拡張 Pascal の概要

筧 捷彦
早稲田大学理工学部

現在，ISO/TC97/SC22/WG2 によって作業中の拡張 Pascal について，その概要を紹介する。紹介は，資料 N262 に基いたものである。

Introduction to Extended Pascal

Katuhiko KAKEHI
Centre for Informatics, Waseda University
Okubo 3, Tokyo 160, JAPAN

Based on the document N262 produced by ISO/TC97/SC22/WG2, features of Extended Pascal are summarized. The purpose of this article is only an introduction to Extended Pascal. Therefore, anyone who wants to know full details should consult directly with the document N262.

１．拡張Ｐａｓｃａｌの制定作業

１．１　作業状況

１９８３年に国際規格[2]となったＰａｓｃａｌは、その定義本体を英国規格協会(BSI)の規格[3]によったものである。その制定にあたっては、標準的なＰａｓｃａｌ[5]のまま規格とするか、実用上の便宜を考えて若干の拡張を加えた上で規格とするかで論争があった。結局、「整合配列」の拡張を含んだ英国案が国際規格となったが、この拡張を含むか含まないかで二つの水準を設けることになったのである（[4]参照）。

各国の規格は、通常、国際規格に即して制定される。ところが、拡張に反対していた米国では、「整合配列」の拡張を含まないものを、国内規格として制定し、機能拡張については別途に規格を定める方向で動きはじめた。ＩＥＥＥでは、１９８４年に早くも拡張Ｐａｓｃａｌの規格原案を作成している。

この動きに呼応して、ＩＳＯ／ＴＣ９７／ＳＣ２２のもとにＷＧ２が設けられ、拡張Ｐａｓｃａｌの国際規格草案作成が進められてきた。１９８７年末になって草案[1]が各国に配布され、これＤＰとして採択するか否かの投票が行なわれた。

ＩＳＯの規格化作業は、通常、つぎのように進行する。

ＷＤ(Working Draft)　－＞　ＤＰ(Draft Proposal)　－＞　ＤＩＳ(Draft International Standard)　－＞　ＩＳ(International Standard)

それぞれ次の段階に移行するには、各国による投票が必要であり、全過程を経るには数年かかるのが常である。もとろん、この過程を短縮する手順も存在してはいるが、拡張Ｐａｓｃａｌの場合にそれを採るかどうかは明らかでない。

このように、拡張Ｐａｓｃａｌの国際規格が究極的にどうなるかは、今の時点では必ずしも予測できるものではないが、本稿では[1]の内容に基づいて、拡張Ｐａｓｃａｌの概要を紹介する。

１．２　Ｐａｓｃａｌ拡張の要求

標準Ｐａｓｃａｌ[5]の処理系が広く使われるようになるにつれ、実用上の要求からいろいろな拡張が求められたし、そのような拡張を施した処理系もいろいろと開発されてきた。その主たるものは、Ｐａｓｃａｌで大型のソフトウェアを開発する際の、種々の不便さに起因するものであった。特に、Ｐａｓｃａｌのプログラムが、”program”に始まり”end.”に終わる一つのテキストであることは、大型のソフトウェアを開発する際に用いられるモジュール化技法にそぐわないものであった。

モジュールとして、まとまったデータや手続き・関数などを組にして表現し、管理できるものが欲しい。その際、単にテキスト上での単位となるばかりでなく、それを分離翻訳して、オブジェクトライブラリとして管理できることが望まれる。勿論、Ｐａｓｃａｌがもつ強い型付けの機能を損なわないものとしたい。

ライブラリとする際の困難なことがらに、型のパラメタ化ができないことがあった。特に、配列型が、その添字型を固定している点が、一番の問題であった。たとえば、行列演算を行なう手続きを用意しようにも、その利用者が任意に寸法を指定できるようにすることは、標準Ｐａｓｃａｌでは不可能であった。そこで、現在の国際規格[2,3]では、「整合配列」を導入して、手続きのパラメタが、任意寸法の配列を受け取れるようにしたのであった。

