ISO/IEC JTC 1/SC 22/OWGV N 0228

Programming languages – C –Designated constructs, *by Olwen Morgan and Metriqa, Ltd*

Date	16 October 2009
Contributed by	Steve Michell (Canada)
Original file name	Microsoft Word - Metriqa C Coding Standard.pdf
Notes	

Moore, Jim

From: Olwen Morgan [olwen.morgan@btinternet.com]
Sent: Friday, October 16, 2009 7:58 AM
To: Moore, Jim
Subject: RE: [SC22-OWGV] Metriqa C Coding Standard

Jim,

As you suggested:

"The author and owner, Olwen Morgan, of the attached document, "Programming languages – C – Designated constructs", hereby grants permission for ISO/IEC JTC1/SC22/WG23 to post the document on its website and to adapt the text of the document for use in standards and other documents."

Regards,
Olwen Morgan

WD/MS1

Working Draft

Programming languages – C – Designated constructs

Contents

0 Foreword

0.1 Language restriction

In critical software applications, it is often desirable to restrict the use of certain programming language constructs. This standard defines constructs (called herein “designated constructs”) in the C programming language whose use may need to be restricted to meet dependability requirements in critical applications.

The use of a construct may be restricted for any of several reasons among which commonly cited ones are that it:

- is non-standard (S)
- has unspecified behaviour or yields an unspecified value (S)
- is likely to be misunderstood by programmers (E),
- has different meanings in closely related languages (S),
- may be prone to be implemented incorrectly (E),
- may impair important non-functional characteristics, including among others: analysability (SE), portability (S), interoperability (SE), security (E) or reliability (E),
- may impair internationalisation (SE),

For reasons marked “(S)”, relevant constructs can be determined from the language standard alone. For those marked “(E)” determination is on empirical grounds. For those marked “(SE)“, the determination has both and theoretical and an empirical basis. Constructs exhibiting such characteristics may be identified in all programming languages.

Ideally any empirical basis of restriction should be founded on clear evidence that a construct is associated with undesirable external attributes of software, particularly dependability attributes. In practice, however, little hard evidence of this nature is generally available and restrictions on some constructs are based on cogent reasoning or even just widely held beliefs about effects on the external attributes of code.

This standard sets out a rationale for the identification of each designated construct that it defines, whether based on evidence, reasoning or belief. It is hoped that codification of both constructs and associated rationales will permit hypotheses regarding usage and dependability to be stated clearly and subjected to rigorous tests.

0.2 Characterising constructs needing to be restricted in C

Usage restrictions typically comprise prohibitions of or limitations on the use of particular kinds of construct in context. In specifying such restrictions three distinct tasks arise:

- determining which constructs should be restricted in which contexts,
- characterising them unambiguously so that they can be identified in context by human reviewers or static checking tools,
- making the characterisations traceable to the language standard.

Among these tasks, characterisation is by far the most demanding. The easiest way to do it is with an appropriate metanotation. This standard uses the SYMELAR metanotation, which has been designed specifically for the purpose of defining language restrictions. SYMELAR is based on BNF and allows restricted constructs to be specified by reference to the C syntax as given in the standard, thus also providing suitable traceability.

The designated constructs identified in this standard are based on the diagnostics issued by a range of commercial C compilers and static analysis tools. Users of this standard should therefore have little difficulty in obtaining tools that will diagnose practically useful subsets of those constructs.

0.3 Basis for construction of coding manuals

The degree of language restriction appropriate to an application is generally related to its software integrity level [3]. Very high integrity applications may warrant the most severe restrictions [4]. Less critical applications may require only a few basic coding rules. Recognising this breadth of application, this standard identifies a wide range of designated constructs but does not specify any particular language subset based on restriction of any particular set of such constructs.

Within this standard each designated construct is identified by a designated construct reference number (DCRN). A user wishing to construct a coding manual by reference to this standard can do so by citing the DCRN of any construct he wishes to control and stating that nature of the restriction to which it is subject. Hence this standard serves as a meta-standard for the production of coding manuals.

1 Scope

This standard specifies:

- C language constructs, called "designated constructs" whose use it may be desirable to restrict in certain application domains,
- requirements for compliant coding manuals
- requirements for compliant diagnostic processors,
- requirements for canonically conforming implementations of the C programming language.

This standard does not specify:

- any particular set of designated constructs whose use is to be:
  - restricted in any particular application domain or
  - defined in any particular coding manual or
  - diagnosed by any particular diagnostic processor.
- any particular capabilities required of diagnostic processors such as:
  - the syntactic form of their diagnostic messages,
  - the manner in which such messages are presented to the user of the processor,
  - the manner in which such messages are associated with the language constructs to which they refer
  - rules of precedence among diagnostic messages whereby, for example, messages relating to contained constructs are presented before or after messages relating to their containing constructs,
  - rules governing the suppression of diagnostic messages for a construct when several could be issued.
- constructs for which the relation between usage and external attributes depends or is supposed to depend on the attributes of graph-theoretic models of source code, such as control flow graphs, data flow graphs and function-call trees.

2 References

2.1 Normative references

The following sources express requirements of this standard by virtue of reference to them within this standard:

[1] ISO/IEC 9899:1999 Programming Languages – C ***< add TC data >***

Note: Reference [1] is commonly called "C99".

[2] ISO/IEC 9899:1990 Programming Languages – C ***< add TC data>***

Note: Reference [2] is commonly called "C90".

2.2 Informative references

The following sources do not express requirements of this standard by virtue of reference to them within this standard (note that item numbering continues from clause 2.1 to ensure uniqueness of referencing):

[3] ISO/IEC 15026:1998 Information technology – System and software integrity levels

[4] ISO/IEC 61508 – ***<full title tbp >***

[5] ISO/IEC 9126-1:2001 Software engineering – Product quality – Part 1: Quality model

[6] *MISRA-C 2004: Guidelines for the Use of C in Critical Systems*, MIRA Ltd., 2004, ISBN 0952415623

[7] Hatton, L., *Safer C*, McGraw-Hill, 1995, ISBN 0-07-707640-0

[8] Koenig,A., *C Traps and Pitfalls*, Addison-Wesley, 1989, ISBN 0-201-17928-8

[9] Plum, T., *C Programming Guidelines*, Plum-Hall Inc., 1989, ISBN 0-911537-07-4.

3 Definitions and conventions

3.1 Terms, abbreviations and acronyms

Terms abbreviations and acronyms used in this standard have the meanings given for them in this clause. Where a standard is cited against the definition of a term, it indicates that the definition given here is derived or adapted from that given in the cited standard. In case of discrepancy between this standard and the cited standard, e.g. owing to updating of the source, the definition given in this standard takes precedence.

The symbol ≈ next to the citation of a standard denotes that the definition given here is technically equivalent (though possibly of different grammatical form) to that given in the cited standard. The symbol ≠ next to the citation of a standard denotes that the definition given here is not technically equivalent to that given in the cited standard.

accuracy *n*. (of a software product) the capability of the product to provide the right or agreed results with the needed degree of precision (≈ ISO 9126)

adaptability *n.* (of a software product) the capability of the product to be adapted for different specified environments without applying actions or means other than those provided for this purpose in the product considered (≈ ISO 9126)

analysability *n.* (of a software product) the capability of the product to be diagnosed for deficiencies or causes of failures in the software, or for parts to be modified to be identified (≈ ISO 9126)

base language standard *n.* the version of the C language standard, by reference to which this standard states definitions of designated constructs.(**Note:** For the current revision of this standard, the base-language standard is C99+TC1 – see Clause 3.1 Normative references)

BNF *abbr.* Backus-Naur form

bounded *adj.* (of a string manipulation function) having the property that it processes only a finite initial portion of any of its string arguments according to the value of an integer argument,

C++ style comment *n.* a comment of the form beginning with two slashes `//` as permitted in the C++ programming language,

changeability *n.* (of a software product) the capability of the product to enable modification to be implemented (≈ ISO 9126)

coding manual *n.* a document specifying constructs in a programming language and controls that are applied to their use in specified circumstances.

constraint *n.* restriction, either syntactic or semantic, by which the exposition of language elements is to be interpreted (≈ ISO/IEC 9899:1999)

construct *n.* a sequence of one or more preprocessing tokens or lexical tokens.

corresponding parameter *n.* of an *ARGUMENt*,

DCRN *abbr.* designated construct reference number

designated construct *n.* a construct defined in this standard and identified by a DCRN for the purpose of simplifying the construction of a coding manual.

diagnosed construct *n.* a construct for each occurrence of which in a program a diagnostic processor provides a diagnostic message.

diagnostic processor *n.* a processor that analyses source code and identifies occurrences of designated constructs within it by means of diagnostic messages.

