the 360 Computers, J. ACM, Vol. 15, No. 1, pp. 37～74 (1968).
9) 大筆，石井，森本，滝：ミニコン用システム記述言語の作成，昭和50年度情報処理学会第16回全国大会予稿集，pp. 341～342 (1975).
10) 市瀬，上田，石田，大筆，小松：TOSBAC-40 用 SP 向システム記述言語・PL/40，昭和52年度電子通信学会全国大会予稿集，p. 1281 (1977).
11) N. Wirth: The Programming Language Pascal, Acta Informatica 1, pp. 35～63 (1971).
12) P. Elzer ほか：PEARL マニュアル，KFK-PDV 1.
13) 工業用コンピュータソフトウェアの標準化動向(第3版)，日本電子工業振興協会，49-A-83(1974).
14) PL/I の標準化に関する調査 (PL/I 専門委員会報告書)，日本電子工業振興協会，52-C-327(1977).
15) Department of Defense Requirement for High Order Computer Programming Languages, "IRONMAN", 14 January 1977.
16) IBM System/7 Application Program Generator. (APG/7) Program Reference Manual, SH 20-9502-00.

付 録

```
******** SYNTAX SPECIFICATION OF TPL-40 ********

 NOTATION : (A)      : A
            A...     : A or A(A...)
            A,B...C  : permutation of A,B,...,C
            [A]      : A or null
            A|B      : A or B
            A[B]...  : A or AB(A[B]...)
            A/B      : Aor B or AB

 COMMENT : no mark --- level-1
           #       --- level-2
           ##      --- level-3

<program> ::= <main>|<external procedure>
<main> ::= <entry>:PROC OPTIONS(MAIN);
   ((<label>:)<statement>;)...END[<entry>];
<external procedure> ::=
   <entry>:PROC[(<parameter>[,]...)]
   [RETURNS[(<data attribute>)]]
   ##[OPTIONS[(REENTRANT|RECURSIVE)]]
   ((<label>:)<statement>;)...END[<entry>];
<statement> ::= <executive statement>|
   <non-executive statement>

<executive statement> ::= <begin block>|<group>|
   <if statement>|#<case statement>|
   <single statement>
<begin block> ::=
   BEGIN;((<label>:)<statement>;)...END[<label>]
<group> ::= DO[<do spec>/<while option>];
   ((<label>:)<executive statement>;)...END[<label>]
<if statement> ::= IF<expression>
   THEN<executive statement>
   [;ELSE<executive statement>]
#<case statement> ::= CASE[<expression>] OF
   (<executive statement>;)...
   OTHERWISE<executive statement>
<single statemtnt> ::=<goto statement>
   <assignment statement>|
   <call statement>|<return statement>|
   <allocate statement>|<free statement>|
   #<tasking statement>|<I/O statement>|
   <null statement>
```

```
<goto statement> ::= GOTO<identifier>
<assignment statement> ::=
   <reference>=(<expression>|<reference>)
<call statement> ::= CALL<entry>
   [(<expression>|<reference>)(,)...]
<return statement> ::= RETURN[(<expression>)]
<allocate statement> ::=
   ALLOC<identifier>SET[(<pointer>)]
<free statement> ::= FREE<pointer>-(<identifier>
<null statement> ::=

<non-executive statement> ::= <procedure block>|
   <declare statement>|#<debug statement>|
   #<protect statement>|
   <format>:<format statement>
<procedure block> ::=
   <entry>:PROC[(<parameter>(,)...)]
   [RETURNS(<data attribute>)];
   [(<label>:)<statement>;]...END[<entry>];
<declare statement> ::= DCL<declaration>(,)...

<declaration> ::=
   [(<integer>)](<identifier>|##[(<declaration>(,)...)]
   [((##<bound>:)<bound>)(,)...)](<attribute>...)
<attribute> ::= <data attribute>|
   STATIC|BASED|##AUTO|PARM|BUILTIN|
   #CONSTANT[(-)<literal constant>)]|VARIABLE|

   #DEF<basic reference>|INIT|EXT|#GLOBAL|
   INIT((([<constant>)...](-)<constant>))(,)...)|
   OPTIONS[MAIN|##REENTRANT|##RECURSIVE)|
   <I/O attribute>|#<tasking attribute>
<data attribute> ::= FIXED[(<constant>)]|
   FLOAT[(<constant>)]|
   BIT[(<constant>)]|CHAR[(<constant>)]|PTR|
   ENTRY  (##[(<parameter descripter>(,)...)]|
   RETURNS[(<data attribute>)]|<I/O attribute>|
   <tasking data attribute>

#<debug statement> ::= DEBUG<identifier>(,)...
#<protect statement> ::= PROTECT<identifier>(,)...
<expression> ::= ((-|¬)<operand>)(<operator>)...
<operand> ::= <reference>|<constant>|
   (<expression>)
<operator> ::= +|-|*|/|**|
   ||&|=|¬=|>|<|>=|<=|¬>|¬<
<reference> ::= (<pointer>-)(<basic reference>

<I/O statement> ::= <stream I/O statement>|
   <record I/O statement>
<stream I/O statement> ::= (GET|PUT)
   ((FILE[(<file>)]|STRING[(<reference>)]
   EDIT[(<data list>)][(<format list>|<remort format>)]|
   #LIST[(<data list>)])
<data list> ::= ((<reference>|<expression>|
   [(<data list>)DO
   (<do spec>/(<while option>))])(,)...
<format list> ::=
   (<format item>|<format iteration>)(,)...
<format iteration> ::=
   (<integer>|[(<constant>)])(<format item>|
   [(<format list>)])

<format item> ::= <data format>|
   <control format>
<remort format> ::= R[(<format>)]
<format statement> ::= FORMAT[(<format list>)]

<record I/O statement> ::= <read statement>|
   <write statement>|<open statement>|
   <close statement>
<read statement> ::= READ(<file option>.
(<into option>|#<ignore option>).
   [(<key option>)].#[(<event option>)])
<write statement> ::=WRITE(<file option>.
   <from option>.(<key option>.#[(<event option>)])
<open statement> ::= OPEN(<file option>
<close statement> ::= CLOSE(<file option>
<I/O attribute> ::= INPUT|OUTPUT|UPDATE|DIRECT|SEQL|
   ENV[(<identifier>|<constant>)
   [,(<identifier>|<constant>)]]
<I/O data attribute> ::= FILE|FORMAT

#<tasking statement> ::= <task statement>|
   <event statement>|
   <semaphore statement>
#<task statement> ::= TRNON[(<task>)]
   [AT(<time constant>)]|
   START[(<task>)]
   ([AFTER(<time constant>).#[(<event option>)])|
   LINK[(<task>)]|STOP|DELAY(<time constant>
#<tasking attribute> ::=
   ENV[(<identifier>|<constant>)]
#<tasking data attribute> ::= TASK|SEMAPHORE
#<event statement> ::= WAIT[(<event>)]
#<event> ::= <task>|<file>
#<semaphore statement> ::=
   RELEASE[(<semaphore>)]|REQUEST[(<semaphore>)]
```

(昭和52年8月31日受付)
(昭和52年9月20日再受付)