制御用ストラクチャードプログラミング言語 SPL の工業用問題向言語への応用†

浜田亘曼†† 平沢宏太郎††
高藤政雄†† 林利弘†††

計算機制御の分野では，ソフトウェア生産性を向上させる有力な方法として高級言語化と問題向言語 (POL: Problem Oriented Language) 化の2つが考えられる．すなわち，汎用的な高級言語を用いて信頼度の高いプログラムを作成するか，特定応用分野の標準化された要求をプログラム仕様として記述させる機能を有する問題向言語を利用するかである．先に報告されている制御用計算機言語 PCL の上位言語である SPL (Software Production Language) は，上述の2つの側面を同時に満し得るように設計されている．

本論文では SPL の POL への応用について述べている．特に SPL の特徴的な機能である，手続きの構造化制御機能，手続きのインライン展開機能，豊富な手続き参照機能，PL/I のそれを高信頼化の面で強化したコンパイル時機能等について論じている．また，SPL の電力系統制御への応用例を用いて，その効果を論じている．

1. まえがき

マイクロプロセッサ，LSI の技術革新に伴い，制御用計算機の適用分野が近年ますます拡大しつつあり，信頼性の高い，保守のしやすいソフトウェアを生産性よく作成する道具の開発が急がれている[1)~3)]．

ソフトウェアの生産性を向上する有力な方法として，汎用の高級言語により信頼性の高いプログラムを作成する手法と，プログラム仕様の標準化を進め，これを記述性よく表現させ，対象のテーブルとプログラムを編集し得る問題向き言語を整備する手法が考えられる．高級言語はプログラムの記述水準を高級化し，記述量や思考量を減らすことにより，また，問題向言語はそれぞれのアプリケーションに特有の表現で処理を記述することによりソフトウェアの生産性，信頼性，ドキュメント性の向上を図るものである．

筆者等は，高級言語と問題向言語の両言語を統一的にサポートできる Software Production Language (SPL) を開発し，その概要をすでに報告したが[4)~9)]，本論文では，問題向言語としての面に焦点を絞り，その機能，適用例の詳細について述べる．

計算機言語の分野では，構造化された制御機能と新データ形定義機能を実現している言語 PASCAL[10)]，構文上のチェックを行っていないが，コンパイル時機能を高級言語で実現している PL/I[11)]，計算機のためのオブジェクト・コードを直接生成できないが，プログラムの仕様を記述できる言語 PDL[12)] 等が一般の高級言語用として個々に開発されている．これに反し，SPL は，構造化された制御機能，手続きのインライン展開機能，コメント風の手続参照機能，手続きおよびデータの階層化およびグループ化機能，マクロ参照時の実引数と仮引数間のデータ属性の不一致等の文法的誤り検出を含むコンパイル時機能等，高級言語と問題向言語に必要な機能を制御用として統合し，しかもコンパクトに実現している点で，他に例のない特徴あるシステムである．特に制御用として必要となる，ビット処理機能や，スーパーバイザ呼び出し機能は，従来より用いられていた制御用計算機言語 PCL[14)] の上位言語とすることにより実現した．

ソフトウェアの生産性を向上させるには，本論文で述べる SPL のような高級言語を利用することに加えて，会話形でプログラム作成を行えるよう計算機および CRT 端末を整備すること，増大するプログラム処理を安価なプロセッサ群に分散させ，かつ簡単なプログラム構造を許すような分散形アーキテクチャを確立すること等，総合的な施策がさらに必要である．

---

† Using SPL as a Problem Oriented Language in Industrial Control Applications by NOBUHIRO HAMADA, KOTARO HIRASAWA, MASAO TAKATO (Hitachi Research Laboratory, Hitachi Ltd.), and TOSHIHIRO HAYASHI (Omika Works, Hitachi Ltd.).

