TC 97

International Standard ISO 6373

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION•МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ•ORGANISATION INTERNATIONALE DE NORMALISATION

Data processing — Programming languages — Minimal BASIC

Traitement de l'information — Langages de programmation — BASIC minimal

First edition — 1984-03-15

UDC 681.3.06 : 800.92

Ref. No. ISO 6373-1984 (E)

Descriptors : data processing, programming languages, information interchange.

Price based on 33 pages

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of developing International Standards is carried out through ISO technical committees. Every member body interested in a subject for which a technical committee has been authorized has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work.

Draft International Standards adopted by the technical committees are circulated to the member bodies for approval before their acceptance as International Standards by the ISO Council.

International Standard ISO 6373 was developed by Technical Committee ISO/TC 97, *Information processing systems*, and was circulated to the member bodies in April 1982.

It has been approved by the member bodies of the following countries :

Belgium	Germany, F. R.	Romania
Canada	Hungary	South Africa, Rep. of
China	Ireland	Spain
Czechoslovakia	Italy	Sweden
Egypt, Arab Rep. of	Japan	Switzerland
Finland	Netherlands	USA
France	Poland	Yugoslavia

The member bodies of the following countries expressed disapproval of the document on technical grounds :

Australia
United Kingdom

Contents

Data processing — Programming languages — Minimal BASIC

1 Scope and field of application

This International Standard is designed to promote the interchangeability of BASIC programs among a variety of automatic data processing systems.

This International Standard establishes :

a) the syntax of a program written in Minimal BASIC;

b) the formats of data and the precision and range of numeric representations which are acceptable as input to an automatic data processing system being controlled by a program written in Minimal BASIC;

c) the formats of data and the precision and range of numeric representations which can be generated as output by an automatic data processing system being controlled by a program written in Minimal BASIC;

d) the semantic rules for interpreting the meaning of a program written in Minimal BASIC;

e) the errors and exceptional circumstances which shall be detected and also the manner in which such errors and exceptional circumstances shall be handled.

Subsequent International Standards for the same purpose will describe extensions and enhancements to this International Standard. Programs conforming to this International Standard, as opposed to extensions or enhancements of this International Standard, will be said to be written in Minimal BASIC.

Although the BASIC language was originally designed primarily for interactive use, this International Standard describes a language that is not so restricted.

The organization of the International Standard is outlined in annex A. The method of syntax specification used is explained in annex B.

2 Conformance

There are two aspects of conformance to this International Standard :

a) conformance by a program written in the language;

b) conformance by an implementation which processes such programs.

A program conforms to this International Standard only when

— each statement contained therein is a syntactically valid instance of a statement specified in this International Standard;

— each statement has an explicitly valid meaning specified herein;

— the totality of statements compose an instance of a valid program which has an explicitly valid meaning specified herein.

An implementation conforms to this International Standard only when

— it accepts and processes programs conforming to this International Standard;

— it reports reasons for rejecting any program which does not conform to this International Standard;

— it interprets errors and exceptional circumstances according to the specifications of this International Standard;

— its interpretation of the semantics of each statement of a standard-conforming program conforms to the specifications in this International Standard;

— its interpretation of the semantics of a standard-conforming program as a whole conforms to the specifications in this International Standard;

— it accepts as input, manipulates, and can generate as output numbers of at least the precision and range specified in this International Standard;

— it is accompanied by a reference manual which clearly defines the actions taken in regard to features which are called "undefined" or "implementation-defined" in this International Standard.

This International Standard does not include requirements for reporting specific syntactic errors in the text of a program.

Implementations conforming to this International Standard may accept programs written in an enhanced language without having to report all constructs not conforming to this International Standard. However, whenever a statement or other program element does not conform to the syntactic rules given herein, either an error shall be reported or the statement or other program element shall have an implementation-defined meaning.

An exception occurs when an implementation recognizes that a program may not perform or is not performing in accordance with this International Standard. All exceptions described in this International Standard shall be reported to the user unless some mechanism provided in an enhancement to this International Standard has been invoked by the user to handle exceptions.

Where indicated, certain exceptions may be handled by the procedures specified in this International Standard; if no procedure is given, or if restrictions imposed by the hardware or the operating environment make it impossible to follow the given procedures, then the exception shall be handled by terminating the program. Enhancements to this International Standard may describe mechanisms for controlling the manner in which exceptions are reported and handled, but no such mechanisms are specified in this International Standard.

