STANDARD **10206**

First edition
1991-04-15

Information technology — Programming languages — Extended Pascal

Langages de programmation — Pascal étendu

Reference number
ISO/IEC 10206 : 1991 (E)

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 10206 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

Annexes A to G are for information only.

Introduction

This International Standard provides an unambiguous and machine independent definition of the programming language Extended Pascal. Its purpose is to facilitate portability of Extended Pascal programs for use on a wide variety of data processing systems.

Language history

The computer programming language Pascal was designed by Professor Niklaus Wirth to satisfy two principal aims

a) to make available a language suitable for teaching programming as a systematic discipline based on certain fundamental concepts clearly and naturally reflected by the language;

b) to define a language whose implementations could be reliable and efficient on then-available computers.

However, it has become apparent that Pascal has attributes that go far beyond those original goals. It is now being increasingly used commercially in the writing of system and application software. With this increased use, there has been an increased demand for and availability of extensions to ISO 7185:1983, *Programming languages - PASCAL*. Programs using such extensions attain the benefits of the extended features at the cost of portability with standard Pascal and with other processors supporting different sets of extensions. In the absence of a standard for an extended language, these processors have become increasingly incompatible. This International Standard is primarily a consequence of the growing commercial interest in Pascal and the need to promote the portability of Pascal programs between data processing systems.

Project history

In 1977, a working group was formed within the British Standards Institution (BSI) to produce a standard for the programming language Pascal. This group produced several working drafts, the first draft for public comment being widely published early in 1979. In 1978, BSI's proposal that Pascal be added to ISO's programme of work was accepted, and the ISO Pascal Working Group (then designated ISO/TC97/SC5/WG4) was formed in 1979. The Pascal standard was to be published by BSI on behalf of ISO, and this British Standard referenced by the International Standard.

In the USA, in the fall of 1978, application was made to the IEEE Standards Board by the IEEE Computer Society to authorize project 770 (Pascal). After approval, the first meeting was held in January 1979.

In December 1978, X3J9 convened as a result of a SPARC (Standards Planning and Requirements Committee) resolution to form a US TAG (Technical Advisory Group) for the ISO Pascal standardization effort initiated by the UK. These efforts were performed under X3 project 317.

In agreement with IEEE representatives, in February 1979, an X3 resolution combined the X3J9 and P770 committees into a single committee called the Joint X3J9/IEEE P770 Pascal Standards Committee. (Throughout, the term JPC refers to this committee.) The first meeting as JPC was held in April 1979.

The resolution to form JPC clarified the dual function of the single joint committee to produce a dpANS and a proposed IEEE Pascal standard, identical in content.

ANSI/IEEE770X3.97-1983, American National Standard Pascal Computer Programming Language, was approved by the IEEE Standards Board on September 17, 1981, and by the American National Standards Institute on December 16, 1982. British Standard BS6192, Specification for Computer programming language Pascal, was published in 1982, and International Standard 7185 (incorporating BS6192 by reference) was approved by ISO on December 1, 1983. Differences between the ANSI and ISO standards are detailed in the Foreword of ANSI/IEEE770X3.97-1983. (BS6192/ISO7185 was revised and corrected during 1988/89; it is expected that ANSI/IEEE770X3.97-1983 will be replaced by the revised ISO 7185.)

Following the decision that the first publication of a standard for the programming language Pascal would not contain extensions to the language, JPC prepared a project proposal to SPARC for an Extended Pascal Standard. When approved by X3 in November 1980, this proposal formed the charter for Project 345.

JPC immediately formed the Extension Task Group to receive all proposals for extensions to the Pascal language, developed the content of proposals so that they were in a form suitable for review by JPC, fairly and equitably reviewed all proposals in light of published JPC policy, and provided a liaison with the public in all matters concerning proposed extensions to the Pascal language.

X3 issued a press release on behalf of JPC in January 1980 to solicit extension proposals or suggestions from the general public. At this time, JPC had already prepared a list of priority extensions; public comment served to validate and supplement the priority list. Criteria for evaluating extensions were established and included machine independence, upward compatibility, conceptual integrity, rigorous definition, and existing practice as prime objectives. Extension proposals submitted by the public and by the JPC membership were developed and refined. JPC procedures guaranteed that proposals would be considered over at least two meetings, affording adequate time for review of the technical merits of each proposal.

By June of 1983, twelve extensions had been designated by JPC as candidate extensions and were published as a Candidate Extension Library. Ongoing work was described in Work in Progress, published with the Candidate Extension Library. This effort served as an interim milestone and an opportunity for the public to review the effort to date.

