Table of Contents

3. Usage requirements

A system shall provide all of the words defined in 6.1 Core Words. It may also provide any words defined in the optional word sets and extensions word sets. No standard word provided by a system shall alter the system state in a way that changes the effect of execution of any other standard word except as provided in this Standard. A system may contain non-standard extensions, provided that they are consistent with the requirements of this Standard.

The implementation of a system may use words and techniques outside the scope of this Standard.

A system need not provide all words in executable form. The implementation may provide definitions, including definitions of words in the Core word set, in source form only. If so, the mechanism for adding the definitions to the dictionary is implementation defined.

A program that requires a system to provide words or techniques not defined in this Standard has an environmental dependency.

See: A.3 Usage requirements

3.1 Data types

A data type identifies the set of permissible values for a data object. It is not a property of a particular storage location or position on a stack. Moving a data object shall not affect its type.

No data-type checking is required of a system. An ambiguous condition exists if an incorrectly typed data object is encountered.

Table 3.1 summarizes the data types used throughout this Standard. Multiple instances of the same type in the description of a definition are suffixed with a sequence digit subscript to distinguish them.

See: A.3.1 Date-types

3.1.1 Data-type relationships

Some of the data types are subtypes of other data types. A data type i is a subtype of type j if and only if the members of i are a subset of the members of j. The following list represents the subtype relationships using the phrase **i => j** to denote **i is a subtype of j**. The subtype relationship is transitive; if i => j and j => k then i => k:

```
+n => u => x;
+n => n => x;
char => +n;
a-addr => c-addr => addr => u;
flag => x;
xt => x;
+d => d => xd;
+d => ud => xd.
```

Any Forth definition that accepts an argument of type i shall also accept an argument that is a subtype of i.

3.1.2 Character types

Characters shall be at least one address unit wide, contain at least eight bits, and have a size less than or equal to cell size.

The characters provided by a system shall include the graphic characters {32..126}, which represent graphic forms as shown in table 3.2.

See: A.3.1.2 Character types

3.1.2.1 Graphic characters

A graphic character is one that is normally displayed (e.g., A, #, &, 6). These values and graphics, shown in table 3.2, are taken directly from ANS X3.4-1974 (ASCII) and ISO 646-1983, International Reference Version (IRV). The graphic forms of characters outside the hex range {20..7E} are implementation-defined. Programs that use the graphic hex 24 (the currency sign) have an environmental dependency.

The graphic representation of characters is not restricted to particular type fonts or styles. The graphics here are examples.

3.1.2.2 Control characters

All non-graphic characters included in the implementation-defined character set are defined in this Standard as control characters. In particular, the characters {0..31}, which could be included in the implementation-defined character set, are control characters.

Programs that require the ability to send or receive control characters have an environmental dependency.

Table 3.1 - Data types

Symbol	Data type	Size on stack
flag	flag	1 cell
true	true flag	1 cell
false	false flag	1 cell
char	character	1 cell
n	signed number	1 cell
+n	non-negative number	1 cell
u	unsigned number	1 cell
n|u 1	number	1 cell
x	unspecified cell	1 cell
xt	execution token	1 cell
addr	address	1 cell
a-addr	aligned address	1 cell
c-addr	character-aligned address	1 cell
d	double-cell signed number	2 cells
+d	double-cell non-negative number	2 cells
ud	double-cell unsigned number	2 cells
d|ud 2	double-cell number	2 cells
xd	unspecified cell pair	2 cells
colon-sys	definition compilation	implementation dependent
do-sys	do-loop structures	implementation dependent
case-sys	CASE structures	implementation dependent
of-sys	OF structures	implementation dependent
orig	control-flow origins	implementation dependent
dest	control-flow destinations	implementation dependent
loop-sys	loop-control parameters	implementation dependent
nest-sys	definition calls	implementation dependent
i*x, j*x, k*x 3	any data type	0 or more cells

1 May be either a signed number or an unsigned number depending on context.

2 May be either a double-cell signed number or a double-cell unsigned number depending on context.

3 May be an undetermined number of stack entries of unspecified type. For examples of use, see 6.1.1370 EXECUTE, 6.1.2050 QUIT.

