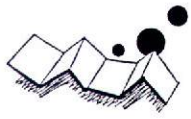
解　説



Pascal 総　説†

疋田輝雄††　武市正人†††

この稿では，短い紙幅ながら，プログラミング言語 Pascal の現時点における全体像を与えることを目標とする．Pascal の歴史，言語仕様，処理系，適用分野，標準規格案，仕様の拡張などについて，それぞれ一通りのことを紹介する．より詳しくは，後続の3篇の解説や稿末にあげた参考文献を参照されたい．本稿と同趣旨の概観としては，Schneider によるものがある[14]．

1. 歴　史

Wirth（ヴィルト）は言語の設計においてつねに成功してきている．かれは1960年代に，Algol 60 の後継として Euler と Algol W を，またシステム記述用言語の元祖となった PL 360 を設計した．そして1968年頃からスイス，チューリヒの連邦工科大学(ETH)において，主として研究・教育用に設計した汎用高級言語が Pascal である*．

Pascal の処理系は最初 CDC 6400 の上に作成された．その開発の方法は，コンパイラをまず Pascal 自身で記述し，一通り完成してから，それを（人間が）機械語に書き直すというものであった．その結果，1971年に言語仕様と処理系の設計に関する論文[19),20)]が公表された．

この後，主としてヨーロッパの大学において，Pascal の処理系を他機種の計算機へ移すことが行われた．そして数年間の使用経験を経て，言語仕様にいくつかの変更が加えられて，1974年に改訂版が公表された[8)]．これは標準 Pascal (Standard Pascal) と称され，現在に至るまで言語の一応の規準として受入れられている．

この頃から Pascal の処理系は各機種の計算機に急速に普及しはじめた．ETH 自身も普及に熱心で，たとえば後述のPコンパイラは，インタプリティブに動くため処理速度に難はあるが，移植の第一段階として実に有効な役割を果した．わが国でも 1975 年頃からぼつぼつ処理系が動き始めた．この頃は処理系は大学や研究所など研究機関で作成され，使用されていたが，最近，2，3年に至って計算機メーカ自身で処理系を供給するところも現れてきた．また処理系の普及，改良につれて，言語の使用分野も当初の研究教育用から，メーカやソフトウェア・ハウスにおけるコマーシャルな目的のものへ広がってきた．

このような普及，とくにコマーシャルな分野への進出に伴って，言語仕様における欠点や不足点が多くの人々によって議論されるようになり，改良および拡張案が数多く提案されてきた．しかしこれら言語仕様の変更，拡張は処理系ごとに勝手になされているのが現状である．また，Wirth による言語仕様の記述にあいまいな点が諸所にあることも事実である．

このような状況のもとで，主としてコマーシャルな要求から，公的な標準規格を求める声が強くなってきた．そして現在，イギリスの Addyman を中心とするグループが精力的に国際標準規格 (ISO) 案作成の作業をすすめている．

2. 言語仕様の特徴

Pascal は Algol 60 の流れをくむ汎用プログラミング言語であり，ブロック構造など大きな骨組みは Algol 60 から受継いでいるが，データ構造など新たに組込まれた機能も多い．言語全体についていえる一般的な特性は，Wirth 自身が Pascal 設計の目的として述べていることとも一致するが[8)]，

i）言語仕様が，高級であると同時に，簡潔かつ明晰であること

ii）処理系の信頼性と効率に対する考慮

の2点であろう．

---

† An Overview of Pascal by Teruo HIKITA (Department of Mathematics, Tokyo Metropolitan University) and Masato TAKEICHI (Department of Computer Science, University of Electro-Communications).

†† 東京都立大学数学科

††† 電気通信大学計算機科学科

* この言語名は何かの略称ではなく，数学者かつ哲学者 Blaise Pascal (1623—1662) の名にちなんだものである．

```
program tablesort (input, output);
  const n=100;
  type string=packed array [1..16] of char;
       item=record key: integer; info: string end;
  var table: array [0..n] of item; i, j: integer;
       c: char;
procedure sort;
    var i, j: integer; w: item;
  begin for i:=2 to n do
    begin w:=table [i]; table [0]:=w; j:= i-1;
      while w. key< table[j]. key do
        begin table [j+1]:=table [j]; j:=j-1
        end;
      table [j+1]:=w
    end
  end;
begin {main program}
  for i:=1 to n do {move data to table}
  begin read (table [i]. key); for j:=1 to 16 do
      begin read (c); table [i]. info [j]:=c end
  end;
  sort;
  for i:=1 to n do {print table}
  begin write (table [i]. key, ' ');
      for j :=1 to 16 do
      begin c:= table [i]. info[j]; write (c) end;
      writeln
  end
end.
```

