The Pascal Programming Language
Bill Catambay, Pascal Developer

Chapter 3

The Pascal Programming Language
by Bill Catambay

Return to Table of Contents

III. Pascal Standards

Language standards were developed so that when code is written on one type of computer, or with one vendor's compiler, the code can be ported to another computer or compiler and still compile and run correctly. This is not a foolproof plan, however, as there are many unique behaviors of different computers. A program that invokes the unique behavior of a specific computer will have to be changed to work on another. Further, most computer environments offer libraries for performing commonly used functions. If the program makes several calls to the system libraries, those calls will most likely need to be replaced when porting the code to another computer, even if the compiler is fully compliant with its languages standards.

Although standards do not resolve the above issues, they do provide a certain amount of consistency within a language construct. The same code may not function, or perhaps even compile, when ported directly to another computer or compiler, but because the standards exist, a programmer familiar with that language will have a basic understanding of what the code is doing. Given an understanding of the libraries and unique properties of the computer to which the code is being ported, the task of porting the code is easier than if there were no language standards at all.

The first standard written for Pascal was developed in 1983, covering what is known as unextended Pascal (ISO 7185). In 1990, the same year that the unextended Pascal standard was updated, the Extended Pascal standard (ISO 10206) was established. The unextended Pascal standard incorporated basic functionality of the original Pascal, while the Extended Pascal standard was introduced to bring Pascal more in line with modern programming needs, thus providing the programmer a more powerful programming tool without sacrificing the elegance of Pascal.

To further meet the demands of the growing technology in computer programming, certain Pascal compilers were established to support Object-Oriented Programming. Although an official standard for Object Pascal has not been established at this writing, in 1993, the Pascal Standards Committee published an "Object-Oriented Extensions to Pascal" Technical Report which provides proposed standards. The members of the committee that assembled this report came from a variety of organizations - from Pace University and the US Air Force, to Apple Computer,

Microsoft, and Digital Equipment Corporation.

Finally, in 1995, John Reagan, a member of the ISO Pascal standards committee and compiler architect for Digital Equipment Corporation (now Compaq), composed a Pascal Standards FAQ. The FAQ addressed questions such as:

- What are the different Pascal standards?
- Who creates the standards?
- What are the required interfaces to Extended Pascal?
- What is the History of Pascal Standards?

All of the standards and reports mentioned above, including the Pascal Standards FAQ, are available from the Pascal Central web site at http://pascal-central.com.

Direct links to specific documents are as follows:

- Unextended & Extended Pascal Standards: http://pascal-central.com/standards.html
- Object Pascal Report: http://pascal-central.com/ooe-stds.html
- Pascal Standards FAQ: http://pascal-central.com/extpascal.html

Unextended Pascal

The unextended Pascal standard is incorporated in ISO 7185. The material covered in the standard is too extensive to discuss here, but the following section provides a summary of ISO 7185.

Unextended Pascal token symbols are reserved symbols used by the compiler for performing operations and calculations (see Table I below).

+	(PLUS)	Binary arithmetic addition; unary arithmetic identity; set union.
-	(MINUS)	Binary arithmetic subtraction; unary arithmetic negation; set difference.
*	(ASTERISK)	Arithmetic multiplication; set intersection.
/	(SLASH)	Floating point division.
=	(EQUAL)	Equality test.

<	(LESS THAN)	Less than test.
>	(GREATER THAN)	Greater than test.
[	(LEFT BRACKET)	Delimits sets and array indices.
]	(RIGHT BRACKET)	Delimits sets and array indices.
.	(PERIOD)	Used for selecting an individual field of a record variable. Follows the final END of a program.
,	(COMMA)	Separates arguments, variable declarations, and indices of multi-dimensional arrays.
:	(COLON)	Separates a function declaration with the function type. Separates variable declaration with the variable type.
;	(SEMI-COLON)	Separates Pascal statements.
^	(POINTER)	Used to declare pointer types and variables; Used to access the contents of pointer typed variables/file buffer variables.
(	(LEFT PARENTHESIS)	Group mathematical or boolean expression, or function and procedure arguments.
)	(RIGHT PARENTHESIS)	Group mathematical or boolean expression, or function and procedure arguments.
<>	(LESS THAN/GREATER THAN)	Non-equality test.
<=	(LESS THAN/EQUAL)	Less than or equal to test; Subset of test.
>=	(GREATER THAN/EQUAL)	Greater than or equal to test; Superset of test.
:=	(COLON/EQUAL)	Variable assignment.

