解　説

Pascal の適用分野†

安　村　通　晃††

1. まえがき

1968 年 N. Wirth によって作られたプログラミング言語 Pascal[1]は，70 年代の新しい言語として研究用・教育用に大学や研究所などで静かなブームを呼んでいた．最近では実用面でも注目され始め，また特にマイコン用の Pascal コンパイラ (UCSD の Pascal など) の普及はめざましい．Pascal 処理系の主たる供給源は，大学，研究所，ソフトウェア会社などであったが，最近では半導体メーカやコンピュータメーカも小型機やマイコン用の処理系を作り出している．ごく最近，日立，三菱や IBM などのメインフレーマも Pascal の本格的サポートに乗り出した．現在では，マイコンから超大型計算機まで世界中のほとんどの計算機で，Pascal が利用可能となっている[3]．

Pascal の普及と実用化の動きと併行して，Pascal 言語の国際規格 (ISO) を定める動きも急ピッチで進んでいる[5]．規格の早期制定は，Pascal で書かれたソフトウェアの互換性を維持する上で，実用上きわめて重要な事である．しかしながら，一部に Pascal の機能拡張を誇示するような動きもある[4]．

ここでは，Pascal の適用分野を実用的な観点から論ずるが，それに先立って，Pascal の設計思想について述べる．従来の言語に多くみられた機能中心主義とは異なる，簡潔性・信頼性・抽象性などの諸特徴を Pascal は備えているが，この Pascal の設計思想を抜きにしては，Pascal の適用分野を語ることはできない．Pascal の適用分野としては，研究用や教育用を除いた実用面では，やや複雑なアルゴリズムとデータ構造を必要とする分野がその対象となることを述べる．その代表的な例として，コンパイラをとり上げ，従来型の言語との比較を行う．最後に，規格の早期制定の必要性と，拡張に対する考え方を述べ，しめくくりとする．

2. Pascal への期待

Pascal が注目されているのは，近年のプログラミング方法論（たとえば，構造化プログラミングなど）や言語設計論[7),8)]の進展と無縁ではない．従来，プログラミング言語に対して，「何ができるか」という機能中心の評価が行われてきたが，今後は，「いかに使えるか」，という使いやすさに重点を置いた評価も必要となる．したがって，適用分野について述べる際にも，Pascal の設計思想[6)]について触れておかなければならない．この観点から Pascal の特徴を 5 つ挙げるとすれば，

1) 簡潔性，2) 信頼性，3) 保守性，4) 抽象性，5) 効率，

となる．これらの諸点について以下順に述べる．

1) 簡潔性 (Simplicity)

Pascal の第 1 の特徴は，この簡潔性にある．Pascal は，複雑すぎないこと，大きすぎないことを最大の目標に設計されている．同一の機能を実現するのに，より構成的，より基本的な言語要素 (Constructs) を用い，重複やムダがないように務めている．したがって，ほかの一部の言語にみられるようなマクロな機能はない．この簡潔性によって，ユーザは Pascal の一部分だけでなく全体を使いこなすことが可能となる．また，副次的効果として，処理系の規模が小さくてすみ，より小型の機械でも実現可能である．

2) 信頼性 (Reliability)[7)]

Pascal が従来の言語と大きく違う点の 1 つに，信頼性の重視という点がある．Pascal では，すべてのデータに型があり (strongly typed)，変数等の実体は使用前に宣言することにより，コンパイル時のチェックを厳しく行うことで，基本的に信頼性が実現されている．あいまいさを生ずる省略形や，データ型の自動変換を行わない点も，信頼性の向上につながっている．また，多くの Pascal 処理系では，配列の添字等の範囲チェ

† Application Areas of the Pascal by Michiaki YASUMURA (Central Research Laboratory of Hitachi Co., Ltd.).