プログラムが一つのテキストから成る、という方針であれば、これでかなりの自由度を得ることができる。その配列を具体的に宣言するのは、そのテキストの中であり、利用者自らが宣言を書くからである。

ところが、モジュールという独立したテキストを置くことにすると、別の問題が生じる。たとえば、行列演算を行なうモジュールを考えてみよう。そのモジュール単独で、”行列”型を定義し、それを使って、そのモジュール中の各手続きや関数を記述したい。しかも、その”行列”型の行列の寸法は、そのモジュールを使う利用者が任意に指定できる仕掛けにしておきたいのである。

これらの他にも、識別子に英数字しか使えないので長い識別子が読みにくい、宣言を書く順番が固定されているのは不便である、入出力に直接アクセスに当るものがない、プログラムでのファイル変数を外界のファ

イルに結び付ける機能がない、関数の結果としてレコードや配列を返すことができない、複素数型がない、などの不満があった。

こうした要求に答える形で原案[1]が作られている。

さらに、並列処理を記述したい、例外処理を記述したい、ハードウェアの細部にアクセスする機能が欲しいなど、要求としてはいろいろあるが、原案[1]では取り上げていない。これらをすべて取り込んだ言語とし、すでにＡｄａがあることを考慮した結果なのであろう。

２．モジュール構造

２．１　プログラムとモジュール

プログラムは、これまでどおりに" program" で始まり" end." で終わる一つのテキスト（主プログラム）だけから成っていてもよいし、それに加えていくつかの必要なモジュール宣言を加えてもよい。

モジュールの利用は、それぞれのモジュールが輸出している名前を輸入することによって行なう。名前の輸入は、任意のブロックで行なうことができる。

拡張Ｐａｓｃａｌの文法は、以上のことしか定めていない。しかし、これを字句どおり、主プログラムとモジュール宣言とをテキストとして一括して翻訳することを要求しているのだ、と解釈するには当らない。翻訳をどうするか、といったことは文法では何も定めていないからである。Ｆｏｒｔｒａｎの文法も、同様の方式になっていることを思い起こせば、事情は明らかであろう。分離翻訳を初めとして、モジュール（や、それらのライブラリ）をどう管理するかは、すべて処理形に委ねられているのである。

標準のモジュールとして、標準入力ファイルinputを与えるStandardInputと、標準出力ファイルoutputを与えるStandardOutputとがある。ただし、これまでどおりに、プログラムパラメタとしてinputやoutputを指定するだけでこれらを引用できるようにもしてある。

２．２　モジュール単位

モジュール宣言は、それぞれ独立したテキストで、" module" で始まり" end." で終わる。一つのモジュールは、インターフェイスとインプレメンテーションの２部から構成される。モジュール宣言は、この２部を一度にまとめて記述したものか、それぞれを独立して記述したものか、のいずれかの形式をとる。

プログラム名は、これまでどおり、プログラムの中ではなんの働きももたない。モジュール名は、モジュール相互を区別するだけの働きをする。また、主プログラムやモジュールは、それぞれパラメタを付すことができる。これらのパラメタは、処理系によって外界の実体と結び付けられるもので、プログラム中での意味は、それぞれ変数宣言の形で宣言しておく。

モジュール宣言の形式

(1)一体形

```
module モジュール名
  [(パラメタ並び)] ;
  export  輸出仕様並び ;
 [import  輸入仕様並び ; ]
      . . . 各種宣言. . .
  end;
 [import  輸入仕様並び ; ]
      . . . 各種宣言. . .
end .
```

(2)分離形

```
module モジュール名 interface
  [(パラメタ並び)] ;
  export  輸出仕様並び ;
 [import  輸入仕様並び ; ]
      . . . 各種宣言. . .
end .

module モジュール名 implementation
  [import  輸入仕様並び ; ]
      . . . 各種宣言. . .
end .
```

一体形でも分離形でも、インターフェイス部で宣言したものは、インプレメンテーション部でそのまま引用できる。インターフェース部では、手続き・関数については、その仕様だけを宣言する。Ａｄａとは違い、インプレメンテーション部に本体に当るものがない。つまり、モジュールが提供する各種のデータなどを自動的に初期設定する機構は用意されていないことに注意しよう。