..	(PERIOD/PERIOD)	Range delimiter.

Table I: Unextended Pascal Special Symbol Tokens

Additionally, the following tokens are Pascal comment delimiters. When placed around text, they indicate text that is not meant for compilation.

{ and }, (* and *)

Unextended Pascal token words are reserved words used by the compiler for performing operations and calculations (see Table II below).

AND	Boolean conjunction operator
ARRAY	Array type
BEGIN	Starts a compound statement
CASE	Starts a CASE statement
CONST	Declares a constant
DIV	Integer division
DO	Follows WHILE and FOR clause, preceding action to take
DOWNTO	In a FOR loop, indicates that FOR variable is decremented at each pass
ELSE	If the boolean in the IF is false, the action following ELSE is executed
END	Ends a compound statement, a case statement, or a record declaration
FILE	Declares a variable as a file
FOR	Executes line(s) of code while FOR loop variable in within range
FUNCTION	Declares a Pascal function
GOTO	Branches to a specified label
IF	Examine a boolean condition and execute code if true
IN	Boolean evaluated to true if value is in a specified set
LABEL	Indicates code to branch to in a GOTO statement
MOD	Modular integer evaluation
NIL	Null value for a pointer
NOT	Negates the value of a boolean expression
OF	Used in CASE statement after case variable
OR	Boolean disjunction operator

PACKED	Used with ARRAY, FILE, RECORD, and SET to pack data storage
PROCEDURE	Declares a Pascal procedure
PROGRAM	Designates the program heading
RECORD	Declares a record type
REPEAT	Starts a REPEAT/UNTIL loop
SET	Declares a set
THEN	Follows the boolean expression after an IF statement
TO	In a FOR loop, indicates that FOR variable is incremented at each pass
TYPE	Defines a variable type
UNTIL	Ends a REPEAT/UNTIL loop
VAR	Declares a program variable
WHILE	Executes block of code until WHILE condition is false
WITH	Specifies record variable to use for a block of code

Table II: Unextended Pascal Word Symbol Tokens

Unextended Pascal supports three categories of data types: **simple**, **structured**, and **pointer**.

The simple data type consists of predefined, enumerated and subrange types.

The predefined simple types are:

INTEGER - integer value
REAL - real number value
CHAR - single character
BOOLEAN - takes values true or false

In unextended Pascal, the enumerated type is defined to allow programmers to establish
types based upon a unique and finite list of values. For example:

```
Type
   ShirtColor   = (green, yellow, blue, red, orange);
   ShirtSize    = (small, medium, large, xlarge);
```

Likewise, unextended Pascal supports sub-range types, such as 1..100, -10..+10, and
'0'..'9'.

The structured data type consists of array, file, record. and set types. All four can be optionally declared as PACKED for cases where the programmer wants to minimize

storage requirements albeit with the potential cost of greater access time to the individual components.

An array in unextended Pascal consists of a fixed number of components (i.e., a linear vector) which are all of the same type. An array's index type establishes the number of components the array contains. Since any ordinal type can be used for the index type, arrays indexed by character, enumerated, or subrange types can be used for a more natural fit to the problem. Because a component type can be an array type in itself, multidimensional arrays are possible. For multidimensional arrays, a consolidated list of dimension indices can be used as an abbreviation alternative. For example:

```
Type
   ShirtStock   = array[ShirtColor] of array[ShirtSize] of integer;
   ShirtPrice   = packed array[ShirtColor, ShirtSize] of real;
```