†† (株)日立製作所中央研究所

ックや，ポインタの無効参照のチェックを実行時に行うオプションを標準的に有している．

3) 保守性 (Maintenability)

ここでいう保守性とは，主として読みやすさ(Readability) のことである．従来の言語が，書きやすさ(writability)に重点を置いていたのと違い，Pascal では，読みやすさの方にむしろ重点がある．保守性は，内容的には前項の信頼性と共通する面が多く，たとえば，変数等の宣言，省略形の禁止，記号等の一様性(uniformity)，などがその例である．読みやすさの強調ということで，言語の処理形態もオンライン向きというよりも，バッチ向きである．

4) 抽象性 (abstraction)

Pascal が簡潔でありながら，強力な能力 (power) を有する要因は，その抽象度の高さにある．その洗練された制御構造と豊富なデータ構造化機能により，従来の言語より比較的高いレベルでプログラミングが可能である．たとえば，従来の Switch や計算型 GOTO は case 文として，また bit ストリングは集合 (set) 型として Pascal では扱うことができる．

5) 効率 (efficiency)

前項の抽象性にもかかわらず，Pascal はコンパイル時実行時の両方の効率を重視した言語である．まず，コンパイル時に関しては，1-パスコンパイルでかつ後戻りなしの1シンボル先読みで解析ができるように文法が定められているため，Pascal処理系のコンパイル速度は従来言語の処理系よりも一般に速い．また，実行時に関しては，マクロな演算がなく，基本的にレジスタを有効利用できるような演算のみが処理対象であるから，メモリ，実行速度の両方共効率を落す要因はない．Pascal-P とよばれるインタプリタは別として，一般に最適化なしの Pascal 処理系は，最適化なしの FORTRAN 処理系と同程度のオブジェクトを出すことが可能である．最適化に関しても事情は同じである．

Pascal は再帰呼出し可能な言語であるが，再帰呼出しの可能な言語では手続き呼出しが遅い，という迷信(myth)が一部にある．この迷信は，①非再帰的なアルゴリズムを再帰的なアルゴリズムと比較しているか，②一部の効率の悪い再帰呼出し可能言語の処理系を例に考えているか，のいずれかに原因があるのではないかと考えられる．①については，たとえば，階乗計算で繰返しを使わずに再帰呼出しを使えば遅いのは当然で，正しく比較するとすれば，スタックを使って自分で再帰呼出しと同等のアルゴリズムをプログラム化したものと比較してみるべきであろう．また ②に関しては，一部の従来型言語で行っているような手続きの入口で毎回 OS のメモリ割当てマクロを発行する方式ではなく，Pascal の多くの処理系で行っている，各手続きのデータ領域の確保はベースレジスタ値の変更のみで行う処理方式に注目して欲しい．実際に測定してみても，Pascal の手続き呼出しと FORTRAN のサブルーチン呼出しとでは，ほとんど差がないことが判るであろう．

以上の特徴を統一的に実現しているのが Pascal 言語であり，このうちどの一つを欠いても Pascal らしさが失われる．Pascal の言語拡張はそれ自体としてはむしろ容易なことであるが，上記の諸特徴を殺さずに拡張を行うのはやや困難なことである．

また，上記の諸特徴も従来型の言語と比較の上での話であり，Pascal 以降の言語と比べて，Pascal に欠点がないことはない[9]．たとえば，Ada と比較してみればよい．しかし，現時点では，実験的な言語を除いて，処理系が広く普及している言語の中で，上記5つの特徴を最も強く実現している言語は，Pascal以外にないと思われる．

3. Pascal の適用分野

これまで Pascal は，主として研究用・教育用として，大学や研究所を中心に使われてきた．研究用としては，まず，Pascal が前章で述べた新しい言語の設計思想を含んでいることから，多くの実験的言語のベースとされてきた．たとえば，Euclid, Clu, Sue, Alphard, Concurrent Pascal, Modula, Ada など Pascal 以後の新しい言語に大きな影響を与えた．公理化の研究も進み，プログラム検証の対象言語にもされた．アルゴリズムの記述に適していることから，主要情報科学関係の論文では，Algol 60 に代って，Pascal をアルゴリズム記述用に用いる例が多い．教育用としても早くから着目され，たとえば従来アメリカの大学では教育用として多く使われてきた PL/C*に代って Pascal が用いられるようになってきた．これは，Pascal がプログラミング言語の基本的要素をすべて備え，かつ系統的なプログラミング教育に適しているためである．Pascal のプログラミングに関する教科書も我が国内外ですでに数多く出版されている．