†† 日立製作所日立研究所
††† 日立製作所大みか工場

2. SPL の概要

本章では，高級言語，問題向言語の両言語を統一的にサポートできる SPL の概要について述べる．

2.1 SPL の開発指針

SPL の開発指針は次のように要約される．

（1） ストラクチャードプログラミング，トップダウンプログラミング等高級言語機能を効果的に実現すること．

（2） 問題向言語を各分野別に拡張性良く，しかも生産性良く作成するには，特定用途のプログラム作成に比して，より複雑なプログラムを効率的に作らねばならず，上述(1)が必要となる．さらに，エンド・ユーザの記述量の減少およびオブジェクト効率の向上を計る．

以上を考慮して，SPL では**図1**に示すような特徴的な機能を充実させている．

2.2 SPL の機能

図1の各機能の概要は下記のとおりである．

（1） 構造化された制御機能

手続きの制御文として，FORTRAN 等の GO TO 文，IF 文等にみられる，プログラム中の任意の点へ制御点を移動できる制御文を禁止し，構造化された下記の制御文を導入する．これにより良い構造をしたストラクチャードなプログラムの作成が可能となる．

・複合ブロック文　・選択ブロック文
・反復ブロック文　・ブロック脱出文

（2） 手続きのインライン展開等の機能

従来のサブルーチン（外部サブルーチンと呼ぶ）リンケージではプログラム実行時のオーバヘッドが多いため，複数の手続き呼び出し文の列を，新しい手続きとして定義することが困難であるが，インライン展開ではこれが容易となり手続きの階層化（トップダウンプログラミング）が可能となる．SPL ではこの他内部サブルーチンと呼ばれる簡易なリンケージ形式を取るサブルーチンも用意されており，プログラムの定義時に上記3つのいずれかを指定できるようになっている．

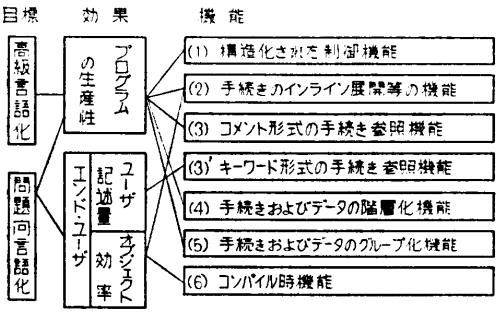


図 1 SPL の開発指針と機能
Fig. 1 Design policy and functions of SPL.

（3） コメント形式，キーワード形式等による手続き参照機能

プログラムの生産性，ドキュメント性をあげるためには，プログラムが分りやすいということが肝要である．このためには，プログラミングの知識のないプロセスエンジニアでも容易に理解できる自然言語風のコメント形式の手続き参照機能が高級言語においても，問題向言語においても重要となる．特に，各手続き呼び出し文にアプリケーション分野特有の名前をつけることにより問題向の表現が可能となる．

また，手続き中の引数が多く，かつ省略が多い場合には，キーワード形式の手続き参照機能が便利である．SPL では，1つの関数定義に対して，両形式のいずれを用いても参照が可能である．

（4） 手続きおよびデータの階層化機能

トップダウンプログラミングにおいては，手続きの階層化とそれに対応したデータ型の段階的詳細化が必要である．

手続きの階層化は(3)項のコメント形式の手続き参照機能により，またデータ型の階層化は新データ型の宣言機能により可能となる．

プログラムの階層についていえば，定義・参照の関係で結ばれる階層の他に，多くの手続きやデータを論理的に分類して得られる意味的な階層が存在し得る．SPL では後者の階層の意味でのグループ化を支援する機能(5)項が用意されている．SPL でいうグループの階層は，主として，データ定義における変数のスコープ（グローバル，ローカルの度合い）に依存して定義される．したがって，手続きのグループはデータのグループに従属して，階層的に位置づけられている．

（5） 手続きおよびデータのグループ化機能

大規模なプログラムの設計効率を向上させる上で，手続きおよびデータのグループ化機能が有用である．このため，SPL では階層ごとの手続きの集合，データの集合をそれぞれ処理モジュール，環境モジュールとして定義することが可能である．なお，SPL でいう処理モジュールは，PARNAS[13] の定義するモジュールに近いものから，従来のサブルーチンの集合にす

ぎないものまで含めて定義することができる．一方環境モジュールは，システムで共通な宣言文（主に変数宣言）を階層的にグルーピングして，モジュールとして管理し，宣言の重複をさけようとするものである．

（6） コンパイル時機能

オブジェクトプログラムの編集および最適化を実現するための機能で，汎用的な標準パッケージから各適用プラントごとに必要な部分をコンパイル時に切り出し編集する％制御文，類似の手続きを同一の関数で定義しておき，コンパイル時に，この類似手続きの処理を行うために使用される％組込み関数，オブジェクトの最適化のために，コンパイル時に各種の演算を行う％代入文，コンパイル時に入力データの合理性チェックの報告を行うための％出力文等が基本となる．