The file type consists of a linear sequence of components all of the same type. The number of components is not fixed. Pascal predefines TEXT as a file type. The TEXT file type consists of a sequence of characters subdivided into variable length lines. A special end-of-line marker is used for line subdivision. Example file types are:

```
Type
   Document   = file of integer;
   Data       = packed file of real;
```

Unextended Pascal supports the RECORD type, which allows the programmer to establish a type containing several components (or fields). In addition to being able to declare fields of varying types, variant records can also be declared such that different record layouts exist based upon the value for the variant field (see Figure 5 below).

```
Type
   ShirtOrder = record
      color:      ShirtColor;
      size:       ShirtSize;
      customer:   string;
      city:       string;
      case USCust: boolean of
            true:  (USState:  packed array[1..2] of char);
            false: (state:    string;
                    country:  string);
      end;
```

Figure 5: Example of Variant Record

A set type consists of the powerset of the set of values of the set's base type. Every value in the set's base type can be represented as an element in the set. Example set types are:

```
Type
   CharSet   = set of char;
   SizeSet   = packed set of ShirtSize;
```

Unextended Pascal also supports pointer types, a method of referencing a variable

using its address. Pointers are often used for referencing record structures, supporting linked lists, an array of data which is connected by means of pointers rather than a sequential index (see Figure 6 below).

```
Type
   Person = ^Persondetails;
   Persondetails = record
      name:         String;
      firstname:    String;
      age:          Integer;
      married:      boolean;
      nextPerson:   Person;
      prevPerson:   Person;
      end;
```

Figure 6: Example of Linked List Record Structure

Refer to the ISO standard for a complete list of unextended Pascal predefined functions, types, and other language constructs.

Extended Pascal

The Extended Pascal standard is incorporated in ISO 10206. The material covered in this document is summarized below.

In addition to the token symbols reserved for unextended Pascal, Extended Pascal supports three additional tokens (see Table III below).

**	(ASTERISK/ASTERISK)	To the real power of
><	(GREATER THAN/LESS THAN)	Set symmetric difference
=>	(EQUAL/GREATER THAN)	Renames identifiers on import and/or export

Table III: Extended Pascal Special Symbol Tokens

In addition to the token words reserved for unextended Pascal, Extended Pascal requires support for several new token words (see Table IV below).

AND_THEN	Boolean operator w/short circuiting
BINDABLE	Bindable to some entity external to the program
EXPORT	Exporting to/from modules

IMPORT	Importing to/from modules
MODULE	Can be compiled, but not executed, by itself
ONLY	Selective import option
OR_ELSE	Boolean operator w/short circuiting
OTHERWISE	Default condition handler in CASE statement
POW	To the integer power of
PROTECTED	Protects exported variables from being altered by importors; Protects function/procedure parameter from being altered by function/procedure
QUALIFIED	Qualified import specification
RESTRICTED	Creation of opaque data types
VALUE	Specifies initial state for component

Table IV: Extended Pascal Word Symbol Tokens

The following, from ISO 10206, is a summary of features in Extended Pascal that are not found in unextended Pascal.

1. Modularity and Separate Compilation

Modularity provides for separately-compilable program components, while maintaining type security.

- 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.

2. Schemata

A schema determines a collection of similar types. Types may be

selected statically or dynamically from schemata.

- Statically selected types are used 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.

3. String Capabilities

The comprehensive string facilities unify fixed-length strings and character values with variable-length strings.

- All string and character values are compatible.
- The concatenation operator (+) combines all string and character values.
- Variable-length strings have programmer-specified maximum lengths.
- Strings may be compared using blank padding via the relational 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.

4. 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.

5. 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.

6. File Extend Procedure

The procedure extend prepares an existing file for writing at its end.

7. Constant Expressions

A constant expression may occur in any context needing a constant value.

8. Structured Value Constructors

An expression may represent the value of an array, record, or set in terms of its components. This is particularly valuable for defining structured constants.

9. Generalized Function Results

The result of a function may have any assignable type. A function result variable may be specified, which is especially useful for functions returning structures.

10. Initial Variable State

The initial state specifier of a type can specify the value with which variables are to be created.