実用目的では，一般の応用プログラムは一応守備範

* Cornell 大学で開発された．PL/I のサブセット言語．

囲であるが，その他，特にやや複雑なアルゴリズムやデータ構造を必要とするような準システムプログラムも適用対象である．たとえば，図形処理システム，言語処理システムなどは準システムプログラムの例である．

これに対し，実時間システムやOSの核の部分，あるいはデータベースシステムのファイルアクセスの部分など純システムプログラムは，現在の Pascal に並列処理，例外処理，直接アクセスファイルの機能などに欠けるため，必ずしも最適ではない．これらの純システムプログラムでも，Pascal で記述しきれない部分の割合が比較的少ない場合には，その部分のみアセンブラでコーディングすることも考えられる．また，Pascal に機能拡張を施す方法も直ちに考えられるが，この点については，5章でまとめて論ずる．

一般応用プログラムとしては，通常の数値計算や簡単なデータ処理等は大体 Pascal の適用分野と考えてよいが，これらの場合でも，最適化の強化，標準関数，ライブラリ等の追加により処理系を拡充する必要がある．これらの点に対しても，第5章で詳述する．

次に，Pascal の適用代表例として，コンパイラをとり上げ[12]，既存のシステム記述言語と比較してみる．比較対象言語としては PL/I のサブセットを基本としたシステム記述言語を考える．以下，コンパイラの主要処理に添って比較を行う．

（1） 入 出 力

コンパイルする際に必要な入出力は，ソーステキストの入出力，および，オブジェクトファイルの出力で，いずれも順編成ファイルでよい．Pascal では，文字型を単位とするテキストファイルと，バイナリを対象とするレコード型のファイル（いずれも順編成）が扱えるから，コンパイラで必要な入出力は，すべて Pascal で記述できる．これに対し，PL/I 系のシステム記述言語では，入出力の機能が備わっていないとすると，入出力に関してはアセンブラでコーディングすることになる．

（2） 文 字 処 理

入力したソーステキストをシンボルに分けるなど，コンパイルには文字処理が頻繁に行われる．Pascal では，文字ストリングは，文字型を要素とする固定長の配列で表わされ，その演算は，各要素である文字の各々に分解して行う必要がある．(ただし，比較や代入は文字ストリングの全体に対して行える．)文字ストリングの連結や部分照合などのマクロな演算はないし，可変長の文字ストリングも扱えない．しかし，これらの機能はコンパイラにとって，あれば便利ではあるが，必要不可欠という機能ではない．PL/I 系のシステム記述言語でも通常 Pascal 同様マクロな演算はないし，可変長のストリングも扱えない．

（3） リスト処理

コンパイラでは，動的なデータ構造を扱うリスト処理が特有の処理である．リスト処理は，ポインタと，構造体（Pascal ではレコード）を組合わせて行う点では，両方の言語とも同様である．しかし，PL/I 系のシステム記述言語のポインタには型がなく，したがってどんな型のものでも指せるのに対し，Pascal では決められた型しか指せない．また，PL/I 系のシステム記述言語は，任意の変数のアドレスをポインタ変数に代入したり，ポインタ変数に対して四則演算が可能であるため，操作ミスを起す可能性が高い．これに対し Pascal では，ポインタは，HEAP と呼ばれる特定領域にとられた実体しか指せない上に，比較と代入の演算しか許されないため，いわゆる dangling reference の問題*の回避ができ，信頼性がより高い．なお，Pascal で異なる型のものをポインタで指そうとする場合は，可変レコード型(variant record) との組合せによって行う．

（4） ビット処理

コンパイラでは，各種のフラグをビット列で表わしそれを操作することも多いが，PL/I 系のシステム記述言語では，ビット列型のデータに対する演算でこれらの処理を行う．これに対し，Pascal では，ビット列を集合 (set) という高い抽象レベルで扱える．すなわち，Pascal の集合では，代入のほか，合併，交わり，差集合の演算が行えるほか，同値性，包含関係，要素判定の検査が行える．各要素はビット番号ではなく，名前で扱える．本来の，Pascal の集合は，演算器上で効率よく扱える大きさ，すなわち高々 64 個程度の要素数(一語長または二語長) しか考えていなかったが，文字型の集合も扱う必要上，もう少し広い範囲の要素数が扱える Pascal 処理系も少なくない．

