ISO DTR 10176 (JTC1/SC22/WG5 N010) RIN WG14/N123 X3J11/90-063

5.1.3.1 Guideline: Character sets used for program text

As far as possible, the language should be defined in terms only of the characters included within ISO 646, avoiding the use of any that are in national use positions. If any symbols are used which are not included within ISO 646 or are in national use positions, an alternative representation for all such symbols should be specified. A conforming processor should be required to be capable of accepting a program represented using only this minimal character set. Great care should be taken in specifying how "non-printing" characters are to be handled, i.e. those characters that correspond to integer values 0 to 32 inclusive and 127, i.e. *null* (0/0) to *space* (2/0) and *delete* (7/13).

NOTES

1 The motivation here is to provide a common basis for representing programs, which does not exist with current (published up to 1986) standards. The characters that are available in all national variants of ISO 646 cannot represent programs in many programming languages in a way that is acceptable to programmers who are familiar with the U.S national variant (usually referred to by its acronym "ASCII"). In particular, square brackets, curly brackets and vertical line are unavailable.

Further, the characters that are available in the International Reference Version of ISO 646 cannot represent programs in many programming languages in a way that is acceptable to programmers who are familiar with a particular national variant of ISO 646. For example, neither the pound nor dollar symbol may be available. The characters that are available in ASCII cannot represent programs in many programming languages in a way that is acceptable to programmers because their terminals support some other national variant of ISO 646.

Consideration needs also to be given to the use of upper and lower case (roman) letters. If only one case is required it should be made clear whether the other case is regarded as an alternative representation (so that, for example, *TIME*, *time*, *Time*, *tImE* are regarded as identical elements) or its use is disallowed in a standard-conforming program. Where both cases are required or allowed, the rules governing their use should be as simple as possible, and exactly and completely specified.

Of the non-printing characters, nearly all languages allow *space* (2/0), and *carriage return* (0/13) *line feed* (0/10) as a pair, though they differ as to whether these characters are meaningful or ignored. How *carriage return* without *line feed* (or vice versa) is to be treated needs consideration, as do constructions such as *carriage return*, *carriage return*, *line feed*. If characters are disallowed that do not show themselves on a printed representation, the undesirable situation may arise where a program may be incorrect though its printout shows no fault. If a tabulation character (0/9) is disallowed, this can cause trouble, since it appears to be merely a sequence of spaces; if allowed, the effect on languages such as Fortran,

168

having a given length of line, has to be considered.

2 The characters that are available in the eight-bit character sets ISO 4873 with DIS 8859 or ISO 6937/2, would be sufficient to represent programs in a way that looks familiar to most (but not APL) programmers. However, in 1989 these standards are not yet widely supported on printers and display terminals.

3 For advice on character set matters, committees should consult ISO/TC97/SC2.

5.1.3.2 Guideline: Character sets used in character literals

Character literals permitted to be embedded in program text in a standard-conforming program should be defined in such a way that each character may be represented using one or more of the following methods:

a) The character represents itself, eg *A*, *B*, *g*, *3*, *+*, *(*.

b) A character is represented by a pair of characters: an escape character followed by a graphic character, e.g. if *&* is the escape character, *&'* to represent apostrophe, *&&* to represent ampersand, *&n* to represent newline.

b) A character is represented by three characters: an escape character followed by two hexadecimal digits that specify its internal value.

Any conforming processor should be required to be able to accept "as themselves" (i.e. as in (1)) at least all printable characters in the "minimal set" defined in 5.1.3.1, apart possibly from any special-purpose characters such as an escape character or those used to delimit literal character strings.

NOTES

1 For reasons of portability it is necessary to provide a common basis for representing character literals in programs, in addition to the characters used for the program text itself. The required character set could be wider than (and for general purpose text handling would need to be wider than) that which is necessary for representation of program statements. Programs must be representable on as many different peripherals and systems as possible; there is therefore a need to reduce the number of characters required to represent a program to the minimum that is consistent with general practice and readability. On the other hand, programs themselves must be able to represent and process as many different characters as possible.

These two requirements make it impossible to represent every character by itself in a literal character string, if the language is to be suitable for general processing of character data.

2 The preceding paragraphs envisage that inside a program each character is storable as a single octet (8-bit byte). See 5.1.3.6 for discussion of handling of multi-octet characters.

3 A particular problem arises with the representation of a space in a character or string literal. It can be represented by a visible graphic character, the argument in favour being that blank spaces in program text should not affect the meaning. However, it can also be represented by itself, the argument in favour being that this is the most natural form of

representation. The indistinguishability of a tabulation character from a sequence of spaces (in a printed representation) is a particular problem since a function that returns the length of a string, in characters, may give different results from two programs that appear identical. There can be further complications when using a "high quality" printer with variable-width characters. Drafting committees are recommended to pay particular attention to these points.

4 The character set in ISO 6937/2 represents some graphic characters as a pair of octets. This is suitable for printing but is difficult to process in operations such as comparison and sorting.

5.1.3.3 Guideline: Character sets used in comments

The standard should define the characters that are permitted in comments in a standard-conforming program.