プログラム-1 Pascal プログラムの一例

簡潔さについては原典[8]の仕様の定義の部分がわずか37ページで済んでいることからも明らかであろう．このことは言語の学習の容易さや処理系の大きさに大いに関係する．言語機能が他言語に比して見劣りしないのにこの簡潔さを保てるのは，仕様の整合性のおかげである．つまり諸機能の定義が一貫しかつ互いに独立的になっており，またハードウェア依存的な規定や例外規定を少なくしようとしているからである．もっとも，マニュアルが薄いのは，他言語と異なり，ファイルや入出力処理が思いきって簡単であるというおかげも多分にある．

処理系の信頼性や効率に対する配慮というのは，たとえば Algol 60 における動的配列や own 変数のように，プログラムの翻訳時や実行時において負担となるような言語機能をできるだけ除くということである．

次に言語仕様の具体的な特徴としては，

- 豊富なかつ洗練された，データ型構成の機能
- 洗練された制御構造

といったところであろう．以下，具体的に言語仕様を紹介していく．なお**プログラム-1**は Pascal プログラムの一例である．このプログラムは表の中に読み込んだデータを key の値によってソートし，結果を出力するものである．

i) 複合文　**begin** S_1;…; S_n **end**
ii) 条件文　**if** B **then** S_1 [**else** S_2]
　　case E **of** C_1: S_1;…; C_n: S_n **end**
iii) 繰返し文　**while** B **do** S
　　repeat S_1;…; S_n **until** B
　　for I:=E_1 {**to** / **downto**} E_2 **do** S
iv) goto 文　**goto** L

S: 文, B: 論理式, E: 式, C: ケースラベル, I: 制御変数, L: ラベル

図-1 Pascal の制御構造

i) 単純型
　・スカラ型　論理型
　　整数型
　　実数型
　　文字型
　　(ユーザの定義する) スカラ型
　・部分範囲型
ii) 構造型
　・配列型
　・レコード型
　・集合型
　・ファイル型
iii) ポインタ型

図-2 Pascal のデータ構造

2.1 制御構造

Pascal の制御構造の種類を**図-1** にあげる．それぞれの意味は自明であろう．現時点で見ると目新しいものはないが，「構造的プログラミング」に適ったものといえる．

2.2 データ型

Pascal のデータ型は，C. A. R. Hoare の考えによるものを取入れたもので，**図-2** のように分類できる．ユーザの定義するスカラ型とは，たとえば，曜日名とか方角名のように，有限個の名前を列挙したもので，スカラ型の変数のとる値はそれらの名前自体である．

例　**var** *status*: (*active*, *waiting*, *completed*);

スカラ型のうち残り4つは標準型と呼ばれ，あらかじめ用意された型である．部分範囲型とは，(実数型以外の) スカラ型の，ある部分範囲として定められる型で，配列の添字の型としてよく用いられる．

例　**var** *i*: 1..100;

構造型とは，上の単純型と後述のポインタ型とを基本的な要素として，段階的に新たな型を構成していく手段である．配列やレコードはおなじみのものである

が，レコードの成分として通常のフィールドのほかにタグフィールドというものを使えるのが面白い．このタグフィールドの（実行時の）値によって，そこより後のフィールドの種類や個数が変化してもよい．これを可変部つきレコード(variant record)と称する．

例
```
var person :
    record
     name: packed array [1..16] of char;
     case married: Boolean of
      true: (nochildren: integer);
      false: ( )
    end;
```

集合型の値は（実数型以外の）単純型の要素からなる「集まり」である．要素の型は1つの単純型に限られ，任意の型の要素からなる集合を考えられるわけではない．また要素の個数に上限のあるのがふつうで，その値は処理系により異なるが，ふつう数十から数百程度である．

例
```
var dset: set of 0..9;
    chset: set of char;
```

Pascal で単にファイルというものは，同じ型の要素からなる「列」のことであり，順編成ファイルにあたる．直接アクセスなどほかの形式のファイルは扱えない．なお，標準入力，出力も，文字を要素とする特別のファイルと見なすことになっている．