（5） テーブル構造

コンパイラの中で作られる各種のテーブルは，両方の言語とも基本的には，配列と構造体（レコード）との組合せで作られることには変わりがない．テーブル

---

* ブロック構造型の言語で，スタック上にあるローカルなデータをポインタが指していると，ブロックから抜けたときスタック上に指されているはずの実体が消えてなくなる問題をいう．

サイズがコンパイル中に可変である必要はまずなく，大体システムジェネレーション時に決まればよい．したがって，この意味では動的配列はコンパイラにとっては不要である．本質的に動的にサイズが変るようなものは，通常，リストの形にする．

PL/I 系のシステム記述言語でも Pascal でも階層的な構造体（レコード）が記述可能であるが，Pascal のレコードにある名前の有効範囲 (scope rule) が PL/I 系のシステム記述言語の構造体ではないことがあるため，各フィールド名にユニークな名前をつける必要が出てくる．また，Pascal がレコードや配列の型の前に packed を指定して詰めた型を機械独立に表わすのに対して，PL/I 系のシステム記述言語では，語境界 (boundary) や，揃え方 (alignment) を機械依存の形で指定することが多い．詰めた型は，コンパイラなどメモリ制約の強いプログラムに対して有効である．

（6） 制 御 構 造

分岐や繰返しに関する制御構造では，両方の言語ともほぼ同内容の機能の文を含むが，Pascal の方がよりまとまっている．PL/I 系のシステム記述言語では，名札 (label) がデータとして扱える点が，信頼性上やや問題である．手続きに関しては，PL/I 系のシステム記述言語で関数がなかったり，再帰的な手続き呼出しが行えないものがある．特に，リストや，トリーなど再帰的なデータ構造を扱うコンパイラにおいて，再帰呼出しが使えない場合は，プログラミングにやや困難を生ずる．

（7） アセンブラとの結合

特別な機械命令や OS のマクロ命令を使用する際には，PL/I 系のシステム記述言語では一般にインラインでアセンブラのコードが記述できるが，Pascal では外部手続きとしてのアセンブラと結合可能にしている処理系が多い．一般にコンパイル中は必ずしもアセンブラルーチンを必要としないが，コンパイラ自身（Pascal で書かれている）を実行する際に使う実行時ルーチンは通常アセンブラで書かれている．

（8） そ の 他

Pascal では，データ型が厳格であり，変数等の宣言により，インタフェースのチェックをはじめ，コンパイル時のチェックが厳しいが，PL/I 系のシステム記述言語では，型の概念があいまいで，省略が許され，インタフェースチェックも行われないなど，コンパイル時のチェックが甘い．したがって，後者で実行時になってはじめて気付くバグが，前者ではコンパイル時にチェックできることがあり，一般に Pascal でのコーディングやデバッグの期間が短くて済む．

以上，PL/I 系のシステム記述言語と Pascal を，コンパイラを適用例として比較してみたが，前者は PL/I のサブセット風の高級アセンブラ言語であり，自由度がある半面信頼性にやや欠けるきらいがある．コンパイラに関してのみいえば，Pascal がより適していると思われる．

4. 規格の必要性

半導体の進歩に支えられた，ハードウェアの性能向上と価格の低下にはめざましいものがあるが，これに比べ，ソフトウェアの生産性の向上は微々たるものである．生産性の飛躍的向上はのぞめず，なおかつ，ソフトウェアの需要そのものは年々増大しているから，このギャップを埋めるのに，既存ソフトウェアの有効活用が望ましい．その際に最も重要なことは，ソフトウェアの互換性と保守性の向上であろう．保守性の向上はさておき，ソフトウェアの互換性を保つためには，プログラミング言語の規格を定めること，ユーザ・メーカ共その規格を守っていくことが特に重要である．