`See: 11.3.1 Data types, 12.3.1 Data types, 14.3.1 Data types, 16.3.1 Data types.`

Table 3.2 - Standard graphic characters

Hex	IRV	ASCII
20		
21	!	!
22	"	"
23	#	#
24	°	$
25	%	%
26	&	&
27	'	'
28	(	(
29	)	)
2A	*	*
2B	+	+
2C	,	,
2D	-	-
2E	.	.
2F	/	/
30	0	0
31	1	1
32	2	2
33	3	3
34	4	4
35	5	5
36	6	6
37	7	7
38	8	8
39	9	9
3A	:	:
3B	;	;
3C	<	<
3D	=	=
3E	>	>
3F	?	?
40	@	@
41	A	A
42	B	B
43	C	C
44	D	D
45	E	E
46	F	F
47	G	G
48	H	H
49	I	I
4A	J	J
4B	K	K
4C	L	L
4D	M	M
4E	N	N
4F	O	O
50	P	P
51	Q	Q
52	R	R
53	S	S
54	T	T
55	U	U
56	V	V
57	W	W
58	X	X
59	Y	Y
5A	Z	Z
5B	[	[
5C	\	\
5D	]	]
5E	^	^
5F	_	_

60	`	`
61	a	a
62	b	b
63	c	c
64	d	d
65	e	e
66	f	f
67	g	g
68	h	h
69	i	i
6A	j	j
6B	k	k
6C	l	l
6D	m	m
6E	n	n
6F	o	o
70	p	p
71	q	q
72	r	r
73	s	s
74	t	t
75	u	u
76	v	v
77	w	w
78	x	x
79	y	y
7A	z	z
7B	{	{
7C	|	|
7D	}	}
7E	~	~

3.1.3 Single-cell types

The implementation-defined fixed size of a cell is specified in address units and the corresponding number of bits. See E.2 Hardware peculiarities.

Cells shall be at least one address unit wide and contain at least sixteen bits. The size of a cell shall be an integral multiple of the size of a character. Data-stack elements, return-stack elements, addresses, execution tokens, flags, and integers are one cell wide.

`See: A.3.1.3 Single-cell types`

3.1.3.1 Flags

Flags may have one of two logical states, true or false. Programs that use flags as arithmetic operands have an environmental dependency.

A true flag returned by a standard word shall be a single-cell value with all bits set. A false flag returned by a standard word shall be a single-cell value with all bits clear.

`See: A.3.1.3.1 Flags`

3.1.3.2 Integers

The implementation-defined range of signed integers shall include {-32767..+32767}.

The implementation-defined range of non-negative integers shall include {0..32767}.

The implementation-defined range of unsigned integers shall include {0..65535}.

`See: A.3.1.3.2 Integers`

3.1.3.3 Addresses

An address identifies a location in data space with a size of one address unit, which a program may fetch from or store into except for the restrictions established in this Standard. The size of an address unit is specified in bits. Each distinct address value identifies exactly one such storage element. See 3.3.3 Data space.

The set of character-aligned addresses, addresses at which a character can be accessed, is an implementation-defined subset of all addresses. Adding the size of a character to a character-aligned address shall produce another character-aligned address.

The set of aligned addresses is an implementation-defined subset of character-aligned addresses. Adding the size of a cell to an aligned address shall produce another aligned address.

`See: A.3.1.3.3 Addresses`

3.1.3.4 Counted strings

A counted string in memory is identified by the address (c-addr) of its length character.

The length character of a counted string shall contain a binary representation of the number of data characters, between zero and the implementation-defined maximum length for a counted string. The maximum length of a counted string shall be at least 255.

`See: A.3.1.3.4 Counted strings`

3.1.3.5 Execution tokens

Different definitions may have the same execution token if the definitions are equivalent.

`See: A.3.1.3.5 Execution tokens`

3.1.4 Cell-pair types

A cell pair in memory consists of a sequence of two contiguous cells. The cell at the lower address is the first cell, and its address is used to identify the cell pair. Unless otherwise specified, a cell pair on a stack consists of the first cell immediately above the second cell.

`See: A.3.1.4 Cell-pair types`

3.1.4.1 Double-cell integers

On the stack, the cell containing the most significant part of a double-cell integer shall be above the cell containing the least significant part.

The implementation-defined range of double-cell signed integers shall include {-2147483647..+2147483647}.

The implementation-defined range of double-cell non-negative integers shall include {0..2147483647}.

The implementation-defined range of double-cell unsigned integers shall include {0..4294967295}. Placing the single-cell integer zero on the stack above a single-cell unsigned integer produces a double-cell unsigned integer with the same value. See 3.2.1.1 Internal number representation.