In 1984, BSI also started work on extensions to Pascal, with an initial aim of providing extensions in a few areas only. In 1985, the ISO Pascal Working Group (then designated ISO/TC97/SC22/WG2, now ISO/IEC JTC1/SC22/WG2) was reconvened after a long break to consider proposals from both ANSI and BSI in an international forum. Thereafter WG2 met at regular intervals to reconcile the national standardization activities in ANSI and BSI and to consider issues raised by the other experts participating in WG2.

The Work in Progress, along with other proposals subsequently received, continued its development until June 1986. The process of reconciling individual candidate extensions among themselves was begun in September 1984 and continued until June 1986. During this phase, conflicts between changes were resolved and each change was reconsidered. Working drafts of the full standard were circulated within JPC and WG2 to incorporate changes from each meeting.

The candidate extensions were then integrated into a draft standard that was issued for public review. The Public Comment Task Group (PCTG) was formed to respond to the public comments and recommend changes to the draft. To promote a unified response on each comment issue, PCTG included members from both WG2 and JPC. All responses and recommended changes required final approval by JPC and WG2. PCTG recommended several substantive changes that were subsequently approved as changes to the draft. These changes were incorporated and a new draft was produced for a second public review.

Project charter

The goal of JPC's Project 345 was to define an implementable, internationally acceptable Extended Pascal Standard.

This International Standard was to encompass those extensions found to be

a) compatible with ANSI/IEEE770X3.97-1983, American National Standard Programming Language Pascal, and

b) beneficial with respect to cost.

JPC's approved program of work included:

a) solicitation of proposals for extended language features;

b) the critical review of such proposals;

c) synthesis of those features found to be acceptable individually and which are mutually consistent into a working draft proposed standard;

d) interface with all interested standards bodies, both domestic and international;

e) submission of the working draft to ISO/TC97/SC22/WG2;

f) synthesis and submission of a draft proposed ANS consistent with any international standard developed;

g) review and correction of the dpANS in light of any comment received during Public Comment and/or Trial Use periods.

Technical development

Extended Pascal incorporates the features from ANSI/IEEE770X3.97-1983 and the following new features:

a) **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.

b) **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.

c) **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.

d) **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.

e) **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.

f) **File Extend Procedure.** The procedure **extend** prepares an existing file for writing at its end.

g) **Constant Expressions.** A constant expression may occur in any context needing a constant value.

h) **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.

i) **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.

j) **Initial Variable State.** The initial state specifier of a type can specify the value with which variables are to be created.

k) **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.

l) **Type Inquiry**. A variable or parameter may be declared to have the type of another parameter or another variable.

m) **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.

n) **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.

o) **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.

p) **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.

q) **Inverse Ord**. A generalization of **succ** and **pred** provides an inverse ord capability.

r) **Standard Numeric Input**. The definition of acceptable character sequences read from a textfile includes all standard numeric representations defined by ISO 6093.

s) **Nondecimal Representation of Numbers**. Integer numeric constants may be expressed using bases two through thirty-six.

t) **Underscore in Identifiers**. The underscore character (_) may occur within identifiers and *is significant* to their spelling.

u) **Zero Field Widths**. The total field width and fraction digits expressions in write parameters may be zero.

v) **Halt**. The procedure **halt** causes termination of the program.

w) **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.

x) **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.

y) **Protected Parameters**. A parameter of a procedure or a function can be protected from modification within the procedure or function.

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

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

B) **Tag Fields of Dynamic Variables.** Any tag field specified by a parameter to the procedure **new** is given the specified value.

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

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

Technical reports

During the development of this International Standard, various proposals were considered but not incorporated due to consideration of time and other factors. Selected proposals may be published as Technical Reports.

Information technology — Programming languages – Extended Pascal

1 Scope

1.1

This International Standard specifies the semantics and syntax of the computer programming language Extended Pascal by specifying requirements for a processor and for a conforming program. Two levels of compliance are defined for both processors and programs.

1.2

This International Standard does not specify

a) the size or complexity of a program and its data that will exceed the capacity of any specific data processing system or the capacity of a particular processor, nor the actions to be taken when the corresponding limits are exceeded;

b) the minimal requirements of a data processing system that is capable of supporting an implementation of a processor for Extended Pascal;

c) the method of activating the program-block or the set of commands used to control the environment in which an Extended Pascal program is transformed and executed;

d) the mechanism by which programs written in Extended Pascal are transformed for use by a data processing system;

e) the method for reporting errors or warnings;

f) the typographical representation of a program published for human reading.

2 Normative reference

The following standard contains provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the edition indicated was valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent edition of the standard listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO 646:1983, *Information processing — ISO 7-bit coded character set for information interchange.*

3 Definitions

For the purposes of this International Standard, the following definitions apply.

NOTE — To draw attention to language concepts, some terms are printed in italics on their first mention or at their defining occurrence(s) in this International Standard.