Pascal の場合，従来の研究所や大学内での使用から実業界での使用へと広がってきて，特に規格の必要性が生じてきた．Pascal 処理系は Wirth のいるスイスの ETH* で作られたものがもととなって移植が行われている間は比較的標準化が自然に行われていたが，その後，各地で独自に Pascal 処理系を作りだされてきたので規格制定を急ぐ必要に迫られてきた．特に，Pascal の場合，処理系を作るのが比較的容易で，また，拡張すること自体も技術的に難しくないことから，規格なしに放置しておくと，さまざまな Pascal 族ができる危険性にある．従来の標準 Pascal の原典である Pascal User Manual and Report[1] の記述のあいまいであることも，規格制定を必要とするもう一つの要因となっている．

現在，Pascal の国際規格 (ISO) を定める作業が急ピッチで進んでいる．その詳細な内容については，本号の別論文を参照していただくとして，ここでは，以下の点についてだけ触れる．すなわち，現在の ISO 規格案は，Pascal User Manual and Report で記述されている標準 Pascal に対して手続きパラメータと動的配列 (conformant array) を除いては仕様の変更と拡張を，基本的に行わず，ただ User Manual and

* スイス連邦工科大学

Report を厳密化しようとしているだけである．Pascal の設計思想を守る上では，これで十分かも知れないが，Pascal の実用的な応用を考える上では，言語の拡張の問題を避けて通るわけにゆかない．次章において，この点を論ずる[10),11)]．

5. 拡張に対する考え方

Pascal 言語の拡張の是非を論ずる上で，次の3つの要請をおく：

（i） Pascal 言語の設計思想を保つこと

（ii） Pascal が広く使われる言語となること

（iii） Pascal が実用的用途に使用可能であること

以上の要請は，必ずしも万人に通用するものではない．たとえば，Pascal をベースに Pascal 類似の言語を作ろうとする人々，実験用・研究用のみに Pascal を用いる人々，あるいは，ある閉鎖社会の中でのみ Pascal を使用しようとする人々にとっては上記の要請は，必ずしもあてはまらない．しかし今は仮に，上の要請を認めた上で，Pascal 言語の拡張についての議論を進めることにする．

まず，第1の Pascal 言語の設計思想を保つためには，Pascal の簡潔さ，信頼性，抽象性，保守性，効率等を失なわせ，これらの統一をこわすような拡張はすべきでない．特に，機能追加は，複雑さを増すことになり，安易に機能を増やすことは慎しみたい．この点は，第3の要請である．実用的用途に使用可能とする，という点と一見矛盾しがちであるが，機能を，ある用途にとって必要不可欠なものと，あれば便利なものとに分けて考え，後者に対してはなるべく拡張を避ける方針をとることにより両者の均衡をとることが大切であろう．特に，従来の標準 Pascal の構文や意味の変更となるもの*や，見出し語 (key word) の追加**となるような拡張は，極力行わないのが望ましい．

次に，Pascal が広く使われるようになるためには，処理系が極力優れたものであるように務めることが重要であろう．たとえば，数値計算のためには，最適化は不可欠であり，数学ライブラリや標準関数の拡充も必要であろう．準システムプログラム向けには，packed 型や可変レコード型のきっちりしたメモリ割付けや，HEAP 領域に対するガーベジコレクションなどもできるだけ行いたい．また，プロファイル，ダンプ，シンボリックデバッグ等のデバッグ機能の充実なども幅広く使われるために必要なことであろう．この点については，処理系が良くなればユーザの数が増えるという関係だけでなく，逆に，ユーザの数が増えれば処理系も良くなる，という相乗効果がある．

第3に，実用的な用途で使用可能な言語とするためには，他言語との連絡機能，分割コンパイル，パラメータにおける動的配列，そのほかの若干の機能拡張は必要と思われる．まず，実用的なプログラムでは，Pascal で表現しきれない部分が含まれることがあり，アセンブラや FORTRAN 等と連結できることが望ましい．この，他言語との連結機能をもった Pascal 処理系は現在でも少なくない．また，実用的なプログラムでは，数十 kstep にも及ぶこともめずらしくないので，分割コンパイルも必要である．現時点では，分割コンパイルはできても，グローバル変数に対するアクセスやインタフェースチェックを行う処理系はさほど多くない．この点は，将来の課題であろう．なお，Pascal に own 変数やコモン変数などの静的な変数を導入している処理系があるが，これらは本来グローバル変数として扱われるべきものであり，こういった拡張は望ましくないと思われる．また，パラメータにおける動的配列は，ISO 案でも，conformant array という形で導入されているが，数学関数ライブラリなどの応用に対して必要不可欠であろうと思われる．そのほか，直接アクセスファイルなども実用目的のプログラムに対しては必要であり，拡張は止むを得ないであろう．