`See: A.3.1.4.1 Double-cell integers`

3.1.4.2 Character strings

A string is specified by a cell pair (c-addr u) representing its starting address and length in characters.

`See: A.3.1.3.4 Counted strings`

3.1.5 System types

The system data types specify permitted word combinations during compilation and execution.

3.1.5.1 System-compilation types

These data types denote zero or more items on the control-flow stack (see 3.2.3.2). The possible presence of such items on the data stack means that any items already there shall be unavailable to a program until the control-flow-stack items are consumed.

The implementation-dependent data generated upon beginning to compile a definition and consumed at its close is represented by the symbol colon-sys throughout this Standard.

The implementation-dependent data generated upon beginning to compile a do-loop structure such as DO ... LOOP and consumed at its close is represented by the symbol do-sys throughout this Standard.

The implementation-dependent data generated upon beginning to compile a CASE ... ENDCASE structure and consumed at its close is represented by the symbol case-sys throughout this Standard.

The implementation-dependent data generated upon beginning to compile an OF ... ENDOF structure and consumed at its close is represented by the symbol of-sys throughout this Standard.

The implementation-dependent data generated and consumed by executing the other standard control-flow words is represented by the symbols orig and dest throughout this Standard.

3.1.5.2 System-execution types

These data types denote zero or more items on the return stack. Their possible presence means that any items already on the return stack shall be unavailable to a program until the system-execution items are consumed.

The implementation-dependent data generated upon beginning to execute a definition and consumed upon exiting it is represented by the symbol nest-sys throughout this Standard.

The implementation-dependent loop-control parameters used to control the execution of do-loops are represented by the symbol loop-sys throughout this Standard. Loop-control parameters shall be available inside the do-loop for words that use or change these parameters, words such as I, J, LEAVE and UNLOOP.

3.2 The implementation environment

3.2.1 Numbers

```
See: A.3.2.1 Numbers
```

3.2.1.1 Internal number representation

This Standard allows one's complement, two's complement, or sign-magnitude number representations and arithmetic. Arithmetic zero is represented as the value of a single cell with all bits clear.

The representation of a number as a compiled literal or in memory is implementation dependent.

3.2.1.2 Digit conversion

Numbers shall be represented externally by using characters from the standard character set.

Conversion between the internal and external forms of a digit shall behave as follows:

The value in BASE is the radix for number conversion. A digit has a value ranging from zero to one less than the contents of BASE. The digit with the value zero corresponds to the character **0**. This representation of digits proceeds through the character set to the decimal value nine corresponding to the character **9**. For digits beginning with the decimal value ten the graphic characters beginning with the character **A** are used. This correspondence continues up to and including the digit with the decimal value thirty-five which is represented by the character **Z**. The conversion of digits outside this range is implementation defined.

See: A.3.2.1.2 Digit conversion

3.2.1.3 Free-field number display

Free-field number display uses the characters described in digit conversion, without leading zeros, in a field the exact size of the converted string plus a trailing space. If a number is zero, the least significant digit is not considered a leading zero. If the number is negative, a leading minus sign is displayed.

Number display may use the pictured numeric output string buffer to hold partially converted strings (see 3.3.3.6 Other transient regions).

3.2.2 Arithmetic

3.2.2.1 Integer division

Division produces a quotient q and a remainder r by dividing operand a by operand b. Division operations return q, r, or both. The identity b*q + r = a shall hold for all a and b.

When unsigned integers are divided and the remainder is not zero, q is the largest integer less than the true quotient.

When signed integers are divided, the remainder is not zero, and a and b have the same sign, q is the largest integer less than the true quotient. If only one operand is negative, whether q is rounded toward negative infinity (floored division) or rounded towards zero (symmetric division) is implementation defined.

Floored division is integer division in which the remainder carries the sign of the divisor or is zero, and the quotient is rounded to its arithmetic floor. Symmetric division is integer division in which the remainder carries the sign of the dividend or is zero and the quotient is the mathematical quotient **rounded towards zero** or **truncated**. Examples of each are shown in tables 3.3 and 3.4.

In cases where the operands differ in sign and the rounding direction matters, a program shall either include code generating the desired form of division, not relying on the implementation-defined default result, or have an environmental dependency on the desired rounding direction.