２．３　名前の輸出入

モジュールがその利用者に提供するものは、インターフェイス部の各種宣言で宣言しておいたものに限られる。モジュールによっては、利用者によりその一部だけを使う例が予想されるものもあるだろう。そこで、輸出する名前を適当にグループ分けすることもできるようにしてある。

輸出する名前をいくつかまとめたものを、拡張Ｐａｓｃａｌでは”インターフェイス”と呼ぶ。インターフェイスにはそれぞれ固有の名前を付ける。モジュール間での名前の輸出入は、このインターフェイス名を使って行なわれる。

輸出仕様は、インターフェイス名と、そのインターフェイス名の下にまとめて輸出する名前を並べたものである。このとき、モジュール内で使っている名前そのものを直接輸出してもよいし、適当に名前を付け変えて輸出してもよい。

```
例：  i1(a,b,c);          ――モジュール内の名前a,b,cそのものの輸出
      i2(a,b=>x,c);       ――a,x,cの輸出（xはbの名前を付け変えたもの）
```

輸入は、モジュール名を指定することによる。このとき、そのモジュール名の下に輸出されているすべての名前を輸入することもできるし、その一部の名前を輸入することもできる。また、輸出されている名前を付け変えて輸入することもできる。こうして輸入した名前は、原則として、モジュール名を付して引用する。

```
例：  i1.a    i2.a    i2.x
```

もちろん、輸入した名前が他のものと重ならないことが明らかなときは、名前単独での引用を許すように指定することもできる。Ａｄａと違い、名前の多重定義を許してはいないから、こうした名前の使い方の制御は、すべて利用者に負わされているのである。

輸入仕様は、以上の輸入の関する指定を与えるものである。

```
例：  i1*;                ――a,b,cすべてを輸入し、単独での引用を許す
      i1*(a=>p);          ――上と同じ。但し、aはpと名前を付け変える
      i1.*;               ――a,b,cすべてを輸入するが、単独での引用を許さない
      i1.*(b=>q);         ――上と同じ。但し、bはqと名前を付け変える
      i1(a,b);            ――a,bだけを輸入し、単独での引用を許す
      i2.(x=>z);          ――xだけを輸入し、zと名前を付け変える。単独での引用を許さない
```

輸出入の対象となる名前は、定数名、変数名、手続き名、関数名のほかに、型名も含まれる。さらに、次節で述べるスキーマ名も対象となる。

３．スキーマ

３．１　スキーマの定義

スキーマとは、型をパラメタ化するためのものである。スキーマのパラメタをディスクリミナントという。ディスクリミナントは、順序型の値に限られているし、それが使えるのも、本質的には部分範囲の上下限の指定に限られているから、そのパラメタ化はごくごく限られている。また、スキーマは必ずディスクリミナントをもたなければならない。

```
例：    matrix(L,M:positive) = array[1..L,1..M]of real;
        square(N:positve) = matrix(N,N);
        sq = square;                        ――スキーマ名の付け直し
```

３．２　スキーマによる型定義

スキーマ名に、ディスクリミナントとなる値を書き添えたものは、通常の書下し型と同じ扱いを受ける。つまり、それぞれ違う型を表わすのである。

例：　a: matrix(10,10);　b: square(10);　c: square(10);　－－a,b,cはそれぞれ別の型をもつ

スキーマを使った書下し型は、必ずすべてのディスクリミナントの値を指定しなければならない。ディスクリミナントの値を指定しない限り、スキーマそれ自体は型とはならない。したがって、なにかの型定義の中で使うことも、変数宣言の型として使うことも許されない。例外として、ポインタ型の被指示型としてスキーマ名を使うことだけは許されている。

例：　anymat = ^matrix;　－－独立した型。new(X,10,3)のようにディスクリミナントを指定して変数を作り出す

－－こうして作り出された変数は、たとえ同じディスクリミナントによっていても、それぞれ別の型となってしまうから、x^:=y^；などの代入さえ許されないことに注意すること。