This International Standard does not specify an order in which exceptions shall be detected or processed.

3 References

ISO 646, *Information processing — 7-bit coded character set for information interchange.*[1]

ISO 4873, *Information processing — 8-bit coded character set for information interchange.*

ISO 6093, *Information processing — Specification for representation of numeric values in character strings for information interchange.*[2]

4 Definitions

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

4.1 BASIC : A term applied as a name to members of a special class of languages which possess similar syntaxes and semantic meanings; acronym for Beginner's All-purpose Symbolic Instruction Code.

4.2 batch-mode : The processing of programs in an environment where no provision is made for user interaction.

4.3 can : The word "can" is used in a descriptive sense to indicate that standard-conforming programs are allowed to contain certain constructions and that standard-conforming implementations are required to process such programs correctly.

4.4 end-of-line : The character(s) or indicator which identifies the termination of a line. Lines of three kinds may be identified in Minimal BASIC : program *lines*, print lines, and *input-reply* lines. *End-of-lines* may vary between the three cases and may also vary depending upon context. Thus, for example, the *end-of-line* in an *input-reply* may vary on a given system depending on the terminal being used in interactive or batch mode.

Typical examples of *end-of-line* are carriage-return, carriage-return line-feed, and end of record (such as end of card).

4.5 error : A flaw in the syntax of a *line* which causes it not to be part of a standard program.

4.6 exception : A circumstance arising in the course of executing a *program* which results from faulty data or computations or from exceeding some resource constraint. Where indicated certain exceptions (non-fatal exceptions) may be handled by the specified procedures; if no procedure is given (fatal exceptions) or if restrictions imposed by the hardware or operating environment make it impossible to follow the given procedure, then the exception shall be handled by terminating the *program*.

4.7 identifier : A character string used to name a *variable* or a function.

4.8 interactive mode : The processing of programs in an environment which permits the user to respond directly to the actions of individual programs and to control the commencement and termination of these programs.

4.9 keyword : A character string, usually with the spelling of a commonly used or mnemonic word, which provides a distinctive identification of a statement or a component of a statement of a programming language.

The keywords in Minimal BASIC are : BASE, DATA, DEF, DIM, END, FOR, GO, GOSUB, GOTO, IF, INPUT, LET, NEXT, ON, OPTION, PRINT, RANDOMIZE, READ, REM, RESTORE, RETURN, STEP, STOP, SUB, THEN, and TO.

4.10 line : A single transmission of characters which terminates with an *end-of-line*.

4.11 machine infinitesimal : The smallest positive value (other than zero) which can be represented and manipulated by a BASIC implementation.

4.12 machine infinity : The positive and negative values of greatest magnitude which can be represented and manipulated by a BASIC implementation. It is not required that manipulations of machine infinity yield non infinite results.

4.13 may : The word "may" is used in a permissive sense to indicate that a standard-conforming implementation may or may not provide a particular feature.

4.14 overflow : With respect to numeric operations, the condition which exists when a prior operation has attempted to generate a result which exceeds machine infinity.

1) At present at the stage of draft. (Revision of ISO 646-1973.)

2) At present at the stage of draft.

With respect to string operations, the condition which exists when a prior operation attempts to generate a result which has more characters than can be contained in a string of maximal length, as determined by the language processor.

4.15 print zone: A contiguous set of character positions in a printed output line which may contain an evaluated *print-statement* element.

4.16 rounding: The process by which a representation of a value with lower precision is generated from a representation of higher precision taking into account the value of that portion of the original number which is to be omitted. For example, rounding X to the nearest integer may be accomplished by INT(X + 0,5).

4.17 shall: The word "shall" is used in an imperative sense to indicate that a program is required to be constructed, or that an implementation is required to act, as specified in order to meet the constraints of standard conformance.

4.18 significant digits: The contiguous sequence of digits between the high-order non-zero digit and the low-order non-zero digit, without regard for the location of the radix point. Commonly, in a normalized floating point internal representation, only the significant digits of a representation are maintained in the significand.

4.19 truncation: The process by which a representation of a value with lower precision is generated from a representation of higher precision by merely deleting the unwanted low order digits of the original representation.