See: A.3.2.2.1 Integer division

Table 3.3 - Floored Division Example

Dividend	Divisor	Remainder	Quotient
10	7	3	1
-10	7	4	-2
10	-7	-4	-2
-10	-7	-3	1

Table 3.4 - Symmetric Division Example

Dividend	Divisor	Remainder	Quotient
10	7	3	1
-10	7	-3	-1
10	-7	3	-1
-10	-7	-3	1

3.2.2.2 Other integer operations

In all integer arithmetic operations, both overflow and underflow shall be ignored. The value returned when either overflow or underflow occurs is implementation defined.

`See: A.3.2.2.2 Other integer operations`

3.2.3 Stacks

`See: A.3.2.3 Stacks`

3.2.3.1 Data stack

Objects on the data stack shall be one cell wide.

3.2.3.2 Control-flow stack

The control-flow stack is a last-in, first out list whose elements define the permissible matchings of control-flow words and the restrictions imposed on data-stack usage during the compilation of control structures.

The elements of the control-flow stack are system-compilation data types.

The control-flow stack may, but need not, physically exist in an implementation. If it does exist, it may be, but need not be, implemented using the data stack. The format of the control-flow stack is implementation defined. Since the control-flow stack may be implemented using the data stack, items placed on the data stack are unavailable to a program after items are placed on the control-flow stack and remain unavailable until the control-flow stack items are removed.

`See: A.3.2.3.2 Control-flow stack`

3.2.3.3 Return stack

Items on the return stack shall consist of one or more cells. A system may use the return stack in an implementation-dependent manner during the compilation of definitions, during the execution of do-loops, and for storing run-time nesting information.

A program may use the return stack for temporary storage during the execution of a definition subject to the following restrictions:

- A program shall not access values on the return stack (using R@, R>, 2R@ or 2R>) that it did not place there using >R or 2>R;
- A program shall not access from within a do-loop values placed on the return stack before the loop was entered;
- All values placed on the return stack within a do-loop shall be removed before I, J, LOOP, +LOOP, UNLOOP, or LEAVE is executed;
- All values placed on the return stack within a definition shall be removed before the definition is terminated or before EXIT is executed.

`See: A.3.2.3.3 Return stack`

3.2.4 Operator terminal

`See: 1.2.2 Exclusions`

3.2.4.1 User input device

The method of selecting the user input device is implementation defined.

The method of indicating the end of an input line of text is implementation defined.

3.2.4.2 User output device

The method of selecting the user output device is implementation defined.

3.2.5 Mass storage

A system need not provide any standard words for accessing mass storage. If a system provides any standard word for accessing mass storage, it shall also implement the Block word set.

3.2.6 Environmental queries

The name spaces for ENVIRONMENT? and definitions are disjoint. Names of definitions that are the same as ENVIRONMENT? strings shall not impair the operation of ENVIRONMENT?. Table 3.5 contains the valid input strings and corresponding returned value for inquiring about the programming environment with ENVIRONMENT?.

Table 3.5 - Environmental Query Strings

String	Value data type	Constant?	Meaning
/COUNTED-STRING	n	yes	maximum size of a counted string, in characters
/HOLD	n	yes	size of the pictured numeric output string buffer, in characters
/PAD	n	yes	size of the scratch area pointed to by PAD, in characters
ADDRESS-UNIT-BITS	n	yes	size of one address unit, in bits
CORE	flag	no	true if complete core word set present (i.e., not a subset as defined in 5.1.1)
CORE-EXT	flag	no	true if core extensions word set present
FLOORED	flag	yes	true if floored division is the default
MAX-CHAR	u	yes	maximum value of any character in the implementation-defined character set
MAX-D	d	yes	largest usable signed double number
MAX-N	n	yes	largest usable signed integer
MAX-U	u	yes	largest usable unsigned integer
MAX-UD	ud	yes	largest usable unsigned double number
RETURN-STACK-CELLS	n	yes	maximum size of the return stack, in cells
STACK-CELLS	n	yes	maximum size of the data stack, in cells

If an environmental query (using ENVIRONMENT?) returns false (i.e., unknown) in response to a string, subsequent queries using the same string may return true. If a query returns true (i.e., known) in response to a string, subsequent queries with the same string shall also return true. If a query designated as constant in the above table returns true and a value in response to a string, subsequent queries with the same string shall return true and the same value.

See: A.3.2.6 Environmental queries, 7.3.1 Environmental queries, 8.3.1 Environmental queries, 9.3.4 Environmental queries, 10.3.2 Environmental queries, 11.3.3 Environmental queries, 12.3.4 Environmental queries, 13.3.2 Environmental queries, 14.3.2 Environmental queries, 15.3.1 Environmental queries, 16.3.2 Environmental queries, 17.3 Additional usage requirements.