11. Relaxation of Ordering of Declarations

There may be any number of declaration parts (labels, constants, types, variables, procedures, and functions) in any order. The prohibition of forward references in declarations is retained.

12. Type Inquiry

A variable or parameter may be declared to have the type of another parameter or another variable.

13. Implementation Characteristics

The constant maxchar is the largest value of type char. The constants minreal, maxreal, and epsreal describe the range of magnitude and the precision of real arithmetic.

14. Case-Statement and Variant Record Enhancements

Each case-constant-list may contain ranges of values. An otherwise clause represents all values not listed in the case-constant-lists.

15. Set Extension

- 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.

16. Date and Time

The procedure GetTimeStamp and the functions date and time, together with the related type TimeStamp, provide numeric representations of the current date and time and convert the numeric representations to strings.

17. Inverse Ord

A generalization of succ and pred provides an inverse ord capability.

18. Standard Numeric Input

The definition of acceptable character sequences read from a textfile includes all standard numeric representations defined by ISO 6093.

19. Nondecimal Representation of Numbers

Integer numeric constants may be expressed using bases two through thirty-six.

20. Underscore in Identifiers

The underscore character (_) may occur within identifiers and is significant to their spelling.

21. Zero Field Widths

The total field width and fraction digits expressions in write parameters may be zero.

22. Halt

The procedure halt causes termination of the program.

23. 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 +, , *, /, =, <> 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, arctan operate on complex values.

24. Short Circuit Boolean Evaluation

The operators and then and or else are logically equivalent to AND and OR, except that evaluation order is defined as left-to-right, and the right operand is not evaluated if the value of the expression can be determined solely from the value of the left operand.

25. Protected Parameters.

A parameter of a procedure or a function can be protected from modification within the procedure or function.

26. Exponentiation

The operators ** and pow provide exponentiation of integer, real, and complex numbers to real and integer powers.

27. Subrange Bounds

A general expression can be used to specify the value of either bound in a subrange.

28. Tag Fields of Dynamic Variables

Any tag field specified by a parameter to the procedure new is given the specified value.

Additionally, Extended Pascal incorporates the following feature at level 1 of this standard:

29. Conformant Arrays

Conformant arrays provide upward compatibility with level 1 of ISO 7185, Programming languages - PASCAL.

Object Pascal

There is no official Object Pascal standard, but a technical report was published in 1993 outlining the recommended standards for Object Pascal. A draft of this report is available on the web at http://pascal-central.com/ooe-stds.html. As I am not an avid user of Object-Oriented programming, only an outline of the draft report will be

included in this paper (Figure 7 below). Further information can be obtained at the above web page.

I. Introduction
II. Scope
III. References
IV. Definitions
V. Definitional Conventions
VI. Compliance
VII. Object Extensions
1. Class Definition
Extension of the Type System, Restrictions on Class Definitions, Contents and Syntax of Class Definitions
(Kinds, Inheritance, Fields, Methods, Constructors, Destructors)
Scope of Entities Defined in a Class, Class Definitions
2. Kinds of Classes
(Concrete, Abstract, Property, Type Model, Views)
3. Inheritance
(The Root Class, Multiple Inheritance, Name Conflicts, Overriding, Abstract Methods, Constructors, and Destructors)
4. Syntax
5. Object Access
The Object Model, Implicit Parameter Self
Polymorphism during Construction and Destruction
Implicit References, Field References, Inherited, Reference Type Coercion
Operations
(Compatibility, Methods Activation, Constructors & Destructors, Assignment, Comparison, Parameter Passing, Membership)
6. Predefined Entities
Null, Copy, Root
(Create, Destroy, Clone, Equal)
TextWritable
(ReadObj and WriteObj)
7. Signatures
8. With Statement
9. Procedure, Function, Constructor, and Destructor Declarations
10. Changes to Export Clause
11. Visibility
12. Extended Pascal Features
13. Suggested Changes to Extended Pascal

Figure 7: Object Pascal Report Outline

Return to Table of Contents Next Chapter

Copyright © 2001 Academic Press. All Rights Reserved.