3. 問題向言語のための機能

以下，コメント形式，キーワード形式の手続き参照機能，手続きのインライン展開機能，コンパイル時機能について，問題向言語への適用を中心に，詳細に説明する．

3.1 コメント形式，キーワード形式の手続きの定義，参照機能

SPL では，自然言語に近い記述を許すコメント形式，キーワード形式の手続き，参照を可能にしているため，次節の例題から分かるように，プログラムの記述性が向上し，プログラム自身を仕様書として利用することも可能である．

表1は，コメント形式とキーワード形式の手続きの参照例と定義例および各形式の特徴についてまとめたものである．コメント形式は記述性の向上を期待できるので，問題向の記述に，またキーワード形式は手続中の引数が多く，かつ省略する場合に有効なため，汎用的標準パッケージの作成に便利である．

3.2 手続きのインライン展開等の機能

手続きのインライン展開機能は，コメント形式の手続き参照機能と一体化して，問題向言語をサポートする機能であり，サブルーチンリンケージの場合に比較してオブジェクトの処理速度の向上を期待できる．

図2は，インライン展開機能の具体例を説明するための図，**図3**はインライン展開の具体例を示した図である．

コンベア1上の素材A，Bとコンベア2上の素材 CD をコンベア3上に ACDB の順に並びかえるロボットR1の操作は，コメント形式の手続きを使用して図3のごとく記述している．図3のメインファンクションにおけるコメント形式の各手続は対応する

表 1 手続き定義と参照

Table 1 Procedure definition and reference.

項目	参照の例（エレメントAをスタックBにプッシュする）	特徴	定義の例（先頭の部分）
コメント形式	PUSH DOWN (A) TO (B)	・自然言語に近い記述を許す ・従って記述性が高い	**function** PUSH DOWN (ELEMENT: **real**) TO (STACK: **array** (100) : **int**) .....
キーワード形式	PUSH (ELEMENT=A, STACK=B)	・手続き中の引数が多く，かつ省略が多い場合に便利 ・従って，汎用的な標準パッケージを作成するのに便利	

注．パラメータに default 値を用いるには INIT 節による．例えば (*P*: **real init** (10.0)) とする．

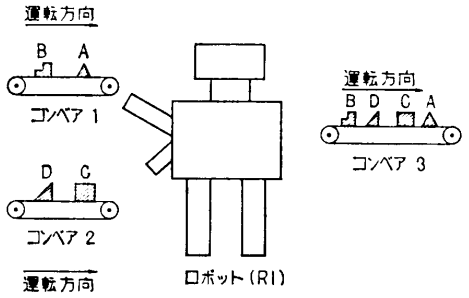
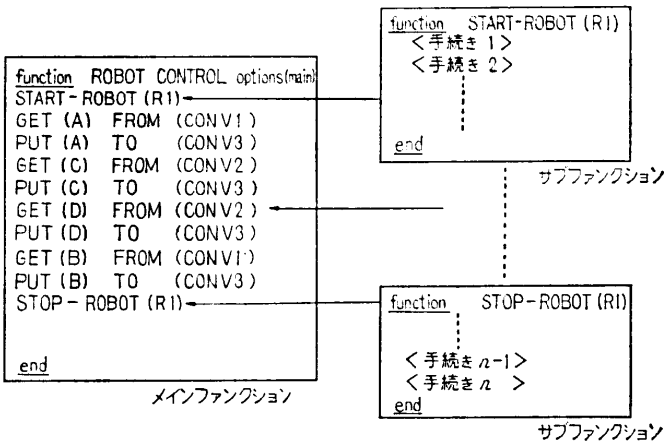


図 2 ロボットによるコンベヤ群の運用制御

Fig. 2 Conveyor line operation by robot.



オプション部が略された時には options (open) と解釈される

図 3 インライン展開の例

Fig. 3 Example of inline expansion.

各サブファンクションでインライン展開される．このようにインライン展開とコメント形式の手続きを組み合せることにより，オブジェクト効率のよい，トップダウン記述の可能な，分りやすい問題向言語の作成が可能になる．

3.3 コンパイル時機能

コンパイル時機能は

（1） 汎用的な標準パッケージから各適用プラントごとに必要な手続きのみをとり出し，コンパクトなオブジェクトを作成するため

（2） コンパイル時実行組み込み関数等と組み合せて，各適用プラントごとに適応したオブジェクトを作成するため

（3） コンパイル時に各種の演算を行い，オブジェクトのコンパクト化，処理速度の向上を計るため

（4） コンパイル時にプロセス変数の合理性チェックを行うため

等に使用される．

上記（1）～（4）項を実行するため，コンパイル時実行文として，％制御文（％選択文，％反復文），％代入文，％出力文があり，また，コンパイル時実行のためのコンパイル時変数の宣言を行う％宣言文がある．