E-behaviour *n.* the behaviour that the implementation provides for a construct in its execution environment

efficiency *n.* (of a software product) the capability of the product to provide appropriate performance, relative to the amount of resources used, under stated conditions (≈ ISO 9126)

fault-tolerance *n.* (of a software product) the capability of the product to maintain a specified level of performance in cases of software faults of of infringement of its specified interface (≈ ISO 9126)

format string *n.* an argument to a formatted I/O function that specifies the format conventions to be applied to subsequent arguments.

functionality *n.* (of a software product) the capability of the product to provide functions which meet stated and implied needs when the product is used under specified conditions (≈ ISO 9126)

implementation-defined behaviour *n.* unspecified behaviour where each implementation documents how the choice is made (≈ ISO/IEC 9899:1999)

implementation-defined value *n.* an unspecified value where each implementation document how the choice is made (≈ ISO/IEC 9899:1999)

implementation-dependent *adj.* (of the behaviour of a construct) unspecified and not necessarily defined.

implementation limit *n.* restriction imposed upon programs by the implementation (≈ ISO/IEC 9899:1999)

indeterminate value *n.* an unspecified value or a trap representation (≈ ISO/IEC 9899:1999)

initialising access *n.* an access to an object that establishes a value for the object by the behaviour of its initializer,

integrity level *n.* A denotation of a range of values of a property of an item necessary to maintain system risks within tolerable limits. For items that perform mitigating functions, the property is the reliability with which the item must perform the mitigating function. For items whose failure can lead to a threat, the property id the limit on the frequency of that failure (≈ISO/IEC 15026:1998)

internationalisation *n.* adaptation of a system for use in different countries or by people of different cultures having different conventions for the interpretation of human-readable output (e.g. formatting of dates, currency amounts, direction of reading)

maintainability *n.* (of a software product) the capability of the product to be modified (≈ ISO 9126)

maturity *n.* (of a software product) the capability of the product to avoid failure as a result of faults in the software (≈ ISO 9126)

minimal epsilon *n.* for a floating type, the floating-point value denoted by a representation in which all but the least significant bit of the mantissa are zero and the exponent is the least value for the type permitted in the **`<float.h>`** header. (**Note:** Such a number is necessarily subnormalised and is not necessarily within the implementation-defined range of representable floating-point values for the type concerned.)

modifying access *n.* an access to an object, other than an initialising access, that establishes a value for the object,

non-modifying access *n.* an access that is neither an initialising access nor a modifying access,

non-standard *adj.* generally, not having a form or not satisfying constraints given in the base language standard; specifically, in the context "a non-standard *x*" where *x* denotes an orthoclass, a construct that an implementation treats as an *x* but does not have a syntactic form derivable from *x* or whose behaviour violates a constraint of the standard.

non-standard preprocessor directive *n.* a source line whose first non-white-space character is hash **`#`** but that does not have the form of a *DIRECTIVE*.

null string *n.* a string containing no characters,

orthoclass *n.* a class of constructs represented by a non-terminal of the orthosyntax

orthorule *n.* a syntactic rule of the form specified in clause 4.1 of this specification.

orthosyntactic metasymbol *n.* any of the metasymbols specified in clause 4.1 of this specification.

orthosyntax *n.* a set of orthorules by which a C language construct is defined in this standard.

pairwise-confusable *adj.* (of identifiers) differing in corresponding character positions in the alphabetic case of characters or having in such corresponding positions respectively **`0`** and **`o`**, **`1`** and **`l`**, **`2`** and **`z`**, or **`5`** and **`s`**.

pararule *n.* a syntactic rule of the form specified in clause 4.2 of this specification.

parasyntactic metasymbol *n.* any of the metasymbols specified in clause 4.2 of this specification.

parasyntax *n.* a set of pararules by which a construct is defined in this standard.

portability *n.* (of a software product) the capability of the product to be transferred from one environment to another (≈ ISO 9126)

proscribed *adj.* (of an identifier) having a spelling that is pairwise-confusable with that of a keyword or another identifier, the spelling of the name of a standard function the spelling of a predefined macro name or identifier or a reserved spelling.

recursive *adj.* (of a function) having the property that its E-behaviour may contain one or more E-behaviours of itself; (of a macro) having the property that its T-behaviour may contain one or more T-behaviours of itself

redundant *adj.* (of a construct) capable of being removed without affecting the value of an expression or the occurrence of side effects,

reliability *n.* (of a software product) the capability of the product to maintain a specified level of performance when used under specified conditions (≈ ISO 9126)

resource utilisation *n.* (of a software product) the capability of the product to use appropriate amounts and types of resources when the product performs its function under stated conditions (≈ ISO 9126)

scalar expression *n.* an expression whose value is of scalar type,

security *n.* (of a software product) the capability of the product to protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or system are not denied access to them (≈ ISO 9126)

software integrity level *n.* the integrity level of a software item (≈ISO/IEC 15026:1998)

SYMELAR *acr.* SYntactic MEtanotation for LAnguage Restriction – the syntactic metanotation used in this standard for defining pararules.

T-behaviour *n.* the behaviour that the implementation provides for a construct in its translation environment

time behaviour *n.* (of a software product) the capability of the product to provide appropriate response and processing times and throughput rates when performing its function under stated conditions (≈ISO 9126)

undefined behaviour *n.* behaviour upon use of a nonportable or erroneous program construct or of erroneous data, for which (ISO/IEC 9899:1999) imposes no requirements (≈ISO/IEC 9899:1999)

Note: Possible undefined behaviour ranges from ignoring the situation completely with unpredictable results, to behaving during translation or program execution in a documented manner characteristic of the implementation (with or without issuance of a diagnostic message), to terminating a translation or execution (with the issuance of a diagnostic message).

understandability *n.* (of a software product) the capability of the product to enable the user or developer to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use (≠ ISO 9126)

unexecutable construct *n.* a construct for which the implementation can provide a T-behaviour but no E-behaviour.

unrepresentable *adj.* (of the value of an expression) not capable of being converted to the result type of the expression without loss of information.

unspecified behaviour *n*. behaviour where (ISO/IEC 9899:1999) provides two or more possibilities and imposes no further requirements on which is chosen in any instance (≈ ISO/IEC 9899:1999)

unspecified value *n*. a valid value of the relevant type where (ISO/IEC 9899:1999) imposes no requirements on which value is chosen in any instance (≈ ISO/IEC 9899:1999)

3.2 Conventions for syntactic description

This standard defines some (but not all) designated constructs by means of syntactic metanotation. For clarity of exposition syntactic rules are segregated into two groups called respectively *orthorules* and *pararules*. Orthorules are transliterated versions of the syntax rules given in the base language standard [1]. Pararules supplement the orthorules and are written in the SYMELAR notation. They define designated constructs only in conjunction with and by reference to the orthorules.

Notes: The prefix *ortho-* is from the Greek *ορθος* meaning straight, right, or proper. It is used here to emphasise the definitive character of orthosyntax, which is transliterated directly from the base language standard. The prefix *para-* is from the Greek *παρα,* meaning beside, and emphasises the supplementary character of the parasyntax.

3.2.1 Orthosyntax

The orthosyntactic metanotation used in this standard to specify the syntax of C language constructs is based on Backus-Naur Form (BNF). The notation has been modified from the original to permit greater convenience of description. Table 3.1 lists the meanings of the various metasymbols.

Table 3.1: Metasymbols in orthorules

Metasymbol	Meaning
=	shall be defined to be
<	direct concatenation (i.e. <u>without</u> an intervening white-space characters)
☐	spaced concatenation (i.e. <u>with</u> an intervening white space character).
|	alternatively, i.e. disjunction
;	end of definition
[*x*]	0 or 1 instances of *x*
`xyz`	the terminal symbol **`xyz`** (represented throughout in this standard by the use of bold courier typeface)
meta-identifier in *lower-case italics*	a nonterminal symbol of the orthosyntax

Except as indicated by the direct concatenation metasymbol or as provided by the base language standard, a sequence of terminal and nonterminal symbols in an orthorule implies the concatenation of the text that they ultimately represent with or without intervening white space characters. The orthosyntax in this standard differs from the syntax in the base language standard solely in the use of different metasymbols. Table 3.2 sets out the correspondence between the two syntaxes.

Table 3.2: Correspondence between orthosyntax and base language syntax

Orthosyntax metasymbol	Base language syntax metasymbol
=	:
<	No explicit symbol. The nature of concatenation is inferred from the context in the base language standard.
|	No explicit symbol. Alternatives start on a new line.
;	New line
[*x*]	x_{opt}
`xyz`	**`xyz`** (conventions are identical)
meta-identifier in *lower-case-italics*	meta-identifier in *lower-case italics* (conventions are identical)

3.2.2 Parasyntax

The parasyntactic metanotation used in this standard to specify designated constructs is also based on Backus-Naur Form (BNF). It uses all of the metasymbols of the orthosyntax except that meta-identifiers for paraclasses are written in italic small capitals. Nonterminal symbols of both the orthosyntax and parasyntax may appear in pararules. There are also curly brace metasymbols that allow recursive productions to be replaced with iterative ones. The metasymbols of the parasyntax are listed in Table 3.3.

Table 3.3: Metasymbols in pararules

Metasymbol	Meaning
{ *x* } { *Y* }	0 or more instances of *x*, one of more instances of *Y*
{ *x* | *Y* }	grouping: either *x* or *Y*
~	relative complement
&	Conjunction
meta-identifier in *ITALIC-SMALL-CAPITALS*	a nonterminal symbol of the parasyntax

3.2.3 Prose conventions

Use of the words *of*, *containing*, and *closest-containing*, when expressing a relationship between terminal or nonterminal symbols shall have the following meanings:

- the *x* of a *y* means the *x* occurring directly in a production defining *y*,
- the *x* in a *y* is synonymous with “the *x* of a *y*”,
- a *y* containing an *x* means any *y* from which an *x* is directly or indirectly derived,
- the *y* closest-containing an *x* means that *y* containing an *x* that does not contain another *y* containing that *x*,
- the y_1, y_2, …, or y_n closest-containing an *x* means that y_i for some *i* in [1,*n*], closest-containing an *x* such that for all *j* in [1,*n*] –[*i*], if a y_j contains that *x*, then that y_j contains that y_i.

In addition to the normal English rules for hyphenation, hyphenation is used in this standard to form compound words that represent meta-identifiers. All meta-identifiers that contain more than one word are written as a unit with hyphens joining the parts.

The meanings of forms that are literally different from but are grammatically entailed by the above forms shall correspond to the meaning of the forms by which they are entailed. For example, “an *x* whose *y* …” means “an *x* where a *y* is the *y* of that *x* …”.