3.3 The Forth dictionary

Forth words are organized into a structure called the dictionary. While the form of this structure is not specified by the Standard, it can be described as consisting of three logical parts: a name space, a code space, and a data space. The logical separation of these parts does not require their physical separation.

A program shall not fetch from or store into locations outside data space. An ambiguous condition exists if a program addresses name space or code space.

See: A.3.3 The Forth dictionary

3.3.1 Name space

The relationship between name space and data space is implementation dependent.

3.3.1.1 Word lists

The structure of a word list is implementation dependent. When duplicate names exist in a word list, the latest-defined duplicate shall be the one found during a search for the name.

3.3.1.2 Definition names

Definition names shall contain {1..31} characters. A system may allow or prohibit the creation of definition names containing non-standard characters.

Programs that use lower case for standard definition names or depend on the case-sensitivity properties of a system have an environmental dependency.

A program shall not create definition names containing non-graphic characters.

See: A.3.3.1.2 Definition names

3.3.2 Code space

The relationship between code space and data space is implementation dependent.

3.3.3 Data space

Data space is the only logical area of the dictionary for which standard words are provided to allocate and access regions of memory. These regions are: contiguous regions, variables, text-literal regions, input buffers, and other transient regions, each of which is described in the following sections. A program may read from or write into these regions unless otherwise specified.

See: A.3.3.3 Data space

3.3.3.1 Address alignment

Most addresses used in ANS Forth are aligned addresses (indicated by a-addr) or character-aligned (indicated by c-addr). ALIGNED, CHAR+, and arithmetic operations can alter the alignment state of an address on the stack. CHAR+ applied to an aligned address returns a character-aligned address that can only be used to access characters. Applying CHAR+ to a character-aligned address produces the succeeding character-aligned address. Adding or subtracting an arbitrary number to an address can produce an unaligned address that shall not be used to fetch or store anything. The only way to find the next aligned address is with ALIGNED. An ambiguous condition exists when @, !, , (comma), +!, 2@, or 2! is used with an address that is not aligned, or when C@, C!, or C, is used with an address that is not character-aligned.

The definitions of 6.1.1000 CREATE and 6.1.2410 VARIABLE require that the definitions created by them return aligned addresses.

After definitions are compiled or the word ALIGN is executed the data-space pointer is guaranteed to be aligned.

See: A.3.3.3.1 Address alignment

3.3.3.2 Contiguous regions

A system guarantees that a region of data space allocated using ALLOT, , (comma), C, (c-comma), and ALIGN shall be contiguous with the last region allocated with one of the above words, unless the restrictions in the following paragraphs apply. The data-space pointer HERE always identifies the beginning of the next data-space region to be allocated. As successive allocations are made, the data-space pointer increases. A program may perform address arithmetic within contiguously allocated regions. The last region of data space allocated using the above operators may be released by allocating a corresponding negatively-sized region using ALLOT, subject to the restrictions of the following paragraphs.

CREATE establishes the beginning of a contiguous region of data space, whose starting address is returned by the CREATEd definition. This region is terminated by compiling the next definition.

Since an implementation is free to allocate data space for use by code, the above operators need not produce contiguous regions of data space if definitions are added to or removed from the dictionary between allocations. An ambiguous condition exists if deallocated memory contains definitions.

`See: A.3.3.3.2 Contiguous regions`

3.3.3.3 Variables

The region allocated for a variable may be non-contiguous with regions subsequently allocated with , (comma) or ALLOT. For example, in:

```
        VARIABLE X  1 CELLS ALLOT
```

the region X and the region ALLOTted could be non-contiguous.

Some system-provided variables, such as STATE, are restricted to read-only access.

3.3.3.4 Text-literal regions

The text-literal regions, specified by strings compiled with S" and C", may be read-only.

A program shall not store into the text-literal regions created by S" and C" nor into any read-only system variable or read-only transient regions. An ambiguous condition exists when a program attempts to store into read-only regions.

3.3.3.5 Input buffers

The address, length, and content of the input buffer may be transient. A program shall not write into the input buffer. In the absence of any optional word sets providing alternative input sources, the input buffer is either the terminal-input buffer, used by QUIT to hold one line from the user input device, or a buffer specified by EVALUATE. In all cases, SOURCE returns the beginning address and length in characters of the current input buffer.