構造型（配列とレコード）に対してはさらに，詰込み型（packed type）の指定をすることができる．この指定をすると，実行時の処理速度を少し犠牲にして，メモリの機械語内にできるだけデータを「詰め込む」ことができる（実際にどの程度まで詰めるかは処理系ごとに異なる）．なお，文字列は，Pascal においては，文字を要素とする詰込み配列という型をもつことになっている．

最後のポインタ型は，いわゆる動的なデータ構造を作る際に用いられるもので，PL/I におけるものと同様である．ポインタとはほかのあるデータを「指す」もので，つまりポインタは値としてそのデータの場所（アドレス）をもつ．これらの指されるデータは，通常に宣言される変数とは異なり，標準手続き new によってプログラム実行中にその場所を（ヒープと呼ばれる領域内に）割付けられる．Pascal においては1つのポインタ変数の指すデータの型は一定でなければならず，ポインタの宣言の際に明示する．ポインタを用いて，プログラムの実行中に全体の形状の変わりうるデータ構造，たとえば線形リストや木構造，また一般にグラフ構造を構成できる．

2.3 型 定 義

以上の単純型，構造型およびポインタ型を用いて，いくらでも複雑な型を構成することができる．その際構成した型には名前をつけることができ，つけられた名前（型名）をさらにほかの型の構成の際に用いることができる．型に名前を与えることを型定義という．

例
```
type string=packed array [1..16] of char;
     item=record key: integer; info: string
            end;
  var w: item;
```

次の例では，線形リストの各頂点の型と，それらの頂点をつなぐポインタの型を定義している．

例
```
type nodep=↑node;
     node=record info: string;
                  next: nodep
          end;
```

2.4 Pascal 批判

Pascal の言語仕様に対する批判は，Habermann によるもの[4]を初めとして数多くあり，一方で，Wirth 自身の Pascal に対する評価もある[21]．Pascal に対する批判は一般に次の2種類に大別できよう．

A）言語仕様の規定におけるあいまいな点あるいは不備な点

B）不足していると思われる言語機能

このB）に含まれる機能はこれまでずい分多くあげられており，Pascal の実際的な応用を考える上でも重要であるが，本稿では第6章で扱うことにする．

A）に属する問題は，一般にプログラミング言語の根本にふれるものが多く[17),18)]，今後の言語の方向を考える上でも参考になる．たとえば文献18)では次のようなものをあげている．

1) 型の同値問題
2) 名前の有効範囲と1パス・コンパイル
3) 要素による集合の構成的表現
4) 可変部つきレコード
5) 手続き（関数）引数の引数
6) 部分範囲からの値の逸脱
7) 初期化されていない変数の参照
8) まだ（ほかのポインタから）指されている動的変数の場所の解放（dangling reference）

ここで1)—3）が意味上のあいまいさ（ambiguity）を，4)—8)はプログラム実行中に予期しないエラーの

起こる危険性 (insecurity) を持っているとしている．たとえば6）は，配列の添字の値が実行中にあやまって範囲を逸脱する危険性などをいっている．つまり後のグループ（危険性）は，翻訳時または実行時のチェック，もしくは言語仕様による危険の防止の必要性のあるものであり，Pascal 以外の言語にも共通するものが多い．これに対して前のグループ(あいまいさ)は言語仕様記述における意味のあいまいさで，いわばPascal 固有の問題であるが，その解決はやはりプログラミング言語一般に関連するところが多い．

ここでは，上のそれぞれについて詳しく説明する余裕がないので，1）の型の同値問題についてのみふれておく．これは，

```
type T1 =array [1..10] of real;
     T2 =array [1..10] of real;
var a1 : T1 ; a2 : T2 ;
```

において，型 $T1$ と $T2$ とは「同じ」型であるのか，言い換えれば，代入文 $a1 := a2$ は許されるのか，という問題である．同じ型であると見なすのは「構造による同値」，違うと見なすのは「名前による同値」と称する．どちらを選ぶかは議論の多いところであるが，どちらかに決めておかないと，言語仕様としてはあいまいさを持つことになる．現状としては，前者の構造による同値を採用している Pascal 処理系が多いようである．