Note: These prose conventions have been adapted from those used in ISO/IEC 7185 for the definition of the Pascal programming language.

3.3 Editorial presentation

From clause 5 onward, the structure and clause numbering of this standard follow those of the base language standard [1]. Subclauses within Clause 5 and succeeding clauses either state definitions or requirements or else have clause titles suffixed with “(NR)” to denote that they state no requirements. Except as explicitly provided otherwise in this standard, all clauses of the base language standard have corresponding clauses in this standard.

3.4 Designated constructs

3.4.1 Definitions

As far as possible, the definition of designated constructs is expressed using terms identical to, consistent with those of the base language standard. Where prose description would be unduly prolix, syntactic metanotation is used to help simplify the specifications. As far as possible such use is confined to the orthosyntax and pararules are used only where it is adjudged that no satisfactory alternative would be possible without them.

3.4.2 Numbering

Definitions for designated constructs are presented in tables. Each construct has an entry containing its unique designated construct reference number (DCRN), its definition and a rationale for its identification. The prefix of each DCRN identifies the clause in the base language standard which the relevant construct is specified.

3.4.3 Rationales

Where the behaviour for a designated construct is undefined, unspecified or implementation-defined, this is noted is bold type in the rationale entry for the construct. Where there is an obvious relationship of undefined, unspecified or implementation-defined aspects of behaviour to some non-functional attribute, the nature of the attribute is stated in bold small capitals.

For some constructs there is a significant consensus that programmers may be prone to make errors if they use them. In these circumstances the rationale for designating the construct is stated as defensive programming in bold type. Generally in this standard the term defensive programming refers to any convention aimed at reducing programmer error by controlling the use of constructs whose use is or may be considered to be conducive to programmer error.

Some designated constructs do not lead to undefined, unspecified or implementation defined behaviour but are designated on one or more of the following bases:

- they may not be portable to implementations conforming to earlier versions of the base language standard or to pre-standard implementations.
- their interpretation in C may differ from their interpretation in related languages based on C, such as C++,
- they may be some benefit in segregating them into particular parts of a translation unit,
- there is past evidence that C implementations have handled them incorrectly,
- there is reason to believe that their occurrence is indicative of programmer error,

Other than stating the basis on which a designated construct has been identified, this standard does not discuss the evidential or rational basis of what users may believe about the use of designated constructs.

3.5 Dependability attributes

Some practitioners use the term “dependability attributes” to refer to all non-functional attributes while others use the latter term to refer to specific kinds of non-functional attributes. Which particular sets of attributes are called dependability attributes varies from context to context but such sets commonly include the following:

- reliability
- maintainability
- availability
- security
- safety

Among these attributes security and safety are properties of the system as a whole rather than the software component considered in isolation. In this standard the term “dependability attribute” refers to the set of the above five non-functional attributes.

3.6 Relationship of non-functional attributes and language usage

Users of this standard should note, however, that relationships to non-functional attributes are stronger for code in development than for code in operational use. They should also appreciate the indirectness of the relationship between internal and external attributes of software. Coding conventions can facilitate the elimination of undesirable non-functional attributes but they cannot guarantee the presence of desirable ones.

Moreover, such facilitation is the *only way* in which they can contribute to external quality. Whether the surrounding practices actually exploit the facilitation is a matter of process quality, not internal product quality. Since process quality varies markedly among different development groups, it is not surprising if difficulties in controlling for process quality may to date have defeated attempts to demonstrate reproducible correlations between internal and external product quality.

3.7 Analysability

In any software engineering process, it is good practice to seek to detect faults in life cycle products at the earliest possible opportunity. In the current state of the art the best feasible practices in detecting programming errors are, in the order in which they can be most productively applied: static checking of code to remove problematic constructs, dynamic checking without execution (e.g. by abstract interpretation) and finally testing. In worst-case circumstances, the cost of detecting an error by testing may be two orders of magnitude greater than that of detecting it by static checking or dynamic analysis.

The use of dynamic analysis is a particularly powerful technique since it is commonly able to examine the potential behaviour of a program *for all possible input conditions*. In favourable circumstances, a dynamic analyser may be able to accomplish an analysis that is effectively equivalent to a program proof. In particular it may be possible to demonstrate that a program exhibits all and only those functions allocated to it in its specification.

The property of providing all and only specified functions is critical in attaining appropriate levels of certain dependability attributes, notably those of reliability and security. Accordingly it can be both desirable and cost-effective to ensure that program source code does not exhibit attributes that hinder the use of dynamic analysis techniques. In practice, this requires the systematic elimination of all constructs that impair the analysability of the code. Hence this standard identifies many constructs that impair such analysability.

4 Compliance

4.1 Coding manuals

4.1.1 Criteria

A coding manual shall comply with this standard if and only, wherever it cites a designated construct for which a definition exists in this standard, it cites the DCRN of that construct within this standard and states that the definition given in this standard is normative.

A coding manual complying with this standard shall be designated as *strictly compliant* if and only all of its designated constructs are cited by reference to their DCRNs in this standard.

4.2 Diagnostic processors

4.2.1 Criteria

A diagnostic processor shall comply with this standard if and only if it:

(a) is capable of analysing a C translation unit and identifying all occurrences within it of at least one class of designated constructs defined in this standard, and

(b) identifies such occurrences to its user by means of diagnostic messages that cite the DCRN of any construct so identified.

A diagnostic processor complying with this standard shall be designated as *strictly compliant* if and only if all of its diagnosed constructs are designated constructs defined in this standard.

4.2.2 Claims

A diagnostic processor purporting to comply with this standard shall be accompanied by a document that:

(a) identifies by means of a list of DCRNs, which of its diagnosed constructs are designated constructs defined in this standard,

(b) wherever it cannot identify all instances of a designated construct states a characterisation of the subclass of instances that it can identify.

Note: Clause 4.2.2(b) is intended to allow legitimate claims of conformance for diagnostic processors that perform no or only limited dynamic analysis and may therefore be able to identify only those occurrences of designated constructs that are identifiable by purely static methods.

5 Environment

5.1 Conceptual models (NR)

5.1.1 Translation environment

5.1.1.1 Program structure (NR)

5.1.1.2 Translation phases

Designated constructs:

DCRN	Definition	Rationale
5.1.1.2-1	A nonempty source file ending in a new-line character that is immediately preceded by a backslash character.	Behaviour for such a construct is **undefined**.
5.1.1.2-2	A nonempty source file ending in a partial preprocessing token or a partial comment.	Behaviour for such a construct is **undefined**.
5.1.1.2-3	A new-line character that is preceded by a white space character.	Some users prefer to suppress trailing white space characters for ergonomic convenience when using editors. Insofar as this makes it easier to amend code, it may contribute marginally to **MAINTAINABILITY.**
5.1.1.2-4	A character sequence that results from token concatenation and is a *universal-character-name*.	Behaviour for such a character sequence is **undefined**.
5.1.1.2-5	A source character for which there is no corresponding execution character.	Behaviour for such a character sequence is **implementation-defined**.
5.1.1.2-6	A sequence of two adjacent identifiers.	Such a construct was tolerated by some pre-standard implementations but behaviour is **undefined** for conforming implementations.
5.1.1.2-7	A tab character used to provide indentation	Expansion of tab characters is implementation-dependent. Consistent indentation style may be lost if source code relying on such expansion is ported between systems. Hence the use of tab characters for indentation impairs a (fairly minor) aspect of **PORTABILITY**.
5.1.1.2-8	A construct exhibiting different brace styles.	Some users believe that the use of a single brace style promotes the **UNDERSTANDABILITY** of code.

5.1.1.3 Diagnostics (NR)

Note: Some of the designated constructs defined in this standard can be detected by exclusively static methods. For many constructs, however, static methods may not be able to detect all cases of the construct that satisfy its definition. Where a diagnostic processor cannot detect all cases, this does not in itself render that processor noncompliant with this standard, provided that the processor is accompanied by documentation stating, for each relevant DCRN, criteria that discriminate between detected and undetected cases and state any differences in diagnostic messages corresponding to different forms of the detected subcases.

5.1.2 Execution environments

Designated constructs:

DCRN	Definition	Rationale
5.1.2-1	A construct for which behaviour may vary according to the manner and timing of static initialization.	The manner and timing of static initialization are **unspecified**.

5.1.2.1 Freestanding environment (NR)

Note: Both C90 and C99 define the notion of a freestanding implementation. The purpose in so doing was to provide for compliance of implementations whose execution environments are embedded processors for which provision of all standard libraries would be either unnecessary or unduly onerous. Most compilers for embedded targets do, however, provide library facilities surpassing the minimal set required of freestanding implementations. A coding manual for the use of C under such an implementation may therefore be significantly more restrictive than one for a hosted implementation. Users of this standard who code for both types of implementation may therefore wish to consider whether they need separate coding manuals for freestanding and hosted environments.

5.1.2.2 Hosted environment

5.1.2.2.1 Program startup

Parasyntax:

STD-MAIN-FUNC-DEC = *FUNCTION-PROTOTYPE*
&
`int main (void)`

| *FUNCTION-PROTOTYPE*
&
`int main (int argc, char *argv[])` ;

Designated constructs:

DCRN	Definition	Rationale
5.1.2.2-1	A *FUNCTION-PROTOTYPE* for **`main`** that is not equivalent to a *STD-MAIN-FUNC-DEC*.	Behaviour is **undefined**.
5.1.2.2-2	A *FUNCTION-PROTOTYPE* for **`main`** that is not a *STD-MAIN-FUNC-DEC*.	Some users believe that adherence to the standard form promotes **UNDERSTANDABILITY**.
5.1.2.2-3	A *translation-unit* containing no *function-definition* for **`main`**.	Behaviour is **undefined**.