３．３　手続き・関数のパラメタとスキーマ

手続きや関数のパラメタとして、同種の型のものを受けてこられるようにする機構としては、現在の規格を引き継いで「整合配列」をそのまま残している。これは、あくまで水準１とした場合のことで、水準０にはふくまれない。

一方で、スキーマによる型のパラメタ化を導入したので、これに呼応する形での同種の型の受け取りを導入する必要があった。これには、パラメタの型指定にスキーマ名を直接書くことにしたのである。当然のことながら、二つ以上のパラメタがあるとき、同じスキーマから作られた別の型のものを許すのか、同じ型のものに限るかの指定ができるようにしたい。同じパラメタ区域に並べたパラメタは、同じ型であることを要求し、別のパラメタ区域に並べたパラメタは、たとえ同じスキーマ名でも別の型であってよいことにする。しかし、これだけでは、変数パラメタと値パラメタのように、構文上、別のパラメタ区域に書くことが義務づけられているものについて、同じ型を要求することができない。そこで、他のパラメタの型を直接書き表わす表記法を導入したのである。

例：　procedure inverse(A:square; var B:A?type);　－－AとBは同じ型でなければならない

もちろん、本体で作業用の変数を取るときにも同様のことをしたいから、一般に型を書かなければならないところには、”名前 .type”という表記法を使ってよいことにし、定数名、変数名、パラメタ名の表わす型を直接書き表わせるようにしたのである。

こうして、スキーマから作られた型の変数や値をパラメタとして受け取ってきても、その型を作り出したさいのスキーマに対するディスクリミナントの値を知らなければ、作業のしようがないことがおおい。そこで、スキーマから作り出された型をもつ変数に対して、ディスクリミナント名を指定して、そのディスクリミナントの値そのものを取り出すための表記法を設けた。

例：　A.N　　X^.M　　X^.L

３．４　文字列型の扱い方

これまでは、packed array[]of char　の形のものを概念的に文字列型とみなして扱ってきたが、いわば、固定長の文字列であり、他の言語に見られるような、可変長の文字列も扱えるように拡張が行なわれた。つまり、最大長をパラメタとするスキーマ string を標準のスキーマとして置き、これから作られる型を可変長文字列型として扱うことにしたのである。

もちろん、これまでどおりに文字列型どうしでの融通をきかせるため、可変長であれ、固定長であれ、いずれも文字列型として、互いに型が整合するものとした。また、代入についても、代入しようとする値の文字列の（実際の）長さが、代入先の変数の最大長以下であれば、代入可能とした。もちろん、固定長の文字

列型では、長さは最大長に等しいとみるのである。

文字列の比較では、＝　＜　＞　＜＞　＜＝　＞＝　を使ったときは、末尾の空白を除いて辞書式順序によることにし、長さも含めての（つまり末尾の空白も含めての）辞書式順序によるものとして、関数EQ,LT,GT,NE,LE,GEを設けた。また、文字列変数からの部分列の切り出し（S[3..8]などと書く）を許し、代入文の左辺にも使えるようにした。

また、文字列用の標準関数として、length(s)－－長さを返す、index(s1,s2)－－s1中でのs2の最初の出現位置を返す、trim(s)－－末尾の空白を取り除いた文字列を返す、を新設したほか、＋で文字列の連結演算を行なうようにしてある。

4．その他の拡張

4．1　宣言順序の自由化と初期値

現在の規格では、宣言の順序が固定されている。ラベル宣言、定数定義、型定義、変数宣言、そして手続き・関数の宣言の順である。しかし、この制限のため、互いに関連するものがバラバラにしか書けない、という不便さがあった。拡張Ｐａｓｃａｌでは、この順序に関する制限を撤廃した。もちろん、ポインタ型や手続き・関数宣言での限られた場合を除き、すべての名前は宣言してからでなければ引用してはならない、という制約はそのまま生きている。