3. 処 理 系

Pascal の処理系は，少し以前までは，最初の ETH の CDC 6000 版[20]（Pascal 6000 と称される）を何らかの形で共通の源とするものが多かった．そのため処理系の内部構成や方式は互いに非常に似通っていた*．また同様の理由で，処理系は Pascal 自身で記述されている場合が多く，つまりいわゆるブートストラッピングの手法によって作成されたのである[11),15)]．このことは言語としての Pascal の，言語処理系記述における優秀性を示している．もっとも，最近では，処理系も多様化し，記述言語のみならず処理系の内部構成や方式もいろいろ異なったものが現れている．しかしここでは，主として CDC 6000 版にそった処理系について，その一般的な特徴を述べる．

前節で述べたように Pascal の言語仕様は簡潔かつ明晰で，しかも実行時における効率が問題となるような機能は意識して取除かれているので，処理系もまたすっきりとした構成をもち，作成上で問題となるような点はほとんどない．大きさも Pascal で記述して 5,000～6,000 行，それ自身で翻訳したオブジェクト・コードで 100 KB 程度である．比較的大きなユーザ・プログラムをコンパイルする際に，入出力などのルーチンや作業領域を考えても，全部で 300 KB 以下ですむ．

別の特徴としては，1パス (one-pass) 方式の処理系が多いということがある．すなわちユーザ・プログラムを先頭から順に1度ながめていくだけでオブジェクト・コードを生成してしまい，中間的なコードを作らないということである．言語仕様中でも1パス方式をやりやすいようにいくつかの点で配慮がなされている．この1パス方式はコンパイルの速度の点で優れているが，オブジェクト・コードの大局的な最適化 (optimization) はやりにくい．なお最近では，この最適化のためや，あるいはミニコンピュータ用として処理系の大きさに対する制限のために，多パス方式のものも増えている．

次にふれておきたいのは移植可能な (portable) 処理系である．そもそも Wirth たちの ETH のグループは Pascal 処理系の移植に熱心であったが，グループの一員である Ammann は CDC 6000 用処理系を作成した際に副産物として，そのコード生成部だけを変更して，Pascal-P コンパイラを作成した[5),13)]．これは仮想的なスタック機械のコード（P-コードと称される）を生成する処理系である．ある計算機上へ Pascal 処理系を移植したいときは，P-コードのインタプリタだけを作成すればよく，1人月程度の労力ですむ．70年代半ば頃，Pascal の普及においてこの Pascal-P の果した役割は非常に大きい．現在でも Pascal-P は ETH から入手可能である．しかしインタプリタ方式で速度に難があり，Pascal がこのように普及してきた現在ではその使命は終わったとみてよいと思われる．またこの P-コードによる処理系は，UCSD 版などマイクロコンピュータ用の処理系のもととなった．

これとは別の移植用の処理系として，H. H. Nägeli によるトランク・コンパイラ (trunk compiler) がある．これは Pascal で記述されたコンパイラ・プログラムであり，計算機のハードウェアに依存する部分を注釈の形で記述してあるものである．この計算機に依存する部分を実際のハードウェア向きに記述し書き込んで処理系を作成することが，とくにわが国において

* たとえば，構文解析の方式は，トップダウン方式でかつ1シンボルの先読み．

何例か行われた[5),6),16)]．この方法は，Pascal-P より多くの作業が必要ではあるが，出来上った処理系の実用性は格段に高い．

4. 適用分野

Pascal の適用分野は急速に広がりつつあるが，同時にその限界も認識されはじめている．ここでは次の4分野に分けて状況を簡単にまとめておく．

- 研 究 用
- 教 育 用
- システム記述用
- 応用プログラム開発用

研究用言語としては，Pascal はごく初期から好評であった．その用途は，アルゴリズムの記述，プログラミング言語上の新概念開発や新言語設計のためのベース，プログラムの正当性の証明における対象言語[7),12)]などである．Pascal が理論的な整合性と同時に実用性をも持っている点が好評の理由であろう．

プログラミングの教育用としては Pascal は現在の言語の中で最適であるように思える．そもそも Wirth 自身が教育用ということを Pascal 設計の目的の1つにあげていた．2，3年前から Pascal によるプログラミングの入門的教科書が国の内外で続々と出版されている．また，Pascal によるコンパイラ・ソースプログラムを読むことは，言語処理系の教育の一手段としても有効であろう[22)]．処理系のコンパクトさや処理の速さともあいまって，教育用言語としての Pascal の地位はますます高くなっていくように思われる．