5.1.2.2.2 Program execution (NR)

5.1.2.2.3 Program termination

Designated constructs:

5.1.2.2.3-1	A *FUNCTION-PROTOTYPE* for **`main`** in which the return type is not compatible with **`int`**.	The termination status returned to the host environment is **unspecified**.

5.1.2.3 Program execution

Designated constructs:

DCRN	Definition	Rationale
5.1.2.3-1	An unexecutable construct (see note 1 below).	Wherever such constructs occur they are highly likely to have resulted from programmer error and the program's behaviour may not be what the programmer intends and the program may not provide its specified **FUNCTIONALITY**.
5.1.2.3-2	A construct whose E-behaviour contains both a modifying and a non-modifying access to an object between consecutive sequence points.	The order of occurrence of the accesses is **unspecified** (see note 2 below).
5.1.2.3-3	A construct whose E-behaviour contains more than one side effect between consecutive sequence points.	The order of occurrence of the side effects is **unspecified** (see note 2 below).

Note 1: Not all unexecutable constructs can be detected by purely static means.

For example, if in the code fragment:

```
if (x < 0) foo_a() else foo_b();
```

the variable **x** is of unsigned integral type, then **foo_a()** is an unexecutable construct and its unexecutability is determinable solely from the type of **x** and the value of zero against which **x** is compared.

In contrast, in the code fragment:

```
int i = 1;

while (i != 3)
{
   i = (i+i) % 7;
}

foo();
```

foo() is unexecutable because the loop causes **i** to cycle through the quadratic residues modulo 7 but, since 3 is not such a quadratic residue, the loop never terminates. This condition is impossible to detect without dynamic analysis and even then some methods of dynamic analysis may fail to detect it.

Note 2: The order of occurrence of accesses and side effects depends on the orders of evaluation of the operands of expression, which are unspecified.

5.2 Environmental considerations

5.2.1 Character sets

Designated constructs:

DCRN	Definition	Rationale
5.2.1-1	A character not in the basic source character set.	Behaviour may be **undefined** or **locale-specific.**.

5.2.1.1 Trigraph sequences

Designated constructs:

DCRN	Definition	Rationale

5.2.1.1-1	A trigraph sequence.	Some users believe that trigraphs impair the **UNDERSTANDABILITY** of code. Also, they may not supported by pre-standard implementations.

5.2.1.2 Multibyte characters

Designated constructs:

DCRN	Definition	Rationale
5.2.1.2-1	A multibyte character.	Support for multibyte characters is **locale-specific**.
5.2.1.2-2	A byte with all bits zero occurring as the second or a subsequent byte of a multibyte character.	Behaviour is **undefined**.
5.2.1.2-3	A comment, *string-literal*, *character-constant* or *header-name* that does not begin in the initial shift state.	Behaviour is **undefined**.
5.2.1.2-4	A comment, *string-literal*, *character-constant* or *header-name* that does not consist of a sequence of valid multibyte characters.	Behaviour is **undefined**.

5.2.2 Character display semantics (NR)

5.2.3 Signals and interrupts (NR)

5.2.4 Environmental limits

5.2.4.1 Translation limits

Designated constructs:

DCRN	Definition	Rationale
5.2.4.1-1	An *external-definition* in which an occurrence of any *iteration-statement* or *selection-statement* causes the depth of nesting of such statements to exceed 127 (C90 = 15).	Such an *external-definition* exceeds minimum implementation **limits**.
5.2.4.1-2	A *preprocessing-file* in which an occurrence of any *IF-DIRECTIVE*, *IFDEF-DIRECTIVE* or *IFNDEF-DIRECTIVE* causes the depth of nesting of such directives to exceed 63 (C90 = 8).	Such a *preprocessing-file* exceeds minimum implementation **limits**.
5.2.4.1-3	A *declarator* containing more than 12 (C90 = 12) *modifiers*.	Such a *declarator* exceeds minimum implementation **limits**.
5.2.4.1-4	A *declarator* in which the nesting of parentheses exceeds 63 (C90 = 31).	Such a *declarator* exceeds minimum implementation **limits**.

5.2.4.1-5	An *expression* in which the nesting of parentheses exceeds 63 (C90 = 32) levels.	Such an *expression* exceeds minimum implementation **limits**.
5.2.4.1-6	A *translation-unit* containing more than 4095 (C90 = 511) distinct *identifier* having external linkage.	Such a *translation-unit* exceeds minimum implementation **limits**.
5.2.4.1-7	A *compound-statement* that is the scope of more than 511 (C90 = 127) distinct *identifier*.	Such a *compound-statement* exceeds minimum implementation **limits**.
5.2.4.1-8	A *preprocessing-translation-unit* containing more than 4095 (C90 = 1024) macro definitions.	Such a *preprocessing-translation-unit* exceeds minimum implementation **limits**.
5.2.4.1-9	A *function-definition* closest-containing more than 127 (C90 = 31) *PARAMETER-DECLARATOR*.	Such a *function-definition* exceeds minimum implementation **limits**.
5.2.4.1-10	A *FUNCTION-CALL-EXPRESSION* closest-containing more than 127 (C90 = 31) *ARGUMENT*.	Such a *FUNCTION-CALL-EXPRESSION* exceeds minimum implementation **limits**.
5.2.4.1-11	A *FLIKE-DEFINE-DIRECTIVE* whose *identifier-list* closets-contains more than 127 (31) *identifier*.	Such a *FLIKE-DEFINE-DIRECTIVE* exceeds minimum implementation **limits**.
5.2.4.1-12	A *MACRO-INVOCATION* whose *identifier-list* closest-contains more than 127 (C90 = 31) *identifier*.	Such a *MACRO-INVOCATION* exceeds minimum implementation **limits**.
5.2.4.1-13	A logical line that exceeds 4095 (C90 = 509) characters.	Such a logical line exceeds minimum implementation **limits**.
5.2.4.1-14	A *character-string-literal* or *wide-string-literal* that contains more than 4095 (509) characters.	Such a literal exceeds minimum implementation **limits**.
5.2.4.1-15	A *declaration* of an object whose size exceeds 65535 (C90 = 32767) bytes.	Such an object exceeds minimum implementation **limits**.
5.2.4.1-16	An *INCLUDE-DIRECTIVE* for which an implementation causes the depth of nesting of included files to exceed 15 (C90 = 8).	Behaviour is **undefined**.
5.2.4.1-17	A *SWITCH-BODY* that closest-contains more than 1023 (C90 = 257) *CASE-CLAUSE*.	Such a *SWITCH-BODY* exceeds minimum implementation **limits**.
5.2.4.1-18	A *struct-declaration* that closest-contains more than 1023 (C90 = 127) *declarator*.	Such a *struct-declaration* exceeds minimum implementation **limits**.
5.2.4.1-19	An *enumerator-list* containing more than 1023 (C90 = 127) *enumeration-constant*.	Such an *enumerator-list* exceeds minimum implementation **limits**.
5.2.4.1-20	A *struct-declaration-list* whose occurrence causes the depth of nesting of *struct-declaration-list* to exceed 63 (C90 = 15).	Such a *struct-declaration-list* exceeds minimum implementation **limits**.

Note: In this clause parenthesised items in the definitions of designated constructs denote corresponding limits in C90.

5.2.4.2 Numerical limits

5.2.4.2.1 Sizes of integer types `<limits.h>` (NR)

5.2.4.2.2 Characteristics of floating types `<float.h>` (NR)

Designated constructs:

DCRN	Definition	Rationale
5.2.4.2.2-1	A *preprocessing-file* in which the *MACRO-NAME* **`FLT_ROUNDS`** expands to a *constant-expression* whose value is –1.	The **implementation-defined** rounding mode is not determinable, which impairs ANALYZABILITY of codes for numerical processes.
5.2.4.2.2-2	A *preprocessing-file* in which the *MACRO-NAME* **`FLT_EVAL_METHOD`** expands to a *constant-expression* whose value is –1.	The **implementation-defined** evaluation method is not determinable, which impairs ANALYZABILITY of codes for numerical processes.
5.2.4.2.2-3	A *preprocessing-file* in which the *MACRO-NAME* **`FLT-EPSILON`** expands to a *constant-expression* whose value is not a minimal epsilon for the **`float`** type.	A value that is not a minimal epsilon may be indicative of a crude implementation of floating-point arithmetic, which may impair the ACCURACY of floating-point computation.
5.2.4.2.2-4	A *preprocessing-file* in which the *MACRO-NAME* **`DBL-EPSILON`** expands to a *constant-expression* whose value is not a minimal epsilon for the **`double`** type.	A value that is not a minimal epsilon may be indicative of a crude implementation of floating-point arithmetic, which may impair the ACCURACY of floating-point computation.
5.2.4.2.2-5	A *preprocessing-file* in which the *MACRO-NAME* **`LDBL-EPSILON`** expands to a *constant-expression* whose value is not a minimal epsilon for the **`long double`** type.	A value that is not a minimal epsilon may be indicative of a crude implementation of floating-point arithmetic, which may impair the ACCURACY of floating-point computation.

Note: The value to which a *MACRO-NAME* in **`<float.h>`** expands may not be the same as a value determined for the corresponding quantity by direct computation.