表2は，コンパイル時実行文，コンパイル時宣言文の具体的適用例をまとめたものである．

％選択文の例では，関数Fの実引数が0の時〈手続き〉$_1$が，0でない時〈手続き〉$_2$がオブジェクトとして選択されることを示している．

％反復文の例では，関数Fの実引数に，関数A，Bを指定するとA，Bのオブジェクトが，また関数Fの実引数に関数Aを指定するとAのみのオブジェクトが生成されることを示している．このように，類似の手続きを同一の関数で定義しておき，コンパイル時にこ

表2 コンパイル時機能
Table 2 Compile time facility.

項目			機能	適用例		
				定義形態	参照形態	コンパイル時出力の形態
％実行文	％制御文	％選択文	汎用的な標準パッケージから各適用プラントに必要な手続きのみをとり出し，コンパクトなオブジェクトを作成する	**function** F(P: **int**); ％ **if** P. EQ. 0 **then** 〈手続き〉$_1$; **else** 〈手続き〉$_2$; **end**; **end** F;	≀ $F(0)$; ≀	≀ 〈手続き〉$_1$ のオブジェクト ≀
		％反復文	例えば，コンパイル時実行組み込み関数SIZEと組み合せ，各適用プラントに適応したオブジェクトを作成する	**function** F (P: **list** (EFFECT)); ％ **var** I: **int**; ％ **for** I=1, ¥SIZE **repeat** $P(I)$; **end**; **end** E;	≀ $F(A, B)$; ≀ $F(A)$; ≀	≀ A; B; のオブジェクト ≀ A; のオブジェクト ≀
	％代入文		コンパイル時に各種の演算を行い，オブジェクトのコンパクト化，処理速度の向上を図る	**function** F(P: **int**); ％ **var** A: **int init** (0); **var** X: **int**; ％ A=P*2+1; X=A; **end** F;	≀ $F(1)$; ≀	≀ X=3; のオブジェクト ≀
	％出力文		コンパイル時にプロセス変数の合理性チェックを行う	**function** F (P: **int**); ％ **if** P. GT. 100 **then** ％ WRITE ('UPPER LIMIT OVER'), **else** ％ WRITE (NORMAL'); **end**; **end** F;	≀ $F(1)$; ≀	≀ 文字例 NORMAL を出力 ≀
％宣言文			コンパイル時実行文で使用するコンパイル時変数の宣言を行い，上記コンパイル時機能を実行する	≀ ％ **var** A: **int init** (10); ≀	≀ ％ B=A; ≀ C=A; ≀	≀ C=10; のオブジェクト ≀

の類似手続きの処理を行って各適用プラントに適応したオブジェクトを生成することが可能である.

%代入文の例では，プロセス変数 p の加工処理 $(2p+1)$ をコンパイル時に前もって行い，効率の高いオブジェクト $X=3$ を生成している.

%出力文の例では，プロセス変数 p が値 100 を超過する場合には，'UPPER LIMIT OVER' のメッセージを出力し，コンパイル時にプロセス変数の合理性チェックの結果を報告できることを示している.

このように，コンパイル時機能は，汎用的な標準パッケージを作成するための道具として特に威力を発揮する.

4. 問題向言語への応用

本章では，SPL の問題向言語への応用を電力系統制御を例にとって示す.

前述したように，SPL の問題向言語としての機能は図1に要約されているが，以下では図中の(2)，(3)，(4)について，これらが具体的にどのように効力を発揮するかを**図4**の例で示す.

図4は，電力系統のラインディスパッチングに関するもので，遮断器 $(X, Y, \cdots, Z)$ の両端の電圧をチェックし，両端の電圧に異常がなければ同期化処理を行い，その後遮断器を閉じる例を示している.

なお，同期化処理は，微少ステップの処理を実行するごとに同期化完了の判定を行う方式を採用している.

図5は，PCL 言語[14]（フォートランを制御用に改良し，ビット処理，構造体の宣言等が可能な言語）で上記ラインディスパッチング処理を記述した例，**図6**は，SPL 言語で記述した例である.