システム記述用言語としての Pascal にはいろいろ問題がある[10)]．Pascal の処理系が最初の CDC 6000 版以来 Pascal 自身で記述されることが多く，またその移植の実績もあることから，Pascal にシステム記述用としての期待が持たれたということがある．事実，言語処理系のほかにも，たとえばクロスリファレンサなどの比較的小さなソフトウェア・ツールの記述に Pascal は実に有効である．しかし大きなソフトウェア，たとえばオペレーティングシステムのようなプログラムの作成には向いていないと考えられる．その理由は言語の機能が不足していることによる．不足している機能の主なものは次の3つであろう：

- 手続きごとの分離コンパイル
- 機械語による記述や割込み処理など，直接的にハードウェアを扱う機能
- 並列処理の機能

応用プログラム開発用としても，上の場合と同様に，言語機能の不足という問題がある（もちろん適用分野によるが）．それにもかかわらず Pascal がソフトウェア開発に使われつつあるのは，（処理系ごとに）不足している機能を言語仕様の拡張もしくは外部ルーチンの呼出しの形で解決しているからである．言い換えればそれだけ Pascal に魅力があるということであろうが，無制限に Pascal の標準から逸脱していくことは，単純性という Pascal の設計思想に反し，またプログラムの互換性を損なう点で問題である．

最後にふれておきたいのは，急速に普及しつつある，マイクロコンピュータ用言語としての Pascal である．処理系として有名なのは UCSD Pascal であり[3)]，またP-コード・インタプリタのハードウェア化も現れている．しかしこの分野でもやはり言語仕様の問題がある．

5. 規 格 案

Pascal のコマーシャルな分野への進出の必然の成行きとして，国際標準規格 (ISO) を作ろうという動きが現在進行中である．

1977年以来，British Standards Institution (BSI) の中に Addyman（マンチェスタ大学）を中心とする作業グループがあり，規格案を作る作業を精力的に進めている．規格案はこれまで，作業案第3版が IEEE Computer 誌の Vol. 12, No. 4 (1979年4月号) に，作業案第5版が ISO 案第1版[1)] として SIGPLAN Notices 誌に公表されている．そして現在のところ最新の版は，1980年8月の作業案第6版である．

この進行中の規格案の思想は，言語仕様の拡張はできるだけしないで原典[8)]の範囲を出ないようにし，むしろその中で記述のあいまいなところをはっきりさせようということであるように思われる．したがって作業中の規格案を見るかぎり本質的に標準 Pascal から離れるところはほとんどない．しかし大きな変更ないし拡張というべき点も1，2あるので，それらを次に述べる．

その第一は何といっても「適合配列」(conformant array) である．現在の Pascal では Algol 60 の動的配列に相当するものはなく，配列の大きさ（添字の上限値，下限値）は宣言時に定まっていなければならない．しかし，たとえば数値計算のパッケージ・プログラムにおいて，配列を引数として渡すときその大きさはいろいろ変りうる方が望ましい．そこで，手続きや

関数の引数として配列が宣言されているときは，その上限と下限も引数として宣言することが許され，このことにより，実際に手続きや関数に配列を実引数として渡すときにいろいろの大きさのものを渡すことができる．つまり動的配列を引数の場合にのみ許そうというものである．

このほかに注目すべき点としては，第2．4節で述べた型の同値がある．そこでは現在の処理系には構造による型の同値を用いるものが多いと述べたが，規格案では名前による同値に基づくものに，はっきり規定するようである．

6. 言語の拡張

Pascal はその簡潔さによって好評を得たが，第4章で述べたように実際的なプログラミングに際してはやはりいろいろの言語機能の不足が感じられる．現在作業中の標準規格案は仕様の拡張はほとんど認めていないが，現実の処理系はすでにいろいろの拡張を行っていることが多い．ここでは，Pascal の言語機能の不足点を見ていくという意味も兼ねて，多くの処理系によく見られる拡張やよく取り上げられる提案についてまとめてみる．なお Pascal は歴史的に，いくつかの新言語設計のためのベースとしての役割をも果したが，そのような大きな「拡張」については別稿「Pascal とそれ以降の言語設計」や，本特集後半の Ada の部を参照されたい．