6 Language

6.1 Notation (NR)

6.2 Concepts

6.2.1 Scopes of identifiers

Designated constructs:

DCRN	Definition	Rationale
6.2.1-1	An *identifier* having no part of its scope outside a *FUNCTION-PROTOTYPE*.	Either the *FUNCTION-PROTOTYPE* in which the *identifier* occurs differs from the *FUNCTION-PROTOTYPE* of the corresponding *function-definition*, or there is no corresponding *function-definition*. Some users believe that such usage impairs **UNDERSTANDABILITY**.
6.2.1-2	An *identifier* having block scope where that block scope is enclosed by the scope of another *identifier* having the same spelling.	Some users believe that the presence of such identifiers impairs **UNDERSTANDABILITY**.
6.2.1-3	An *identifier* that is not the *identifier* of at least one *direct-declarator* within the *translation-unit* in which it occurs.	Such an *identifier* is undeclared and will be treated as if it had been declared with type **`int`**. Some users believe that allowing types to default to **`int`** impairs the **UNDERSTANDABILITY** of ode.

6.2.2 Linkages of identifiers

Designated constructs:

DCRN	Definition	Rationale
6.2.2-1	An *identifier* appearing with both internal and external linkage in a single *translation-unit*.	Behaviour is **undefined**.
6.2.2-2	An *identifier* with internal linkage or a *MACRO-NAME* that does not differ from a distinct *identifier* with internal linkage or *MACRO-NAME* names that do not differ within the first 63 (C90 = 31) characters, regardless of alphabetic case.	Behaviour is **undefined**.
6.2.2-3	An *identifier* with external linkage or a *MACRO-NAME* that does not differ from a distinct *identifier* with external linkage or *MACRO-NAME* names that do not differ within the first 31 (C90 = 6) characters, regardless of alphabetic case.	Behaviour is **undefined**.
6.2.2-4	An *identifier* that has block scope and that is declared with the *storage-class-specifier* **`extern`**.	The behaviour provided by pre-standard implementations may differ from that provided by a conforming implementation thus impairing **PORTABILITY**.

6.2.3 Name spaces of identifiers

Designated constructs:

DCRN	Definition	Rationale
6.2.3-1	An *identifier* that is declared in one more than one of the name spaces of a *translation-unit*.	Some users believe that the presence of such identifiers impairs **UNDERSTANDABILITY**.

6.2.4 Storage durations of identifiers

Designated constructs:

DCRN	Definition	Rationale
6.2.4-1	An access to an object outside its lifetime.	Behaviour is **undefined**.
6.2.4-2	A non-modifying access to an object whose value is indeterminate.	Behaviour may be **undefined** depending on the context of usage.
6.2.4-3	A *FUNCTION-BLOCK* containing an *expression* that denotes the lvalue of an object whose lifetime is not contained in that *FUNCTION-BLOCK*.	Some users believe that access by a function to objects not local to its *FUNCTION-BLOCK* impairs the **UNDERSTANDABILITY** and **MAINTAINABILITY** of the code. Non-local accesses also contravene certain special-purpose conventions such as data-flow programming.

6.2.5 Types (NR)

6.2.6 Representations of types

6.2.6.1 General (NR)

6.2.6.2 Integer types (NR)

6.2.7 Compatible and composite types

6.3 Conversions

6.3.1 Arithmetic operands (NR)

6.3.1.1 Boolean, character, and integers (NR)

6.3.1.2 Boolean type (NR)

6.3.1.3 Signed and unsigned integers

Designated constructs:

DCRN	Definition	Rationale
6.3.1.3-1	A construct whose behaviour converts a value of integral type to an integral type in which its value cannot be represented.	The effects of such a conversion are **implementation-defined**.

Note: Several sub-cases can be identified for DCRN **6.3.1.3-1** and a diagnostic processor may distinguish among them by issuing different diagnostic messages. In particular a diagnostic processor may distinguish cases in which the construct concerned is an *EXPLICIT-CAST-EXPR*, where the explicit nature of the conversion may indicate a particular intention of the programmer.

6.3.1.4 Real, floating and integer (NR)

6.3.1.5 Real floating types

Designated constructs:

DCRN	Definition	Rationale
6.3.1.5-1	A construct whose behaviour converts other a value of floating type to a value of a shorter floating type.	The effects of the conversion may be **undefined** or **implementation-defined** depending on the value concerned.
6.3.1.5-2	A construct whose behaviour converts a value of floating type to a value of integral type.	The effects of the conversion may be **undefined** or **implementation-defined** depending on the value concerned.
6.3.1.5-3	A construct whose behaviour converts a value of integral type to a value of floating type.	The effects of the conversion may be **undefined** or **implementation-defined** depending on the value concerned.

Note: Several sub-cases can be identified for each of DCRNs **6.3.1.5-1**, **6.3.1.5-2** and **6.3.1.5-3**. A diagnostic processor may distinguish among them by issuing different diagnostic messages. . In particular a diagnostic processor may distinguish cases in which the construct concerned is an *EXPLICIT-CAST-EXPR*, where the explicit nature of the conversion may indicate a particular intention of the programmer.

6.3.1.6 Real and complex (NR)

6.3.1.7 Usual arithmetic conversions (NR)

6.3.2 Other operands

6.3.2.1 Lvalues, arrays and function designators

Designated constructs

DCRN	Definition	Rationale
6.3.2.1-1	An lvalue that does not denote an object when evaluated.	E-behaviour is **undefined**.

6.3.2.2 Void

Designated constructs

DCRN	Definition	Rationale
6.3.2.2-1	An *expression* that is not an *expression-statement* and whose type is `void`.	Some users, believing such constructs likely to have resulted from programmer error, regard their prohibition as **defensive programming**.

Note: Particular sub-cases can be identified for DCRN **6.3.2.2-1**, e.g. when the construct concerned is the *expression* of an *EXPLICIT- COMMA-EXPRESSION* or when it is an *EXPLICIT-CAST-EXPR* that casts to `void`. A diagnostic processor may distinguish among sub-cases by issuing different diagnostic messages.

6.3.2.3 Pointers (NR)

6.4 Lexical elements

Orthosyntax:

token = *keyword*
| *identifier*
| *constant*
| *string-literal*
| *punctuator* ;

preprocessing-token = *header-name*
| *identifier*
| *pp-number*
| *character-constant*
| *string-literal*
| *operator*
| *punctuator*
| each non-white-space character that cannot be one of the above ;

Designated constructs:

DCRN	Definition	Rationale
6.4-1	A *preprocessing-token* that cannot be converted to an actual token.	T-behaviour of the *preprocessing-token* is **undefined** which impairs **analysability**.
6.4-2	A *identifier* that is not a *keyword* but that an implementation does not treat as an *identifier*.	Such a construct is likely to be a non-standard keyword supported by the implementation. Its presence in code will impair **analysability**.
6.4-3	A ` or ‟ that is not a *header-name,* an *identifier*, a *pp-number*, a *character-constant*, a *string-literal*, an *operator* or a *punctuator*	T- behaviour is **undefined**.

Note: Examples of DCRN 6.4-2 are common, for example in C compilers provided as part of C++ implementations or in cross-compilers for embedded targets. A diagnostic processor may distinguish among different sub-cases by issuing different diagnostic messages.

6.4.1 Keywords

Orthosyntax:

keyword = **`auto`** | **`break`** | **`case`** | **`char`** | **`const`** | **`continue`** |
`default` | **`do`** | **`double`** | **`else`** | **`enum`** | **`extern`** |
`float` | **`for`** | **`goto`** | **`if`** | **`inline`** | **`int`** | **`long`** |
`register` | **`restrict`** | **`return`** | **`short`** | **`signed`** |
`sizeof` | **`static`** | **`struct`** | **`switch`** | **`typedef`** |
`union` | **`unsigned`** | **`void`** | **`volatile`** | **`while`** | **`_Bool`** |
`_Complex` | **`_Imaginary`** ;

Parasyntax:

NON-C90-KEYWORD = **`inline`** | **`restrict`** | **`_Bool`**
| **`_Complex`** | **`_Imaginary`** ;

Designated constructs:

DCRN	Definition	Rationale
6.4.1-1	A *NON-C90-KEYWORD*.	The presence of such keywords impairs **PORTABILITY** of code among implementations conforming to earlier version of the base language standard.

6.4.2 Identifiers

6.4.2.1 General

Orthosyntax:

identifier = *identifier-nondigit*
| *identifier* < *identifier-nondigit*
| *identifier* < *digit*

identifier-nondigit = *nondigit*
| *universal character-name*
| other implementation-defined characters ;

non-digit = **`_ | 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`**
| **`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`**

digit = **`0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`** ;

Designated constructs:

DCRN	Definition	Rationale
6.4.2.1-1	A proscribed *identifier*.	Some users believe that the presence of proscribed *identifier*s impairs the **understandability** and thence the **maintainability** of the code.
6.4.2.1-2	An *identifier* that contains a *universal-character-name*.	The presence of a *universal-character-name* in an *identifier* impairs **PORTABILITY** of code among implementations conforming to earlier version of the base language standard.
6.4.2.1-3	An *identifier* that contains an *identifier-non-digit* that is neither a *non-digit* nor *universal-character-name*.	Behaviour is **implementation-defined**.

Note: Diagnostic processors identifying occurrences of DCRN 6.4.2.1-1 may distinguish between occurrences in standard headers and elsewhere in a *preprocessing-file*. They may also distinguish instances of pairwise confusability from other instances.

6.4.2.2 Predefined identifiers (NR)

6.4.3 Universal character names (NR)

Orthosyntax:

universal-character-name = `\u` < *hex-quad*
| `\U` < *hex-quad* ;

hex-quad = *hexadecimal-digit* < *hexadecimal-digit* <
hexadecimal-digit < *hexadecimal-digit* ;

Designated constructs:

DCRN	Definition	Rationale
6.4.2.1-1	A *universal-character-name*.	Correct use of universal character-names is critical in internationalisation of software. Some users consider it useful for a diagnostic processor to identify all occurrences of such characters to facilitate manual review.
6.4.2.1-2	A *universal-character-name* that specifies a character whose short identifier is less than 00A0 (other than 0024, 0040, or 0060) or in the range D800 to DFFF inclusive.	Behaviour is **undefined.**

6.4.4 Constants

Orthosyntax:

constant = *floating-constant*
| *integer-constant*
| *enumeration-constant*
| *character-constant* ;

Designated constructs:

DCRN	Definition	Rationale
6.4.4-1	A *constant* whose value is unrepresentable in an object of arithmetic type.	Behaviour of an unrepresentable value is **undefined**.