従来の PCL 言語では，プログラムの記述性が低いため，数多くのコメント文を必要とし（図5の例ではコメント文は記述されていない），したがって，コーディング量がかなり多くなる．また，新しい種類の遮断器が新設されるたびに，図5Ⓨ Ⓩのごとくごく1部分しか異ならないプログラムを追加する必要があり，プログラムの保守上好ましくない.

一方 SPL では，図6の SWITCH-ON ルーチン

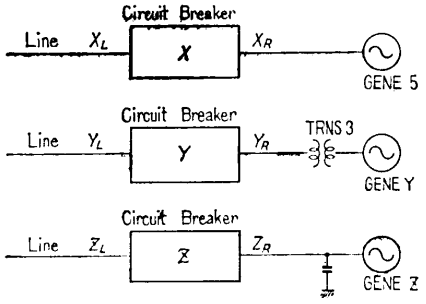


図4 電力系統
Fig. 4 Power system.

```
      SUBROUTINE DISPAG(CBX)/* CBX with generator */
      IF (STATE1(VOLT,CBX,RIGHT)) GO TO 500
      IF (STATE1(VOLT,CBX,LEFT)) GO TO 100
      STOP
100   DO 400 I=1,MAX
      IF (STATE2(SYNC,CBX)) GO TO 500
      CALL SYNCG(GENE5)
400   CONTINUE
      STOP
500   CALL EXCITE(BRAKER,CBX)
      END                                              Ⓧ

      SUBROUTINE DISPAT(CBY)/* CBY with Transformer */
      IF(STATE1(VOLT,CBY,RIGHT)) GO TO 500
      IF(STATE1(VOLT,CBY,LEFT)) GO TO 100
      STOP
100   DO 400 I=1,MAX
      IF(STATE2(SYNC,CBY)) GO TO 500
      CALL SYNCT(TRNS3)
400   CONTINUE
      STOP
500   CALL EXCITE(BRAKER,CBY)
      END                                              Ⓨ

      SUBROUTINE DISPAC(CBZ)/* CBZ with Capacitor */
      :
      :
      END                                              Ⓩ
```

図5 PCL によるプログラム例
Fig. 5 An example of programming by PCL.

```
function LINE DISPATCHING (CB) options (open);
    if STATE (VOLT) AT (CB, RIGHT) is ABNORM then
      if STATE (VOLT) AT (CB, LEFT) is ABNORM then
        stop;
      else
        SWITCH-ON AT (CB);
      end;
    else
      SWITCH-ON AT (CB);
    end;
end LINE;

function SWITCH-ON AT (CB) options (open);
    if SYNCHRONIZATION AT (CB) is ABLE then
      EXCITE (BRAKER) AT (CB);
      :
    else
      stop;
    end;
end SWITCH-ON;

function SYNCHRONIZATION AT (CB): logical options (open);
    % if CB is CBX then % APPARA='GENE5';
           is CBY then % APPARA='TRNS3';                ©
    end;
    for I=1, MAX repeat
      if STATE (SYNC) AT (CB) is DOUKI then
        return (.true.);
      else
        SYNCHRONIZE BY (APPARA);
      end;
    end;
    return (.false.);
end SYNCHRONIZATION;
```

各変数・定義名は別途定義済みとする.
仮引数の宣言は省略した.

図6 SPL によるプログラム例
Fig. 6 An example of programming by SPL.

から呼び出される SYNCHRONIZATION AT ルーチンのコンパイル時処理により，機器の性格を 'GENE 5', 'TRNS 3' 等の機器名より割り出し，それに対応した処理を選ばせることが出来るので余分なパラメータを必要とせず，類似な機能の処理群を SYNCHRONIZATION AT ルーチンに統合して管理できる．また同図の©部に示すように%選択文（この% if…is… 文は多重分岐用である.）を利用して，各遮断器別の同期調整用機器名の検索を集中的に処理させることができるが，この処理はコンパイル時に完了し，選択された部分の結果のみが，オブジェクトとして現れる．

従来方式では，電圧関係と同期関係のステータス・チェックはパラメータの数が異なるため別のサブルーチン名を用いる必要があるが，SPL の例では整数形または文字列のリストが許されるので同一マクロ命令で呼び出すことができる．このように SPL のコンパイル時機能を活用することにより，汎用的な標準パッケージを容易に作成することができる．

また，この SPL の例では，options (open) の指定により手続き SYNCHRONIZATION AT (CB) をインライン展開しているが，これにより，ラインディスパッチング関数の手続きの流れが，処理速度を低下することなく，非常に分りやすくなっている．