変数の初期設定が宣言部に書くことができない、というのも、現在の規格に対する不満の一つであった。そこで、初期設定を宣言（ないし定義）の一つとして設けることにした。

例：
```
var  x, e: real;
value  x:= 3.14159;  e:= 2.71828;
```

単純型の値や集合型の値については、その値（定数値）を直接書き表わすものがあったが、配列やレコードについてはそれがなかった。そこで、これらの表記法を導入した。基本的には、一番外側に型名を書き、”[”と”]”とで囲って、添字またはフィールドの指定とその値とを書き並べるのである。このとき、いくつもの添字やフィールドを指定して一勢に同じ値を与えること、添字の範囲を指定して同じ値を与えること、配列については指定しなかった残り全部に同じ値を与えること、また、レコードについては可変部のタグフィールドの値を与え、それに合せて特定の可変要素として値を与えることができるようにした。

例：
```
type  sq3  = sq(3);
      kind = (Tinteger,Treal);
      intreal = record
                  id_No: integer;
                  case k:kind of
                    Tinteger: (intval: integer);
                    Treal:    (realval: real  )
                end;
      setir  = set of kind;
```

の型定義の下で
```
sq3 [1:[1:1.0;2..3:0.0]; 2:[1:0.0;2:1.0;3:0.0]; 3:[1..2:0.0;3:1.0]]  －－単位行列
sq3 [1:[1:1.0;oterwise 0.0]; otherwise [otherwise 0.0]]  －－(1,1)要素だけ1.0，他は0.0
intreal [id_No=103; case k=Treal; realval=2.2360679]
setir []
```

集合型の場合は、現在の規格との整合性から、型名を前置しないものも許されている。

こうした構造型の値の表記法では、指定する添字やタグフィールドの値が定数であれば、要素それぞれに

与える値の指定は一般の式でもよい。このようなものは、式の中で構造型の値を表わすのに使うことができる。値の指定が定数（または定数の固定した要素）から成る式のとき、特に定数式という（標準の関数のうち数学的な関数なら使ってもよい）。

初期設定では、定数式を使う。また、定数定義では" =" の右辺は定数式であればなんでもよいことになった。つまり、構造型の定数名を定めることが可能となったのである。

例：　const null = chr(0); zero = sq3 [oterwise [otherwise 0.0]];

４．２　入出力の拡張

既存のファイルに対して、末尾に書き加えることができるようになった。このためには、rewrite に代えて extend(f) として始めればよい。extend は、ファイル位置を末尾（の次）に置くだけのことで、それ以外は rewrite と変わることがない。

直接アクセスのファイル型が新たに導入された。

例：　var da: file[1..100]of intreal;

これに合せて、モードとして、" 検査" 、" 生成" に加えて" 変更" が追加された。次に読み書きするべき位置を指定するための手続きとして、seekread(f,n), seekwrite(f,n) が設けてある。このとき、位置の指定 n はその時点での実際に書かれている位置のいずれかか、その最後の位置の次の位置かのいずれかでなければならない。こうしたその時点での書かれている位置の情報を得るために関数 position(f) と lastposiston(f) とが用意され、さらにファイルが空かどうかを調べるための論理型関数 empty(f) も用意されている。さらに位置決めには seekupdate(f,n) があり、モードを" 変更" に変える（" 生成" か" 変更" かになっているものに限る）ことができる。put(f) や update(f) は" 生成" か" 変更" かのモードの下で使うことができ、get(f) は" 検査" か" 変更" のモードで使うことができる。update(f) は位置を変えずに書き換える（末尾の次に位置するときは次の位置に書き加える）。put(f) はその位置のデータを書き換え（末尾の次に位置するときは書き加え）、get(f) はその位置のデータを読み取る。どちらも位置が一つ先に進む。

get(f) も put(f) も、これまであった直接アクセスでないファイルに対する操作と整合していることに注意しよう。直接アクセスでないファイルに対しても update(f) を使うことも許されているが、もはやその後は、get, put, update のいずれも適用できなくなる。