4.20 underflow: The condition which exists when a prior operation has attempted to generate a result, other than zero, which is less in magnitude than machine infinitesimal. This term is applied only to numeric operations.

5 Characters and strings

5.1 General description

The character set for BASIC is contained in the International Reference Version of ISO 646. Strings are sequences of characters and are used in BASIC *programs* as comments (see clause 20), as *string-constants* (see clause 7), or as *data* (see clause 16).

5.2 Syntax

The syntax shall be defined as follows:

1)	*letter*	= A/B/C/D/E/F/G/H/I/J/K/L/M/N/O/P/Q/R/S/T/U/V/W/X/Y/Z
2)	*digit*	= 0/1/2/3/4/5/6/7/8/9
3)	*string-character*	= *quotation-mark/quoted-string-character*
4)	*quoted-string-character*	= *exclamation-mark/number-sign/dollar-sign/percent-sign/ ampersand/apostrophe/left-parenthesis/right-parenthesis/ asterisk/comma/solidus/colon/semicolon/less-than-sign/equals-sign/ greater-than-sign/question-mark/circumflex-accent/underline/unquoted-string-character*
5)	*unquoted-string-character*	= *space/plain-string-character*
6)	*plain-string-character*	= *plus-sign/minus-sign/full-stop/digit/letter*
7)	*remark-string*	= *string-character**
8)	*quoted-string*	= *quotation-mark/quoted-string-character*/quotation-mark*
9)	*unquoted-string*	= *plain-string-character/(plain-string character unquoted-string-character* plain-string-character)*

5.3 Examples

The following examples are taken from 5.2:

7) ANY CHARACTERS AT ALL (?!*!!) CAN BE USED IN A "REMARK".

8) "SPACES, AND COMMAS, CAN OCCUR IN QUOTED STRINGS."

9) COMMAS CANNOT OCCUR IN UNQUOTED STRINGS.

5.4 Semantics

The *letters* shall be the set of uppercase roman letters contained in the ISO 7-bit coded character set in positions 4/1 to 5/10.

The *digits* shall be the set of arabic digits contained in the ISO 7-bit coded character set in positions 3/0 to 3/9.

The remaining *string-characters* shall correspond to the remaining graphic characters in positions 2/0 to 2/15, 3/10 to 3/15, 5/14 and 5/15 of the ISO 7-bit coded character set.

The names of characters are specified in table 1.

The coding of characters is specified in table 2; however, this coding applies only when *programs* and/or input/output data are exchanged by means of coded media.

5.5 Exceptions

None.

5.6 Remarks

Other characters from the ISO 7-bit coded character set (including control characters) may be accepted by an implementation and may have a meaning for some other processor (such as an editor), but have no prescribed meaning within this International Standard. Programs containing characters other than the *string-characters* described in 5.4 and 5.2 and *end-of-line* characters are not standard-conforming *programs*.

The various kinds of characters and strings described by the syntax correspond to the various uses of strings in a BASIC *program*. *Remark-strings* may be used in *remark-statements* (see clause 20). *Quoted-strings* may be used as *string-constants* (see clause 7). *Unquoted-strings* may be used in addition to *quoted-strings* as *data* elements (see clause 18) without being enclosed in *quotation-marks*; *unquoted-strings* cannot contain leading or trailing *spaces*.

6 Programs

6.1 General description

BASIC is a line-oriented language. A BASIC *program* is a sequence of *lines*, the last of which shall be an *end-line* and each of which shall contain a keyword. Each *line* shall contain a unique *line-number* which serves as a label for the *statement* contained in that *line*.

6.2 Syntax

The syntax shall be defined as follows:

1)	*program*	= *block* end-line*
2)	*block*	= *statement-line/for-block*
3)	*statement-line*	= *line-number statement end-of-line*

4)	*line-number*	=	*digit digit? digit? digit?*
5)	*end-of-line*	=	*[implementation-defined]*
6)	*end-line*	=	*line-number end-statement end-of-line*
7)	*end-statement*	=	END
8)	*statement*	=	*data-statement/def-statement/dimension-statement/gosub-statement/goto-statement/ if-then-statement/input-statement/let-statement/on-goto-statement/option-statement/ print-statement/randomize-statement/read-statement/remark-statement/restore-statement/ return-statement/stop-statement*
9)	*line*	=	*statement-line/end-line/for-line/next-line*

6.3 Examples

The following example is taken from 6.2:

6) 999 END

6.4 Semantics

A BASIC *program* shall be composed of a sequence of *lines* ordered by *line-numbers*, the last *line* of which shall be an *end-line*. Program *lines* shall be executed in sequential order, starting with the first *line*, until

— some other action is dictated by execution of a control *statement* or *for-block*, or

— a fatal exception occurs, or

— a *stop-statement* or *end-statement* is executed.