6.4.4.1 Integer constants

Orthosyntax:

integer-constant = *decimal-constant* < [*integer-suffix*]
| *octal-constant* < [*integer-suffix*]
| *hexadecimal-constant* < [*integer-suffix*] ;

decimal-constant = *nonzero-digit*
| *decimal-constant* < *digit* ;

octal-constant = `0`
| *octal-constant* < *octal-digit* ;

hexadecimal-constant = *hexadecimal-prefix* < *hexadecimal-digit*
| *hexadecimal-constant* < *hexadecimal-digit* ;

hexadecimal-prefix = **0x** | **0X** ;

nonzero-digit = **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** ;

octal-digit = **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** ;

hexadecimal-digit = **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**
| **a** | **b** | **c** | **d** | **e** | **f**
| **A** | **B** | **C** | **D** | **E** | **F** ;

integer-suffix = *unsigned-suffix* < [*long-suffix*]
| *unsigned-suffix* < *long-long suffix*
| *long-suffix* < [*unsigned-suffix*]
| *long-long-suffix* < [*unsigned-suffix*] ;

unsigned-suffix = **u** | **U** ;

long-suffix = **l** | **L** ;

long-long-suffix = **ll** | **LL** ;

DCRN	Definition	Rationale
6.4.4.1-1	An *integer-constant* that denotes a value of a type other than **int** but does not contain an *integer-suffix*.	Some users believe that failure to use an explicit suffix for such an *integer-constant* impairs **UNDERSTANDABILITY**.
6.4.4.1-2	An *integer-constant* that: (a) has not resulted from expansion of a macro, and (b) is not contained by an *initializer*, and (c) denotes a value that is neither zero nor one.	Such an *integer-constant* (often called a “magic constant”) may represent a configuration parameter. Some users believe that failure to give it a symbolic definition, either as a macro or a value of const-qualified type, impairs **MAINTAINABILITY**.
6.4.4.1-3	A *long-long-suffix*.	The presence of such suffices may impair **PORTABILITY** among implementations conforming to earlier versions of the base language standard.

Note: A diagnostic processor may identify constructs similar to DCRN 6.4.4.1-2 such as a *integer-constant* that denotes a value other than zero or one, e.g. two. The values zero and one are excluded from the definition of DCRN 6.4.4.1-2 because most uses of them are not magic numbers.

6.4.4.2 Floating constants

Orthosyntax:

floating-constant = *decimal-floating-constant*
| *hexadecimal-floating-constant* ;

decimal-floating-constant = *fractional-constant*
< [*exponent-part*] < [*floating-suffix*]

```
                                |       digit-sequence < exponent-part < [ floating-suffix ] ;

hexadecimal-floating-constant   =       hexadecimal-prefix
                                                 < hexadecimal-fractional-constant
                                                 < binary-exponent-part
                                                 < [ floating-suffix ]
                                |       hexadecimal-prefix
                                                 < hexadecimal-digit-sequence
                                                 < binary-exponent-part
                                                 < [ floating-suffix ] ;

fractional-constant             =       [ digit-sequence ] < . < digit-sequence
                                |       digit-sequence ;

exponent-part                   =       e < [ sign ] < digit-sequence
                                |       E < [ sign ] < digit-sequence ;

sign                            =       + | - ;

digit-sequence                  =       digit
                                |       digit-sequence < digit ;
hexadecimal-fractional-constant         =       [ hexadecimal-digit-sequence ] < .
                                                         < hexadecimal-digit-sequence
                                        |       hexadecimal-digit-sequence < . ;

binary-exponent-part                    =       p < [ sign ] < digit-sequence
                                        |       P < [ sign ] < digit-sequence  ;

hexadecimal-digit-sequence      =       hexadecimal-digit
                                |       hexadecimal-digit-sequence < hexadecimal-digit ;

floating-suffix                         =       f | l | F | L ;
```

Designated constructs:

DCRN	Definition	Rationale
6.4.4.2-1	A *floating-constant* containing a *floating-suffix* that is **`f`** or **`F`**.	Some users believe that failure to use an explicit suffix for such a *floating-constant* impairs **UNDERSTANDABILITY**.
6.4.4.2-2	A *floating-constant* that: (a) has not resulted from expansion of a macro, and (b) is not an *initializer*, and (c) denotes a value that is neither zero nor one.	Such a *floating-constant* (often called a "magic constant") may represent a configuration parameter. Some users believe that failure to give it a symbolic definition, either as a macro or a value of const-qualified type, impairs **MAINTAINABILITY**.

6.4.4.2-3	A *hexadecimal-floating-constant*.	The use of such constants may impair **PORTABILITY** of code among implementations conforming to earlier versions of the base language standard.

Note: A diagnostic processor may identify constructs similar to DCRN 6.4.4.2-2 such as a *floating-constant* that denotes a value other than zero or one, e.g. two. The values zero and one are excluded from the definition of DCRN 6.4.4.1-2 because most uses of them are not magic numbers.

6.4.4.3 Enumeration constants (NR)

6.4.4.4 Character constants

Orthosyntax:

```
character-constant              =     ` < c-char-sequence < ' ;
                                |     L < ' < c-char-sequence < ' ;

character-constant              =     ' < c-char-sequence < '
                                |     L < ' < c-char-sequence < ' ;

c-char-sequence                 =     c-char
                                |     c-char-sequence < c-char ;

c-char                          =     escape-sequence
                                |     any member of the source character set except the
                                      single-quote ', backslash \, or new-line character ;

escape-sequence                 =     simple-escape-sequence
                                |     octal-escape-sequence
                                |     hexadecimal-escape-sequence
                                |     universal-character-name ;

simple-escape-sequence          =     \' | \" | \? | \\ | \a | \b
                                |     \f | \n | \r | \t | \v  ;

octal-escape-sequence           =     \ < octal-digit
                                |     \ < octal-digit < octal-digit
                                |     \ < octal-digit < octal-digit < octal-digit ;

hexadecimal-escape-sequence     =     \x < hexadecimal-digit
                                |     hexadecimal-escape-sequence < hexadecimal-digit ;
```

Parasyntax:

```
character-constant              =     INTEGER-CHARACTER-CONSTANT
                                |     WIDE-CHARACTER-CONSTANT  ;

INTEGER-CHARACTER-CONSTANT      =     ` < c-char-sequence < ' ;

WIDE-CHARACTER-CONSTANT         =     L < ' < c-char-sequence < ' ;

VALUE-ESCAPE-SEQUENCE           =     escape-sequence
                                &     OCT-OR-HEX-ESCAPE-SEQUENCE ;

OCT-OR-HEX-ESCAPE-SEQUENCE      =     \ < OCTAL-ESC-DIGITS
```

| \ < *HEXADECIMAL-ESC-DIGITS* ;

OCTAL-ESC-DIGITS = *octal-digit*
| *octal-digit* < *octal-digit*
| *octal-digit* < *octal-digit* < *octal-digit* ;

HEXADECIMAL-ESC-DIGITS = *hexadecimal-digit*
| *HEXADECIMAL-ESC-DIGITS* < *hexadecimal-digit* ;

Designated constructs:

DCRN	Definition	Rationale
6.4.4.4-1	A *character-constant* beginning with **L**.	Support for wide characters is **implementation-defined**.
6.4.4.4-2	A *INTEGER-CHARACTER-CONSTANT* that contains more than one *c-char*.	The number of characters permitted in a *character-constant* is **implementation-defined**.
6.4.4.4-3	A non-standard *character-constant*.	Behaviour is **undefined**.
6.4.4.4-4	A non-standard escape sequence.	Support for non-standard escape sequences is **implementation-defined**.
6.4.4.4-5	An *VALUE-ESCAPE-SEQUENCE* that is contained by an *INTEGER-CHARACTER-CONSTANT*and whose *OCTAL-ESC-DIGITS* or *HEXADECIMAL-ESC-DIGITS* denote a value that is outside the range of representable values for the type **unsigned char**.	A **constraint** is violated if the value lies outside the range of the relevant type.
6.4.4.4-6	A *VALUE-ESCAPE-SEQUENCE* that is contained by a *WIDE-CHARACTER-CONSTANT* and whose *OCTAL-ESC-DIGITS* or *HEXADECIMAL-ESC-DIGITS* denote a value that is outside the range of representable values for the type **wchar_t**.	A **constraint** is violated if the value lies outside the range of the relevant type.
6.4.4.4-7	A *character-constant* that has not resulted from expansion of a macro, and is not an *initializer*.	Such a *character-constant* (often called a “magic constant”) may represent a configuration parameter. Some users believe that failure to give it a symbolic definition, either as a macro or as a value of const-qualified type, impairs **MAINTAINABILITY**.