The minimum size of the terminal-input buffer shall be 80 characters.

The address and length returned by SOURCE, the string returned by PARSE, and directly computed input-buffer addresses are valid only until the text interpreter does I/O to refill the input buffer or the input source is changed.

A program may modify the size of the parse area by changing the contents of >IN within the limits imposed by this Standard. For example, if the contents of >IN are saved before a parsing operation and restored afterwards, the text that was parsed will be available again for subsequent parsing operations. The extent of permissible repositioning using this method depends on the input source (see 7.3.3 Block buffer regions and 11.3.4 Input source).

A program may directly examine the input buffer using its address and length as returned by SOURCE; the beginning of the parse area within the input buffer is indexed by the number in >IN. The values are valid for a limited time. An ambiguous condition exists if a program modifies the contents of the input buffer.

`See: RFI 0006 Writing to Input Buffers`

3.3.3.6 Other transient regions

The data space regions identified by PAD, WORD, and #> (the pictured numeric output string buffer) may be transient. Their addresses and contents may become invalid after:

- a definition is created via a defining word;
- definitions are compiled with : or :NONAME;
- data space is allocated using ALLOT, , (comma), C, (c-comma), or ALIGN.

The previous contents of the regions identified by WORD and #> may be invalid after each use of these words. Further, the regions returned by WORD and #> may overlap in memory. Consequently, use of one of these words can corrupt a region returned earlier by a different word. The other words that construct pictured numeric output strings (<#, #, #S, and HOLD) may also modify the contents of these regions. Words that display numbers may be implemented using pictured numeric output words. Consequently, . (dot), .R, .S, ?, D., D.R, U., and U.R could also corrupt the regions.

The size of the scratch area whose address is returned by PAD shall be at least 84 characters. The contents of the region addressed by PAD are intended to be under the complete control of the user: no words defined in this Standard place anything in the region, although changing data-space allocations as described in 3.3.3.2 Contiguous regions may change the address returned by PAD. Non-standard words provided by an implementation may use PAD, but such use shall be documented.

The size of the region identified by WORD shall be at least 33 characters.

The size of the pictured numeric output string buffer shall be at least (2*n) + 2 characters, where n is the number of bits in a cell. Programs that consider it a fixed area with unchanging access parameters have an environmental dependency.

`See: A.3.3.3.6 Other transient regions, 11.3.5 Other transient regions.`

3.4 The Forth text interpreter

Upon start-up, a system shall be able to interpret, as described by 6.1.2050 QUIT, Forth source code received interactively from a user input device.

Such interactive systems usually furnish a **prompt** indicating that they have accepted a user request and acted on it. The implementation-defined Forth prompt should contain the word **OK** in some combination of upper or lower case.

Text interpretation (see 6.1.1360 EVALUATE and 6.1.2050 QUIT) shall repeat the following steps until either the parse area is empty or an ambiguous condition exists:

a) Skip leading spaces and parse a name (see 3.4.1);

b) Search the dictionary name space (see 3.4.2). If a definition name matching the string is found:

1. if interpreting, perform the interpretation semantics of the definition (see 3.4.3.2), and continue at a);
2. if compiling, perform the compilation semantics of the definition (see 3.4.3.3), and continue at a).

c) If a definition name matching the string is not found, attempt to convert the string to a number (see 3.4.1.3). If successful:

1. if interpreting, place the number on the data stack, and continue at a);
2. if compiling, compile code that when executed will place the number on the stack (see 6.1.1780 LITERAL), and continue at a);

d) If unsuccessful, an ambiguous condition exists (see 3.4.4).

3.4.1 Parsing

Unless otherwise noted, the number of characters parsed may be from zero to the implementation-defined maximum length of a counted string.

If the parse area is empty, i.e., when the number in >IN is equal to the length of the input buffer, or contains no characters other than delimiters, the selected string is empty. Otherwise, the selected string begins with the next character in the parse area, which is the character indexed by the contents of >IN. An ambiguous condition exists if the number in >IN is greater than the size of the input buffer.

If delimiter characters are present in the parse area after the beginning of the selected string, the string continues up to and including the character just before the first such delimiter, and the number in >IN is changed to index immediately past that delimiter, thus removing the parsed characters and the delimiter from the parse area. Otherwise, the string continues up to and including the last character in the parse area, and the number in >IN is changed to the length of the input buffer, thus emptying the parse area.