The syntax as described generates *programs* which contain no *spaces* other than those occurring in *remark-statements*, in certain *quoted-strings* and *unquoted-strings*, or where the presence of a *space* is explicitly indicated by the metaname *space*.

Special conventions shall be observed regarding *spaces*. With the following exceptions, *spaces* may occur anywhere in a BASIC *program* without affecting the execution of that *program* and may be used to improve the appearance and readability of the *program*.

Spaces shall not appear

a) at the beginning of a *line*;

b) within keywords;

c) within the word TAB in a *tab-call*;

d) within *numeric-constants*;

e) within *line-numbers*;

f) within function or *variable* names;

g) within multicharacter *relation* symbols.

In addition, *spaces* which appear in *quoted-strings* and *unquoted-strings* are significant.

All keywords in a *program* shall be preceded by at least one *space* and, if not at the end of a *line*, shall be followed by at least one *space*.

Each *line* shall begin with a *line-number*. The values of the integers represented by the *line-numbers* shall be positive and non-zero; leading zeroes shall have no effect. *Statements* shall occur in ascending *line-number* order.

The manner in which the end of a statement *line* is detected is determined by the implementation; for example, the *end-of-line* may be a carriage-return character, a carriage-return character followed by a line-feed character, or the end of a physical record.

Lines in a standard-conforming *program* may contain up to 72 characters; the *end-of-line* indicator is not included within this 72 character limit.

The *end-statement* serves both to mark the physical end of the main body of a *program* and to terminate the execution of the *program* when encountered.

6.5 Exceptions

None.

6.6 Remarks

Local editing facilities may allow for the entry of statement *lines* in any order and also allow for duplicate *line-numbers* and lines containing only a *line-number*. Such editing facilities usually sort the *program* into the proper order; in the case of duplicate *line-numbers*, the last *line* entered with that *line-number* is retained. In many implementations, a line containing only a *line-number* (without trailing *spaces*) is deleted from the *program*.

7 Constants

7.1 General description

Constants can denote both scalar numeric values and string values.

A *numeric-constant* is a decimal representation in positional notation of a number. There are four general syntactic forms of *numeric-constants*:

a) implicit point representation:	sd ... d
b) explicit point unscaled representation:	sd ... drd ... d
c) explicit point scaled representation:	sd ... drd ... dEsd ... d
d) implicit point scaled representation:	sd ... dEsd ... d

where

d is a deciumal *digit*;

r is a *full-stop*;

s is an optional *sign*;

E is the explicit character E.

A *string-constant* is a character string enclosed in *quotation-marks* (see clause 5).

7.2 Syntax

The syntax shall be defined as follows:

1)	*numeric-constant*	= *sign? numeric-rep*
2)	*sign*	= *plus-sign/minus-sign*
3)	*numeric-rep*	= *significand exrad?*
4)	*significand*	= *(integer full-stop?)/(integer? fraction)*
5)	*integer*	= *digit digit**
6)	*fraction*	= *full-stop digit digit**
7)	*exrad*	= *E sign? integer*
8)	*string-constant*	= *quoted-string*

7.3 Examples

The following examples are taken from 7.2:

1) −21.

3) 1E10
   5E-1
   .4E + 1

4) 500
   1

6) .255

8) "XYZ"
   "X − 3B2"
   "1E10"

7.4 Semantics

The value of a *numeric-constant* is the number represented by that constant. "E" stands for "times ten to the power"; if no *sign* follows the symbol *E*, then a *plus-sign* is understood. *Spaces* shall not occur in *numeric-constants*.

A *program* may contain numeric representations which have an arbitrary number of *digits*, though implementations may round the values of such representations to an implementation-defined precision of not less than six significant decimal digits.