6.4.5 String literals

Orthosyntax:

string-literal = **"** < [*s-char-sequence*] < **"**
| **L"** < [*s-char-sequence*] < **"** ;

s-char-sequence = *s-char*
| *s-char-sequence* < *s-char* ;

s-char = *escape-sequence*
| any member of the source character set except the double-quote **"**, backslash \, or new-line character ;

Parasyntax:

CHARACTER-STRING-LITERAL = `"` < [*s-char-sequence*] < `"` ;

WIDE-STRING-LITERAL = `L"` < [*s-char-sequence*] < `"` ;

Designated constructs:

DCRN	Definition	Rationale
6.4.5-1	A *string-literal* beginning with **`L`**.	Support for wide character strings is **locale-specific**.
6.4.5-2	Adjacent occurrences of a *CHARACTER-STRING-LITERAL* and a *WIDE--STRING-LITERAL*.	Support for wide character strings is **locale-specific**.
6.4.5-3	A *string-literal* containing non-standard escape sequence.	Support for non-standard escape sequences is **unspecified.**
6.4.5-4	A null character that is not the last *s-char* contained in a *string-literal*.	Such occurrences of null characters may lead to unexpected results if the string is an argument to an unbounded string processing functions. Some users therefore consider that they impair **UNDERSTANDABILITY**.
6.4.5-5	A *string-literal* containing a *simple-escape-sequence*.	Some users believe that embedding such escape sequences in strings impairs **UNDERSTANDABILITY**.
6.4.5-6	A *string-literal* appearing in a context such that its stored representation is subject to a modifying access.	The effect of such an access is **undefined**.
6.4.5-7	A *string-literal* that has not resulted from expansion of a macro, and is not an *initializer*.	Such a *string-literal* (often called a “magic constant”) may represent a configuration parameter. If it does, some users believe that failure to give it a symbolic definition, either as a macro or as a value of const-qualified type, impairs **maintainability**.

6.4.6 Punctuators

Orthosyntax:

```
punctuator  =   [ | ] | ( | ) | { | } | . | -> | ++ | -- | & | * | + | -
            |   ~ | ! | / | % | << | >> | < | > | <= | >= | == | ^ | | | &&
            |   || | ? | : | ; | ... | = | *= | /= | %= | += | -= | <<=
            |   >>= | &= | ^= | |= | , | # | ## | <: | :> | <% | %> | %:
            |   %:%: ;
```

Parasyntax:

SUBSTITUTE-PUNCTUATOR = `<: | :> | <% | %> | %: | %:%:` ;

Designated constructs:

DCRN	Definition	Rationale

6.4.6-1	A *SUBSTITUTE-PUNCTUATOR*.	The presence of a *SUBSTITUTE-PUNCTUATOR* may impair **PORTABILITY** among implementations conforming to earlier versions of the base language standard.

6.4.7 Header names

Orthosyntax:

```
header-name          =    < < h-char-sequence < >
                     |    " < q-char-sequence < " ;

h-char-sequence      =    h-char
                     |    h-char-sequence < h-char ;

h-char               =    any member of the source character set
                          except the new-line character and >

q-char-sequence      =    q-char
                     |    q-char-sequence < q-char

q-char               =    any member of the source character set
                          except the new-line character and "
```

Parasyntax:

```
STD-HEADER-NAME           =    < < STD-HU-CHAR-SEQUENCE < > ;

USER-HEADER-NAME          =    " < STD-HU-CHAR-SEQUENCE < " ;

STD-HU-CHAR-SEQUENCE      =    STD-HU-BEFORE-PERIOD < . < LETTER ;

STD-HU-BEFORE-PERIOD      =    STD-HU-CHAR & LETTER
                          |    STD-HU-BEFORE-PERIOD < STD-HU-CHAR ;

STD-HU-CHAR               =    LETTER
                          |    digit ;
```

Designated constructs:

DCRN	Definition	Rationale
6.4.7-1	A *header-name* that is neither a *STD-HEADER-NAME* nor a *USER-HEADER-NAME*.	The mapping from header names to corresponding source file names is **undefined** if non-standard forms of header name are used but is unique (although **implementation-defined**) if a standard form is used.
6.4.7-2	A *STD-HU-CHAR-SEQUENCE* containing more than 8 (C90 = 6) *STD-HU-CHAR*s.	The mapping from header names to corresponding source file names is **undefined** if non-standard forms of header name are used but is unique (although **implementation-defined**) if a standard form is used.
6.4.7-3	A *header-name* whose *h-char-sequence* contains ` , \ , " , // , or /*	T-behaviour is **undefined**.

6.4.7-4	A *header-name* whose *q-char-sequence* contains `` ` `` , `\` , `//` , or `/*`	T-behaviour is **undefined**.
6.4.7-5	A *header-name* that is not contained by an *INCLUDE-DIRECTIVE*.	Behaviour is **undefined**.

Note: Several sub-cases of DCRNs 6.4.7-1 and 6.4.7-2 may be identified. A diagnostic processor may distinguish among them by issuing different diagnostic messages.

6.4.8 Preprocessing numbers

Orthosyntax:

pp-number = *digit*
| `.` < *digit*
| *pp-number* < *digit*
| *pp-number* < *identifier-nondigit*
| *pp-number* < **`e`** < *sign*
| *pp-number* < **`E`** < *sign*
| *pp-number* < **`p`** < *sign*
| *pp-number* < **`P`** < *sign*
| *pp-number* < `.` ;

Parasyntax:

ALL-DIGIT-PP-NUMBER = *digit*
| *ALL-DIGIT-PP-NUMBER* < *digit* ;

Designated constructs:

DCRN	Definition	Rationale
6.4.8-1	An *ALL-DIGIT-PP-NUMBER* that begins with `0` and contains a *nonzero-digit* that is either `8` or `9`.	Such a construct may have been intended to be an *octal-constant* but is very likely to be the result of a programmer's error. Behaviour is **undefined.**
6.4.8-2	A *pp-number* containing **`p`** or **`P`**.	The presence of such a *pp-number* may impair **PORTABILITY** among implementations conforming to earlier versions of the base language standard.

6.4.9 Comments

Designated constructs:

DCRN	Definition	Rationale
6.4.9-1	A comment containing `/*`	Such a construct may be indicative of an attempt to write a nested comment and T-behaviour is **undefined.**
6.4.9-2	The characters `*/` occurring outside a comment.	Such a construct may be indicative of an attempt to write a nested comment and T-behaviour is **undefined.**

6.4.9-3	A comment beginning with the characters `//`.	The presence of such comments may impair **PORTABILITY** among implementations conforming to earlier versions of the base language standard.

6.5 Expressions

Parasyntax:

SIDE-EFFECTIVE-OPERATOR = **++ | -- | == | *= | /= | %= | += |**
-= | <<= | >>= | &= | ^= | |= ;

OLD-STYLE-COMP-ASSGN-OP = **=* | =/ | =% | =+ |=- | =<< | =>> | =& ;**

Designated constructs:

DCRN	Definition	Rationale
6.5-1	An *expression* in which the stored value of an object is accessed by an lvalue that does not have one of the following types: (a) a type compatible with the effective declared type of the object, or (b) a qualified version of a type compatible with the effective type of object, or (c) a type that is the signed or unsigned type corresponding to the effective type of the object, or (d) a type that is the signed or unsigned type corresponding to a qualified version of the effective type of the object, or (e) an aggregate or union type that (recursively) includes one of the aforementioned types among its members, or (f) a character type.	The effect of such an access is **undefined**.
6.5-2	An *expression* whose E-behaviour causes an object to have its stored value modified more than once between sequence points.	The effect of such multiple modifications is **undefined**.
6.5-3	An *expression* whose value is dependent on the order of evaluation of the operands of any *expression* that it contains..	The value of such an expression is **undefined** or **implementation-defined** depending on the expression.
6.5-4	An *expression* in whose E-behaviour an exceptional condition arises.	Subsequent E-behaviour is **undefined**.
6.5-5	An *OLD-STYLE-COMP-ASSGN-OP*	Some pre-standard implementations supported these as alternative ways of writing compound assignment operators but they were not included in C90. Corresponding behaviour under a conforming implementation is **undefined**.
6.5-6	An *expression* containing operators of different precedence without intervening parentheses.	Some users believe that such usage impairs thye **UNDERSTANDABILITY** of code.
6.5-7	An *expression* in which lack of spacing makes the expression difficult to read.	Some users believe that such usage impairs thye **UNDERSTANDABILITY** of code.

6.5.1 Primary expressions

Orthosyntax:

```
primary-expr    =       identifier
                |       constant
                |       string-literal
                |       ( expression )
```

6.5.2 Postfix operators

Orthosyntax:

```
postfix-expression      =       primary-expression
                        |       postfix-expression [ expression ]
                        |       postfix-expression ( [ argument-expression-list ] )
                        |       postfix-expression    identifier
                        |       postfix-expression -> identifier
                        |       postfix-expression ++
                        |       postfix-expression -
                        |       (  type-name  )  {  initializer-list  }
                        |       (  type-name  )  {  initializer-list  ,  }  ;

argument-expression-list:
                assignment-expr
                argument-expression-list , assignment-expr
```

Parasyntax:

```
postfix-expression      =       primary-expr
                        |       SUBSCRIPT-EXPRESSION
                        |       FUNCTION-CALL-EXPRESSION
                        |       DIRECT-ACCESS-EXPRESSION
                        |       INDIRECT-ACCESS-EXPRESSION
                        |       POST-INCREMENT-EXPRESSION
                        |       POST-DECREMENT-EXPRESSION
                        |       COMPOUND-LITERAL  ;

SUBSCRIPT-EXPRESSION            =       postfix-expression [ expression ] ;

FUNCTION-CALL-EXPRESSION        =       FUNCTION-DESIGNATOR
                                                ( [ argument-expression-list ] ) ;

FUNCTION-DESIGNATOR             =       postfix-expression ;

DIRECT-ACCESS-EXPRESSION        =       postfix-expression    identifier ;

INDIRECT-ACCESS-EXPRESSION      =       postfix-expression -> identifier ;

POST-INCREMENT-EXPRESSION       =       postfix-expression ++ ;

POST-DECREMENT-EXPRESSION       =       postfix-expression -- ;

COMPOUND-LITERAL                =       (  type-name  )  {  initializer-list  }
                                |       (  type-name  )  {  initializer-list  ,  }  ;

argument-expression-list        =       ARGUMENT
```

| | | | *argument-expression-list* **,** *ARGUMENT* ;
|---|---|---|
| *ARGUMENT* | = | *assignment-expr* ; |

6.5.2.1 Array subscripting

Designated constructs:

DCRN	Definition	Rationale
6.5.2.1-1	A *SUBSCRIPT-EXPRESSION* whose *postfix-expression* does not have pointer to object type.	Such a construct violates a **constraint**.
6.5.2.1-2	An *SUBSCRIPT-EXPRESSION* whose *expression* does not have integer type.	Such a construct violates a **constraint**.