Parsing may change the contents of >IN, but shall not affect the contents of the input buffer. Specifically, if the value in >IN is saved before starting the parse, resetting >IN to that value immediately after the parse shall restore the parse area without loss of data.

3.4.1.1 Delimiters

If the delimiter is the space character, hex 20 (BL), control characters may be treated as delimiters. The set of conditions, if any, under which a **space** delimiter matches control characters is implementation defined.

To skip leading delimiters is to pass by zero or more contiguous delimiters in the parse area before parsing.

3.4.1.2 Syntax

Forth has a simple, operator-ordered syntax. The phrase A B C returns values as if A were executed first, then B and finally C. Words that cause deviations from this linear flow of control are called control-flow words. Combinations of control-flow words whose stack effects are compatible form control-flow structures. Examples of typical use are given for each control-flow word in Annex A.

Forth syntax is extensible; for example, new control-flow words can be defined in terms of existing ones.

This Standard does not require a syntax or program-construct checker.

3.4.1.3 Text interpreter input number conversion

When converting input numbers, the text interpreter shall recognize both positive and negative numbers, with a negative number represented by a single minus sign, the character **-**, preceding the digits. The value in BASE is the radix for number conversion.

3.4.2 Finding definition names

A string matches a definition name if each character in the string matches the corresponding character in the string used as the definition name when the definition was created. The case sensitivity (whether or not the upper-case letters match the lower-case letters) is implementation defined. A system may be either case sensitive, treating upper- and lower-case letters as different and not matching, or case insensitive, ignoring differences in case while searching.

The matching of upper- and lower-case letters with alphabetic characters in character set extensions such as accented international characters is implementation defined.

A system shall be capable of finding the definition names defined by this Standard when they are spelled with upper-case letters.

3.4.3 Semantics

The semantics of a Forth definition are implemented by machine code or a sequence of execution tokens or other representations. They are largely specified by the stack notation in the glossary entries, which shows what values shall be consumed and produced. The prose in each glossary entry further specifies the definition's behavior.

Each Forth definition may have several behaviors, described in the following sections. The terms **initiation semantics** and **run-time semantics** refer to definition fragments, and have meaning only within the individual glossary entries where they appear.

`See: A.3.4.3 Semantics, RFI 0005.`

3.4.3.1 Execution semantics

The execution semantics of each Forth definition are specified in an **Execution:** section of its glossary entry. When a definition has only one specified behavior, the label is omitted.

Execution may occur implicitly, when the definition into which it has been compiled is executed, or explicitly, when its execution token is passed to EXECUTE. The execution semantics of a syntactically correct definition under conditions other than those specified in this Standard are implementation dependent.

Glossary entries for defining words include the execution semantics for the new definition in a *name* **Execution:** section.

3.4.3.2 Interpretation semantics

Unless otherwise specified in an **Interpretation:** section of the glossary entry, the interpretation semantics of a Forth definition are its execution semantics.

A system shall be capable of executing, in interpretation state, all of the definitions from the Core word set and any definitions included from the optional word sets or word set extensions whose interpretation semantics are defined by this Standard.

A system shall be capable of executing, in interpretation state, any new definitions created in accordance with 3. Usage requirements.

`See: A.3.4.3.2 Interpretation semantics`

3.4.3.3 Compilation semantics

Unless otherwise specified in a **Compilation:** section of the glossary entry, the compilation semantics of a Forth definition shall be to append its execution semantics to the execution semantics of the current definition.

`See: RFI 0007 Distinction between` *`immediacy`* `and` *`special compilation semantics`*`.`

3.4.4 Possible actions on an ambiguous condition

When an ambiguous condition exists, a system may take one or more of the following actions:

- ignore and continue;
- display a message;
- execute a particular word;
- set interpretation state and begin text interpretation;
- take other implementation-defined actions;
- take implementation-dependent actions.

The response to a particular ambiguous condition need not be the same under all circumstances.

3.4.5 Compilation

A program shall not attempt to nest compilation of definitions.

During the compilation of the current definition, a program shall not execute any defining word, :NONAME, or any definition that allocates dictionary data space. The compilation of the current definition may be suspended using [(left-bracket) and resumed using] (right-bracket). While the compilation of the current definition is suspended, a program shall not execute any defining word, :NONAME, or any definition that allocates dictionary data space.

`See: A.3.4.5 Compilation`

Table of Contents

Next Section