Numeric-constants may also have an arbitrary number of *digits* in the *exrad*, though non-zero constants whose magnitude is outside an implementation-defined range may be treated as exceptions. It is recommended that the implementation-defined range for *numeric-constants* be approximately 1E − 38 to 1E + 38 or larger. Constants whose magnitudes are less than machine infinitesimal shall be replaced by zero, while constants whose magnitudes are larger than machine infinity shall be reported as causing an overflow.

A *string-constant* has as its value the string of all characters between the *quotation-marks*; *spaces* shall not be ignored. The length of a *string-constant*, i.e. the number of characters contained between the *quotation-marks*, is limited only by the length of a *line*.

7.5 Exceptions

The evaluation of a *numeric-constant* causes an overflow (non-fatal; the recommended recovery procedure is to supply machine infinity with the appropriate sign, report it and continue).

7.6 Remarks

Since this International Standard does not require that strings with more than 18 characters be assignable to *string-variables* (see clause 8), conforming *programs* can use *string-constants* with more than 18 characters only as elements in a *print-list*.

It is recommended that implementations report constants whose magnitudes are less than machine infinitesimal as underflows and continue.

8 Variables

8.1 General description

Variables in BASIC are associated with either numeric or string values and, in the case of *numeric-variables*, may be either simple *variables* or references to elements of one or two dimensional arrays; such references are called subscripted *variables*.

Simple-numeric-variables shall be named by a *letter* followed by an optional digit.

Subscripted *numeric-variables* shall be named by a *letter* followed by one or two *numeric-expressions* enclosed within parentheses.

String-variables shall be named by a letter followed by a *dollar sign*.

explicit declarations of variable types are not required; a *dollar-sign* serves to distinguish *string-variables* from *numeric-variables*, and the presence of a *subscript* distinguishes a subscripted *variable* from a simple one.

8.2 Syntax

The syntax shall be defined as follows:

1)	*variable*	= *numeric-variable / string-variable*
2)	*numeric-variable*	= *simple-numeric-variable / numeric-array-element*
3)	*simple-numeric-variable*	= *letter digit?*
4)	*numeric-array-element*	= *numeric-array-name subscript*
5)	*numeric-array-name*	= *letter*
6)	*subscript*	= *left-parenthesis numeric-expression (comma numeric-expression)? right parenthesis*
7)	*string-variable*	= *letter dollar-sign*

8.3 Examples

The following examples are taken from 8.2:

```
3)  X
    A5
4)  V(3)
    W(X,X + Y/2)
7)  S$
```

8.4 Semantics

At any instant in the execution of a *program*, a *numeric-variable* is associated with a single numeric value and a *string-variable* is associated with a single string value. The value associated with a *variable* may be changed by the execution of *statements* in the *program*.

The length of the character string associated with a *string-variable* can vary during the execution of a *program* from a length of zero characters (signifying the null or empty string) to 18 characters.

Simple-numeric-variables and *string-variables* are declared implicitly through their appearance in the *program*.

A subscripted *variable* refers to the element in the one or two-dimensional array selected by the value(s) of the subscript(s). The value of each subscript is rounded (see 4.16) to the nearest integer. Unless explicitly declared in a *dimension-statement*, subscripted *variables* are implicitly declared by their first appearance in a *program*. In this case, the range of each subscript is from zero to ten inclusive, unless the presence of an *option-statement* indicates that the range is from one to ten inclusive. *Subscript expressions* shall have values within the appropriate range (see clause 19).

The same *letter* shall not be the name of both a simple *variable* and an array, nor the name of both a one-dimensional and a two-dimensional array.

There is no relationship between a *numeric-variable* and a *string-variable* whose names agree except for the *dollar-sign*.

At the initiation of execution the values associated with all *variables* shall be implementation-defined.

8.5 Exceptions

An integer obtained as the value of a subscript expression is not in the range of the explicit or implicit dimensioning bounds (fatal).

8.6 Remarks

Since initialization of *variables* is not specified, and hence may vary from implementation to implementation, *programs* that are intended to be transportable should explicitly assign a value to each *variable* before any *expression* involving that *variable* is evaluated.

There are many commonly used alternatives for associating implementation-defined initial values with *variables*; it is recommended that all *variables* are recognizably undefined in the sense that an exception will result from any attempt to access the values of any *variable* before that *variable* is explicitly assigned a value.