また，SPL の例では，下記のコメント形式の手続き参照機能を使用している．

STATE(A) AT(B)
　　B地点のAの状態
SYNCHRONIZATION AT(B)
　　B地点の同期化状態
SWITCH-ON AT(B)
　　B地点で同期投入処理を行う
SYNCHRONIZE BY(C)
　　Cを使用し同期化処理を行う
EXCITE(D) AT(B)
　　B地点でDを付勢し投入する

このように，プログラムに「てにをは」をつけることにより，プログラムの意味が明瞭となり，プログラムのドキュメント性が大幅に向上する．

5. むすび

以上，問題向応用，すなわち，標準化応用を中心に，制御用ストラクチャードプログラミング言語 SPL の機能，適用例について述べた．

制御用計算機のソフトウェアを問題向に標準化していくためには，

（1） プログラミングの知識のないプロセスエンジニアでもソフトウェアを容易に理解できること

（2） 顧客の多種多様な仕様を吸収できるプログラミングが可能であり，しかもプログラムの保守が簡単であること

（3） オブジェクト効率の良いプログラムを作成出来ること

が重要である．

このため，SPL は

（1） コメント形式，キーワード形式の手続きの参照機能

（2） コンパイル時機能

（3） 手続きのインライン展開機能

を備えているが，これら機能が問題向応用すなわち，標準化応用に，充分効果のあることを電力系統制御を例にとって示した．

なお，SPL の使用経験によると，(1)制御用分野では，データの入力から出力までの処理ステップ数が比較的小さく，しかも物理的意味の明確な分岐が多いので，制御文等でオプションをさらに充実させる要望がある反面，(2)機能が多様なため，初心者がなじみやすくできるように，ベーシック機能をオプション機能から完全に分離した運用ガイドの整備（実施済み）の要望などもあった．また，第1章でふれた会話形化，ハードとソフトを一体としたモジュル化（分散処理化）などの要請も根強く存在している．

今後，信頼性の高いプログラムの作成，プログラムの標準化の推進，問題向言語の開発等に広く SPL を活用していく予定である．

終りに，本研究の遂行にあたり，御指導御鞭撻いただいた日立製作所日立研究所奥田博士，川本博士，同じく大みか工場伊沢工場長，宅間副工場長，北之園部長，高井副技師長，システム開発研究所中田博士に深謝します．また，SPL の設計者である，日立製作所システム開発研究所の野木氏には言語機能に関して有益な御討論をいただいた．

参考文献

1) Dahl, O. et al.: Structured Programming, Academic Press (1973).
2) Stevens, W. P. et al.: Structured Design, IBM Systems Journal, No. 2 (1974).
3) IBM: Improved Programming Technologies, IBM Management Overview (1971).

4) 林ほか：制御用ストラクチャードプログラミング言語 SPL の開発思想，昭和 51 年度情報処理学会第 17 回全国大会 1.
5) 野木ほか：制御用ストラクチャードプログラミング言語 SPL の言語仕様の特徴，昭和 51 年度情報処理学会第 17 回全国大会 2.
6) 中所ほか：制御用ストラクチャードプログラミング言語 SPL の処理系の特徴，昭和 51 年度情報処理学会第 17 回全国大会 3.
7) 浜田ほか：制御用ストラクチャードプログラミング言語 SPL の問題向言語への応用，昭和 51 年度情報処理学会第 17 回全国大会 4.
8) 林，野木，中田，浜田：制御用トップダウン・ストラクチャードプログラミング言語—SPL—，日立評論，Vol. 60, No. 3 (1978).
9) 野木，中所，中田，浜田，林：トップダウン・プログラミング言語 SPL におけるモジュール概念について，情報処理学会ソフトウェア工学 3-1 (1977).
10) Wirth, N.: The Programming Language Pascal, Acta Informatica, 1 (1971).
11) 竹下：PL/I (複合プログラミング言語)，日本経営出版会 (1970).
12) Caine, S. H. et al.: PDL-A tool for software design, IEEE Tutorial on Software Design Techniques, 2nd edition (1977).
13) Parnas, D.: Designing Software for Ease of Extension and Contraction, Proceedings of 3rd International Conference on Software Engineering (1978).
14) 桑原ほか：プロセス制御用言語 PCL の開発，日立評論，Vol. 54, No. 9 (1972)

(昭和 53 年 8 月 25 日受付).
(昭和 54 年 7 月 19 日採録)