外界にあるファイルとの結び付けを直接的に制御するための手続きも用意された。このための型 BindingType を設けた。この型は、レコード型で、少なくとも、外界のファイルをしていする名前を収めた文字列型のフィールド name と外界のファイルと結び付いているかどうかを示す論理型のフィールド bound とをもつ。いわゆる" オープン" に当るのが手続き bind(f,b) であり、b には name フィールドの値を指定した BindingType 型の値を書く。" クローズ" にあたる手続きとしては unbind(f) がある。

外界のファイルとの結び付きを知るために、関数 binding(f) があり、その時点でのファイル変数 f についての Bindingtype の値を返す。

こうした外界との関連として、TimeStampType が標準的に設けられている。この型はレコード型に相当し、年、月、日、時、分、秒にあたるフィールド Year,Month,Day,Hour,Minute,Second と、年月日が正当であるかどうか、時分秒が正確であるかどうかを示す論理型のフィールド DateValid,TimeValid とをもつ。呼出しの時点での正確な TimeStampType 型の値を返す関数 timestamp と、TimeStampType 型の値から、年月もしくは時分秒を取りだし文字列として表現して返す関数 date(t), time(t) も合せて用意されている。

また、read文やwrite文では、ファイルを対象とするほかに、文字列を対象とすることも許した。

４．３　複素数型の導入

数値計算用としては、Ｆｏｒｔｒａｎ並に複素数も扱いたい、という要求があった。これに答えて、複素数型を新設し、complex という型名を標準的に設けた。＋－＊／の演算はもちろん、これまでにもあった関

数 abs,sqr,sin,cos,exp,ln,sqrt,arctan も複素数型に対して適用できることに拡張した。複素数型専用として、偏角、実部、虚部を取り出す関数 arg,re,im を設けるとともに、複素数値を作り出す関数 cmplx(r,i), polar(r,theta) を用意した。

4．4　結果型の拡張

結果型としては、ファイル型およびファイル型を要素にもつ型以外であれば、任意の型を許すことにした。こらに合せて、関数の結果として返された値の要素を取り出すことも許した。

また、これまで関数の結果を与えるのには、関数名に対する代入という形を用いていたが、この他に結果を表わす名前の導入も認め、この名前に対する代入でも結果を指定できるようにした。

例：
```
function imaginary(i:real)=c: complex;
begin  c:=cmplx(0.0,i) end;
```

4．5　その他

・識別子の途中に"_"を使ってもよいことにした。これは、長い名前の読みやすさを考慮したものである。

・整定数として、３６進までの任意の基数を許した。

例：　16#FFFF　8#7623　36#1ZR

・処理系の定める値を知るためのものとして、これまでの Maxint に加えて、Maxchar,MinReal,MacReal,Epsreal という標準の定数名を設けた。

・case文やレコード型の可変部に、範囲の指定を許すことにし、また、指定のなかったその他の値を指示するために otherwise という予約語が使えるようにした。

・集合型について、対称差を求める演算 <> と補集合を求める演算 - とを設けたほか、集合の要素数を求める関数 card(s) を設けた。また、集合のすべての要素に関する繰返しのために for e in s という形式を新設した。

・順序型について、与えられた値から相対的に何個か前または後にある値を返す関数 relval(o,n) を設けた。

・プログラムの実行を中止するための手続き halt を新設した。

5．まとめ

以上に見てきたように、拡張Ｐａｓｃａｌの現規格に対する大幅な拡張は、モジュールとスキーマおよび直接アクセスのファイルを設けたことに尽きる。これを除いては、いずれも現規格に取り込んでもその本質を変えない程度のものである。

参考文献

[1]ISO/TC97/SC22, Working Draft Programming Language Extended Pascal, N262, Nov.1986.

[2]ISO, Programming Languages--Pascal, ISO7185, 1983.

[3]BSI, Specification for Computer programming langauge Pascal, BSI6192, 1982.

[4]石畑清他、Ｐａｓｃａｌの標準化－ＩＳＯ規格全訳とその解説、bit別冊、共立出版、1984.

[5]Jensen,K. and Wirth,N., Pascal－user manual and report, Lecture Note on Computer Science, Springer, Vol.18, 1974.　邦訳（原田賢一）、Ｐａｓｃａｌ、培風館、1981.