しかし，これらの拡張を行う上で，処理系の範囲で済むものは良いが，規格に触れるものに対しては，特別の注意が必要である．すなわち，規格に抵触する拡張で必要不可欠なものは，できるだけ，① 隠しオプションにする，② 使用するユーザに拡張であることを意識して使わせる，③ 将来の規格との相違に備え，使用範囲を局限する，といった配慮が必要である．必要不可欠な拡張は，なるべく早い段階で将来の Pascal 規格に含め，標準化を図っていくことも大切である．

制御用プログラムや，OS の核の部分など特殊用途の機能，たとえば，マクロの呼出し，並列処理，例外処理，メモリへの直接アクセス，型の無視等は，一般ユーザには不用な機能であり，かつ Pascal を複雑にさせるので，たとえば隠しオプションとすることが望ましい．

---

* たとえば，演算子の順位変更や FOR 文制御変数の有効範囲の局所化等．

** たとえば，CASE 文中の otherwise 節や，ループ中の escape 等．

6. あ と が き

我々が Pascal に期待するのは，従来の言語に対するような機能の豊富さではなく，簡潔であるとか，信頼性が高いとかいった使い心地の良さである．実用的な見地からは，言語の普及の範囲が広いとか，処理系が小型で済みかつ作りやすいとかいった利点もある．

Pascalの適用分野は研究用や教育用から始まり，実用目的で注目され出したのは比較的最近である．その用途も一般の応用プログラムのほか，言語処理システムなど準システムプログラムの範囲までである．しかし，Pascalの実用化が進み，適用範囲が広がってくると，OS の核の部分など純システムプログラムへの適用も考えられてくる．

Pascalの規格制度の動きも急ピッチで進んでいる．Pascalのもつ利点を壊さないためにも，また，標準化を維持するためにも，不要の拡張は避け，必要な拡張は互換性を失わない形で行うという配慮が望ましい．

Pascalは実用目的のプログラミングに十分耐えうる言語であり，そのためにも，規格の早期制定と標準化の推進，処理系の拡充，ユーザ層の拡大，将来におけるごく一部の機能拡張（特に，分割コンパイルの本質的解決）などが望まれる．

参 考 文 献

1) Jensen, K. and Wirth, N.: Pascal User Manual and Report, p. 170, Springer-Verlag (1974).
2) Fletcher, D.: Pascal Power-Users loves it, Vendors are getting the message, and Standards are on the way, Datamation, pp. 142-145 (July 1979).
3) Schneider, G. M.: Pascal: An Overview, Computer, pp. 61-66 (Apr. 1979).
4) Bate, R. R. and Johnson, D. S.: Language extensions, utilities boost Pascal's Performance, Electronics, pp. 111-121 (June 1979).
5) ISO/TC 97/SC 5/WG 4: Specification for the Computer Programming Language Pascal, revised version of 97/5 N 565, p. 74 (Aug. 1980).
6) Hoare, C. A. R.: Hints on Programming Language Design, Stanford Artificial Intelligence Laboratory Memo 224, Computer Science Department Report No. CS-403 (Oct. 1973).
7) Myers, G. J.: Software Reliability, p. 360, John Wiley & Sons (1976).
8) Richard, F. and Ledgard, H. F.: A Reminder for Language Designers, Sigplan Notices, Vol. 12, No. 12, pp. 73-82 (Dec. 1977).
9) Wirth, N.: An Assessment of the Programming Language Pascal, Sigplan Notices, Vol. 10, No. 6, pp. 23-30 (June 1975).
10) 野下浩平：Pascal の処理系について，東京大学大型計算機センターニュース，Vol. 11, No. 4, pp. 53-55 (1979).
11) 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 (Feb. 1980).
12) 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，pp. 100-131 (Dec. 1976).

（昭和 55 年 11 月 20 日受付）