SLAC–PUB–5505
April 1991
(M)

Status of Pascal Standardization*

BEBO WHITE
Stanford Linear Accelerator Center
Stanford University, Stanford, CA 94309

Presented at SHARE 76, San Francisco, CA, February 24–March 1, 1991

* Work supported in part by the Department of Energy contract DE–AC03–76SF00515.

Features of Extended Pascal

° Modularity and Separate Compilation

° Schemata

° String Capabilities

° Binding of Variables

° Direct Access File Handling

° File Extend Procedure

° Constant Expressions

° Structured Value Constructors

° Generalized Function Results

° Initial Variable State

° Relaxation of Ordering of Declarations

° Type Inquiry

° Implementation Characteristics

° Case-Statement and Variant Record Enhancements

° Set Extensions

Even More Features....

- Date and Time
- Inverse ORD
- Standard Numeric Input
- Non-Decimal Representation of Numbers
- Underscores in Identifiers
- Zero Field Widths
- Halt
- Complex Numbers
- Short Circuit Boolean Evaluation
- Protected Parameters
- Exponentiation
- Subranges Bounds
- Tag Fields of Dynamic Variables
- Conformant Arrays

Example of Modularity

```
module  employee_sort  interface;

    export employee_sort =
(sort_by_name,sort_by_clock_number,employee_list);

    import generic_sort;

   type
      employee = record
          last_name,first_name : string(30);
         clock_number : 1..maxint;
     end;

        employee_list(num_employees : max_sort_index) =
          array [1..num_employees] of employee;

    procedure sort_by_name(employees : employee_list;
              var something_done : Boolean);

    procedure sort_by_clock_number(employees : employee_list;
              var something_done : Boolean);

end.
```

Modularity

° Each module exports one or more interfaces containing entities (values, types, schemata, variables, procedures, and functions) from that module, thereby controlling visibility into the module.

°A variable may be protected on export, so that an importer may use it but not alter its value. A type may be restricted, so that its structure is not visible.

° The form of a module clearly separates its interfaces from its internal details.

° Any block may import one or more interfaces. Each interface may be used in whole or in part.

° Entities may be accessed with or without interface-name qualification.

° Entities may be renamed on export or import.

° Initialization and finalization actions may be specified for each module.

° Modules provide a framework for implementation of libraries and non-Pascal program components.

Example of Schemata

```
type
   SWidth = 0..1023;
   SHeight = 0..2047;
   Screen(width: SWidth; height: SHeight) =
           array [0..height, 0..width] of boolean;

   Matrix(M,N: integer) = array [1..M,1..N] of real;

   Vector(M: integer) = array [1..M] of real;

   Color = (red,yellow);
    Color_Map(formal_discriminant: color) =
     record
      case formal_discriminant of
     red: (red_field : integer);
     yellow : (yellow_field : integer);
    end;

function bound : integer;
 var s : integer;
   begin
    write('How big?');
    readln(s);
   bound := s;
   end;

var
    My_Matrix : Matrix(10,10);
    My_Vector : Vector(bound);   { Notice the run-time expression! }
   Matrix_Ptr : ^Matrix;
  X,Y : integer;

   begin
    readln(x,y);
    new(Matrix_Ptr,X,Y);
   end
```

Schemata

° Statically selected types are uses as any other types are used.

° Dynamically selected types subsume all the functionality of, and provide functional capability beyond, conformant arrays.

° The allocation procedure NEW may dynamically select the type (and thus the size) of the allocated variable.

° A schematic formal-parameter adjusts to the bounds of its actual-parameters.

° The declaration of a local variable may dynamically select the type (and thus the size) of the variable.

° The with-statement is extended to work with schemata.

° Formal schema discriminants can be used as variant selectors.

String Capabilities

° All string and character values are compatible.

° The concatenation operator (+) combines all string and character values.

° String may be compared using blank padding via the relation operators, or using no padding via the functions EQ, LT, GT, NE, LE, and GE.

° The functions LENGTH, INDEX, SUBSTR, and TRIM provide information about, or manipulate, strings.

° The substring-variable notation makes accessible, as a variable, a fixed-length portion of a string variable.

° The transfer procedures READSTR and WRITESTR process strings in the same manner that READ and WRITE process textfiles.

° The procedure READ has been extended to read strings from textfiles.

Binding of Variables

° A variable may optionally be declared to be bindable. Bindable variables may be bound to external entities (file storage, real-time clock, command lines, etc.). Only bindable variables may be so bound.

° The procedures BIND and UNBIND, together with the related type BINDINGTYPE, provide capabilities for connection and disconnection of bindable internal (file and non-file) variables to external entities.

° The function BINDING returns current or default binding information.

Direct Access File Handling

° The declaration of a direct-access file indicates an index by which individual file elements may be accessed.

° The procedures SEEKREAD, SEEKWRITE, and SEEKUPDATE position the file.

° The functions POSITION, LASTPOSITION, and EMPTY report the current position and size of the file.

° The update file mode and its associated procedure UPDATE provide in-place modification.

Set Extensions

° An operator (><) computes the set symmetric difference.

° The function CARD yields the number of members in a set.

° A form of the for-statement iterates through the members of a set.

Complex Numbers

° The simple-type COMPLEX allows complex numbers to be expressed in either Cartesian or polar notation.

° The monadic operators + and - and dyadic operators +, -, *, /, =, [and] <> operate on complex values.

° The functions CMPLX, POLAR, RE, IM, and ARG construct or provide information about complex values.

° The functions ABS, SQR, SQRT, EXP, LN, SIN, COS, [and] ARCTAN operate on complex values.

Object-Oriented Extensions to Pascal

X3J9 is in the process of developing a technical report on object-oriented extensions to Pascal. This work is being done in cooperation with major vendors of Pascal and with users of Pascal. This report is expected to be completed by the 4th quarter, 1991.

Scope of Object Pascal Study

- ° Definition of Objects
- ° Inheritance of characteristics of one object by another
- ° Overriding of inherited characteristics
- ° Creation and initialization of instances of objects
- ° Destruction and removal of objects
- ° Sharing of code between objects
- ° Dynamic and static selection of object methods
- ° Security of objects
- ° Extension of existing language features for objects
- ° New functions for dealing with objects
- ° Integration with existing language features in Pascal