これまでによく見られた拡張を次に列挙する．

- 構造型の定数の構成法（配列やレコードに対して）
- 変数の初期値設定
- own 変数
- 巾乗演算子
- case 文における otherwise
- 繰返し文の途中からの exit (escape)
- 種々の文字列処理機能
- 順編成ファイル以外の形式のファイル
- 端末によるインタラクティブな入出力
- 入出力における format 記述
- 分離コンパイル機能
- Fortran などで記述された外部ルーチンの呼出し
- コンパイル時におけるマクロ機能 (include 文など)
- オーバレイ機能

以上のうちで最後の4つは，言語の機能というよりもむしろ，言語処理系や支援システムの機能として扱うべきものであろう．言語機能の追加の方法としては，言語仕様そのものの変更ないし追加と，外部ルーチンの呼出しの形のものとがありうるが，一般的には後者の方が，プログラムの互換性の意味では望ましいと思われる．

上にあげた機能のなかでまず目につくのは，Pascal のファイルや入出力の定義に関する不満が多いことである．また，これよりもっと重要な問題は分離コンパイル (separate compilation) であろう．

現在の Pascal では手続きや関数はすべて主プログラムの中で定義し，全体をひとまとまりのプログラムとして一度にコンパイルしなければならない（一括コンパイル）．いわば外部ルーチンは標準手続き・関数のみなのである．この処理方式は，大きなシステムの作成の際には，コンパイル時間やプログラム修正の手間の点で問題になってくる．手続きごとにコンパイルしてあとで結合することにすればよいわけであるが，ある手続きのコンパイルの際に大域的な名前（環境）をどのように持込むかということが新たな問題になる．これに対していろいろの解決策が提案され，試みられている[9]．

7. む す び

文献8）において Wirth は，Pascal 設計の目的として次の2点をあげている：

i）いくつかの基本的な概念にもとづいた系統的な作業としてプログラミングを教育するために適した言語を使用可能にすること，そして，それらの概念はその言語において明確にかつ自然に反映されていること．

ii）現存の計算機の上に，信頼性があり，しかも実行効率の良い処理系を作成すること．

Pascal が世に出て10年を経た．それはまずアカデミックな世界で成功をおさめ，さらに最近はコマーシャルな世界にも急速に進出しつつある．Pascal は上記の2つの目的をすでに果し，さらに設計者も予期しなかった広い世界へ広がっていくように見える．

Pascal の功績と限界もはっきりしてきたようである．第4章で述べたことでもあるが，その功績は次のようにまとめられようか：

1) プログラミング言語の諸概念の明晰化．このことによる，プログラミングとプログラム保守の容易さ

2) 言語上の諸概念の研究発展のためのベース，あるいは新言語設計のためのベース

3) プログラミング教育用としての言語

一方，Pascal の限界は次のようなものであろう：

4) ファイル機能の弱さ，一括コンパイルなどによる，コマーシャルな適用への限界

5) さらに，ハードウェアを直接的に扱う機能や，並列処理の機能のないことによる，システム記述用言語としての限界

個々の処理系によっては，上の4)，5) でふれた機能は補われていることが多いが，標準の言語仕様にそれらがない以上，限界と称すべきであろう．そしてこの限界は，そもそもの Pascal の設計思想から考えれば当然のこととともいえよう．

こうした限界があるとはいえ，Pascal は，適切な分野で，ますますその重要性を増す趨勢である．わが国にも遅ればせながら定着し始めた．そして何よりもわれわれは，Pascal の言語仕様上の明晰な諸概念の認識と普及とを願いたいものである．