NOTE - For publication in the pages of a journal, some languages make no restriction on permitted characters in comments, beyond making it clear where the comment finishes. For inclusion on a computer file, however, it is preferable to restrict the characters to those that are widely available, to help portability. Since comments are intended for human reading and hence escape mechanisms are unnecessary, there is no disadvantage in printing characters simply representing themselves (apart of course from any characters or sequences of characters marking the end of the comment), and in limiting non-printing characters to those (like carriage return and line feed) necessary for layout purposes.

5.1.3.4 Guideline: Character sets used for data

The programming language standard should be defined in such a way that it is not assumed that character data processed by a program is anything other than a sequence of octets whose meaning depends on the context. However, a conforming processor should be required at least to be able to input, manipulate and output characters from the minimal character set defined in 5.1.3.1 above.

NOTES

1 The objective here is to provide a common basis for processing data. Many programs will assume that their data is expressed in ASCII or some other variant of ISO 646. But if the standard assumes that *all* data is expressed in any one particular character set, it will cause difficulties for many users.

2 See also the guideline on collating sequences (5.1.3.5 below).

5.1.3.5 Guideline: Collating sequences

The standard should specify completely the default collating sequence to be provided by a conforming processor, and preferably that this should be that implied by the ordering of the characters in the minimal character set drawn from ISO 646 as defined in 5.1.3.1 above. If the default collating sequence is other than that implied by ISO 646, means should be provided whereby the user may optionally

switch to the ISO 646 collating sequences, and consideration should be given to providing means for the user optionally to switch to alternative collating sequences, whether or not the defined default collating sequence is that based on ISO 646.

NOTES

1 Programs which perform ordering of character data are in general not portable unless the collating sequence is completely defined. This guideline ensures that such programs will be portable at least where only those characters drawn from the minimal character set defined in 5.1.3.1 are used.

2 Drafting committees may wish to consider further guidance relating to characters not included in the minimal character set, especially where ordering of character data is a major anticipated use of the language.

3 Possible means of including alternative collating sequences are language features or processor options (see 5.1.9).

4 Possible reasons for wishing to provide such alternative means are to obtain maximum processing efficiency by use of a processor-defined internal character set, or to allow orderings more useful for particular purposes, e.g. $a=A < b=B < \ldots < z=Z$. (ISO 646 implies $0 < 1 < \ldots < 9 < A < B \ldots < Z < a < b \ldots < z$, which is not always convenient.)

5.1.3.6 Guideline: Use of other character sets

The standard should ensure that it is possible within the language to support the handling of characters from a wide range of character sets, including multi-octet character sets and non-English single-octet character sets.

NOTES

1 For some applications, and for some classes of users for all applications, it is vital for the language to have the ability to accept and manipulate data from character sets other than the minimal character set needed for the basic purpose of specifying programs. For some users this need will be greater than the need for international interchange. An important task for any language standards committee is to ensure that it is possible for each of these needs to be met in a standard-conforming way.

2 The applications will require both the ability to manipulate multi-octet characters and the capability of international interchange. This may imply two or more alternative representations of the same "character" (data object), one of which will be a representation (for interchange purposes) in the minimal character set defined in 5.1.3.1.

3 In general it should be possible to use multi-octet or non-English single-octet character sets in program text, character literals, comment, and data without recourse to the use of processors which are not standard-conforming. Programs using such characters in program text, literals or comments may not be standard-conforming and in general will be less portable internationally than those using only the minimal character set, but may still be portable within the applications community for those programs. Defined mappings from other character sets to the minimal character set of the language, and the presence of suitable processor options, are likely to maximize benefits and usability for differing requirements.

4 At the time that this Technical Report is published, consideration of the issues addressed by this guideline is being actively pursued. Language standards committees are strongly urged to ascertain the latest developments in this area before specifying requirements.

5.1.4 Guideline: Error detection requirements

Requirements should be included covering error detection, reporting and handling, with appropriate conformity clauses. The standard should specify a minimum set of errors which a conforming processor must detect (in the absence of any masking errors); minimum level of accuracy and readability of error reports; whether an error is fatal or non-fatal; and, for non-fatal errors, the minimum recovery action to be taken.

NOTES

1 The objective of this guideline is to enhance the value of standards to users. The inclusion of requirements on error detection, reporting and handling provides a minimum level of assurance to the programmer of assistance from the processor in identifying errors.

2 See 4.1.3 for a definition of the term "error" in this context.

3 That an error is statically determinable (see 4.1.3) does not imply that the processor must necessarily determine it statically rather than dynamically.

4 It is recognized that requiring provision of specific error detection requirements within the standard entails a certain overhead in a conforming processor. It is a matter for each standards committee to determine how severely such overhead will affect the users of the language concerned, and consequently whether requiring detection is worthwhile. It is of course open to the committee to specify or recommend the provision of processor options which would permit the user to control the use of error detection (see 5.1.9).

5.1.4.1 Checklist of potential errors

The following is a list of typical errors which can arise in the submission of program text to a processor. Drafting committees should check all of the following for relevance to their language, and the standard produced should address all that are appropriate, plus others specific to the language concerned. This list is not to be considered either as exhaustive or as prescriptive.

In all cases the standard should specify whether the error concerned is fatal or non-fatal. Depending on the design and philosophy of the language, it may occur that a particular usage is not invalid (whereas it would be in another language) but that users would nevertheless benefit from the availability of a warning message within the processor.

5.1.4.1.1 Errors of program structure

[a] unmatched brackets - either open without close, or vice versa. Note: this covers all sorts of bracket: (), [], {} etc.