謝辞 査読者のていねいな評は原稿の改良のために非常に有益であった．心から感謝する．

参 考 文 献

1) Addyman, A. M.: A draft proposal for Pascal, SIGPLAN Notices, Vol. 15, No. 4, pp. 1-66 (1980).
2) Ammann, U.: On code generation in a Pascal compiler, Software-Practice and Experience, Vol. 7, pp. 391-423 (1977).
3) Bowles, K. L.: Microcomputer Problem Solving Using PASCAL, Springer (1977).
4) Habermann, A. N.: Critical comments on the programming language Pascal, Acta Inform., Vol. 3, pp. 47-57 (1973).
5) 疋田輝雄：コンパイラのキットを用いた PASCAL の移植，日経エレクトロニクス，No. 149, pp. 101-131 (1976).
6) Hikita, T. and Ishihata, K.: An extended PASCAL and its implementation using a trunk, Rep. Comput. Centre Univ. Tokyo, Vol. 5, pp. 23-51 (1976).
7) Hoare, C. A. R. and Wirth, N.: An axiomatic definition of the programming language PASCAL, Acta Inform., Vol. 2, pp. 335-355 (1973).
8) Jensen, K. and Wirth, N.: PASCAL: User Manual and Report, Springer (1974); 2nd ed. (1975), 和田英一訳：プログラム言語 PASCAL の文法，bit, 1976年4月号 (後半の Report の部分の訳).
9) 鍵政豊彦，荒木俊郎，都倉信樹：PASCAL 内部手続きの分離翻訳，信学論 (D), Vol. J63-D, pp. 177-182 (1980).
10) 木下 恂：PASCAL はシステム記述言語に適しているか？，情報処理，Vol. 21, No. 2, pp. 180-182 (1980).
11) Lecarme, O. and Peyrolle-Thomas, M.-C.: Self-compiling compilers: an appraisal of their implementation and portability, Software-Practice and Experience, Vol. 8, pp. 149-170 (1978).
12) Luckham, D. C. and Suzuki, N.: Verification of array, record, and pointer operations in Pascal, ACM Trans. Prog. Lang. Syst., Vol. 1, pp. 226-244 (1979).
13) Nori, K. V., Ammann, U., Jensen, K., Nägeli, H. H. and Jacobi, Ch.: The Pascal 'P' compiler: implementation notes (revised ed.), Berichte Nr. 10, Institut für Informatik, Eidgenossische Technische Hochschule, Zürich (1976).
14) Schneider, G. M.: Pascal: an overview, Computer, Vol. 12, No. 4, pp. 61-66 (1979).
15) Shimasaki, M., Fukaya, S., Ikeda, K. and Kiyono, T.: An analysis of Pascal programs in compiler writing, Software-Practice and Experience, Vol. 10, pp. 149-157 (1980).
16) Takeichi, M.: Pascal-implementation and experience, J. Fac. Engrg. Univ. Tokyo Ser. B, Vol. 34, pp. 129-136 (1977).
17) 和田英一：プログラム言語 Pascal ①～⑩, bit, 1978年1～10月号.
18) Welsh, J., Sneeringer, W. J. and Hoare, C. A. R.: Ambiguities and insecurities in Pascal, Software-Practice and Experience, Vol. 7, pp. 685-696 (1977).
19) Wirth, N.: The programming language Pascal, Acta Inform., Vol. 1, pp. 35-63 (1971).
20) Wirth, N.: The design of a PASCAL compiler, Software-Practice and Experience, Vol. 1, pp. 309-333 (1971).
21) Wirth, N.: An assessment of the programming language PASCAL, IEEE Trans. Software Engrg., Vol. SE-1, pp. 192-198 (1975).
22) Wirth, N.: Algorithms+Data Structures=Programs, Prentice-Hall (1976), 片山卓也訳：アルゴリズム＋データ構造＝プログラム，科学技術出版社 (1979).

付 録 PUG

Pascal に関する情報交換のための（私的な）組織として Pascal Users Group（略称 PUG）があり，現在世界中から3,000名以上が加入している．その主たる

活動は Pascal News という100ページほどのタイプ印刷の雑誌を年4回程度の割合で発行することである．この Pascal News は1974年から出ている Pascal Newsletter の後身で，（現在のところ）最新号はNo. 18（1980年5月）である．R. Shaw (DEC, アトランタ）が編集していて，記事の内容は，

・Here and There with Pascal

Pascal に関するニュース，本，書評，論文，会議などの紹介．

・Applications

Pascal による各種応用プログラムのソース・リスト．

・Articles

Pascal に関する投稿小論文．

・Open Forum for Members

メンバによる討論の場．

・Implementation Notes

各機種の計算機上の Pascal 処理系の紹介．

といったところである．しかしときには，標準規格案だけで1号分になってしまうこともある．PUG に加入するには，日本からは，タスマニア大学（オーストラリア）の A. Sale 教授に申込むことになっている（年会費は現在8 A$）．

（昭和55年10月31日受付）