6.5.2.2 Function calls

Designated constructs:

DCRN	Definition	Rationale
6.5.2.2-1	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* does not have type pointer to function returning **`void`** or returning an object type other than array type.	Such a construct violates a **constraint**.
6.5.2.2-2	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* is not a *PARENTHESISED-IDENTIFIER*.	Other forms of function-designator in this context may render code that contains them less tractable to analysis thus impairing **ANALYSABILITY**.
6.5.2.2-3	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* denotes a function for which the containing *translation-unit* contains no corresponding *FUNCTION-PROTOTYPE*.	The semantics of calls to such functions permit only limited type-checking thus impairing the **ANALYSABILITY** of any translation unit that contains them.
6.5.2.2-4	A *FUNCTION-CALL-EXPRESSION* closest-containing an *ARGUMENT* that denotes a value that is not of object type.	Passing arguments of non-object (i.e. function) type impairs the **ANALYSABILITY** of code.

6.5.2.2-5	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* denotes a function for which the containing *translation-unit* contains a corresponding *FUNCTION-PROTOTYPE* that does not contain `, ...` and whose *argument-expression-list* does not contain exactly as many *ARGUMENT* as there are *declarator* in the *parameter-type-list* of that *FUNCTION-PROTOTYPE*.	The effect of such a function-call is **undefined**.
6.5.2.2-6	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* denotes a function for which the containing *translation-unit* contains a corresponding *FUNCTION-PROTOTYPE* that does contain `, ...` and whose *argument-expression-list* does not contain at least as many *ARGUMENT* as there are *declarator* in the *parameter-type-list* of that *FUNCTION-PROTOTYPE*.	The effect of such a function-call is **undefined**.
6.5.2.2-7	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* denotes a function for which the containing *translation-unit* contains a corresponding *K-AND-R-FUNCTION-DECLARATOR* and whose *argument-expression-list* does not contain exactly as many *ARGUMENT* as there are *identifier* in the *identifier-list* of that *K-AND-R-FUNCTION-DECLARATOR*.	The effect of such a function-call is **undefined**.
6.5.2.2-8	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOr* denotes a function for which the containing *translation unit* contains a corresponding *FUNCTION-PROTOTYPE* and in which the type of each closest-contained *ARGUMENT* is not compatible, after promotion, with the type of the corresponding parameter in the corresponding *FUNCTION-PROTOTYPE*.	The effect of such a function-call is **undefined**.
6.5.2.2-9	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* denotes a function for which the containing *translation unit* contains a corresponding *K-AND-R-FUNCTION-DECLARATOR*.and in which the type of each closest-contained *ARGUMENT* is not compatible, after promotion, with the type of the corresponding parameter in the corresponding *K-AND-R-FUNCTION-DECLARATOR*., unless one of the following is true of the type of the *ARGUMENT* and the type of the parameter: (a) one promoted type is a signed integer type and the other promoted type is the corresponding unsigned integer type, and the value of the argument is representable in both types, or (b) both types are pointers to qualified or unqualified versions of a character type or **`void`**.	The effect of such a function-call is **undefined**.
6.5.2.2-10	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* denotes a function that accepts a variable number of arguments.	The semantics of calls to such functions permit only limited type-checking thus impairing the **ANALYSABILITY** of any translation unit that contains them.

6.5.2.2-11	A *FUNCTION-CALL-EXPRESSION* closest-containing an *ARGUMENT* whose E-behaviour contains a side effect.	The order of evaluation for the *argument-expression-list* is **unspecified**.
6.5.2.2-12	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* denotes the function **`main`**.	Behaviour is **undefined** if the result is a recursive call of **`main`**.
6.5.2.2-13	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* denotes a recursive function.	The amount of memory required to run any possible instance of such a call may not be tractable to determination by static or dynamic analysis, thus impairing **ANALYZABILITY**.
6.5.2.2-14	A *FUNCTION-CALL-EXPRESSION* the E-behaviour of whose *function-designator* contains a side effect.	Some users believe that such usage impairs the **UNDERSTANDABILITY** of code.

Note: Coding manuals for high-integrity applications may prohibit recursive functions outright because it is typically infeasible to predict the maximum amount of memory that they may require at execution time.

6.5.2.3 Structure and union members

Designated constructs:

DCRN	Definition	Rationale
6.5.2.3-1	A *DIRECT-ACCESS-EXPRESSION* whose *postfix-expression* does not have structure or union type.	Such a construct violates a **constraint**.
6.5.2.3-2	An *INDIRECT-ACCESS-EXPRESSION* whose *postfix-expression* does not have structure or union type.	Such a construct violates a **constraint**.
6.5.2.3-3	A *DIRECT-ACCESS-EXPRESSION* whose *identifier* does not denote a member of the structure or union type object of its *postfix-expression*.	Such a construct violates a **constraint**.
6.5.2.3-4	An *INDIRECT-ACCESS-EXPRESSION* whose *identifier* does not denote a member of the structure or union type object of its *postfix-exrpession*.	Such a construct violates a **constraint**.

6.5.2.4 Postfix increment and decrement operators

Designated constructs:

DCRN	Definition	Rationale
6.5.2.4-1	A *POST-INCREMENT-EXPRESSION* whose *postfix-expression* does not have qualified or unqualified real or pointer type or is not a modifiable lvalue.	Such a construct violates a **constraint**.
6.5.2.4-2	A *POST-DECREMENT-EXPRESSION* whose *postfix-expression* does not have qualified or unqualified real or pointer type or is not a modifiable lvalue.	Such a construct violates a **constraint**.

6.5.2.4-3	A *POST-INCREMENT-EXPRESSION* whose *postfix-expression* has enumerated type.	Some users believe that application of increment and decrement operators to values of enumerated types is a common cause of programming errors and view prohibition of such usage as **defensive programming**.
6.5.2.4-4	A *POST-DECREMENT-EXPRESSION* whose *postfix-expression* has enumerated type.	Some users believe that application of increment and decrement operators to values of enumerated types is a common cause of programming errors and view prohibition of such usage as **defensive programming**.
6.5.2.4-5	A *POST-INCREMENT-EXPRESSION* whose *postfix-expression* does not have integer type.	Some users believe that application of increment and decrement operators to values of anything other than integer types is a common cause of programming errors and view prohibition of such usage as **defensive programming**.
6.5.2.4-6	A *POST-INCREMENT-EXPRESSION* whose *postfix-expression* does not have integer type.	Some users believe that application of increment and decrement operators to values of anything other than integer types is a common cause of programming errors and view prohibition of such usage as **defensive programming**.
6.5.2.4-7	A *POST-INCREMENT-EXPRESSION* whose *postfix-expression* is not an *IDENTIFIER*.	Some users believe that application of increment and decrement operators to values of anything other than *expression* that are *identifier* is a common cause of programming errors and view prohibition of such usage as **defensive programming**.
6.5.2.4-8	A *POST-INCREMENT-EXPRESSION* whose *postfix-expression* is not an *IDENTIFIER*.	Some users believe that application of increment and decrement operators to values of anything other than *expression* that are *identifier* is a common cause of programming errors and view prohibition of such usage as **defensive programming**.

6.5.2.5 Compound literals

Designated constructs:

DCRN	**Definition**	**Rationale**
6.5.2.5-1	A *COMPOUND-LITERAL* whose *type-name* specifies neither an object type nor an array of unknown size.	Such a construct violates a **constraint**.
6.5.2.5-2	A *COMPOUND-LITERAL* whose *type-name* specifies a variable length array type.	Such a construct violates a **constraint**.
6.5.2.5-3	An *initializer-list* of a *COMPOUND-LITERAL* that attempts to provide a value for an object not contained within the entire unnamed object specified by the *COMPOUND-LITERAL*.	Such a construct violates a **constraint**.
6.5.2.5-4	A *COMPOUND-LITERAL* that is contained by a *FUNCTION-BLOCK* and whose *initializer-list* contains an *expression* that is not a constant-expression.	Such a construct violates a **constraint**.

6.5.3 Unary operators

Orthosyntax:

unary-expression	=	*postfix-expression*
	|	**++** *unary-expression*
	|	**--** *unary-expression*
	|	*unary-operator cast-expression*
	|	**sizeof** *unary-expression*
	|	**sizeof (** *type-name* **)** ;
unary-operator	=	**&** | ***** | **+** | **-** | **~** | **!** ;

Parasyntax:

unary-expr	=	*postfix-expression*
	|	*PRE-INCREMENT-EXPRESSION*
	|	*PRE-DECREMENT-EXPRESSION*
	|	*UNARY-OP-EXPR*
	|	*SIZEOF-UNARY-EXPR*
	|	*SIZEOF-TYPE-NAME* ;
PRE-INCREMENT-EXPRESSION	=	**++** *unary-expression* ;
PRE-DECREMENT-EXPRESSION	=	**--** *unary-expression* ;
UNARY-OP-EXPR	=	*AMPERSAND-EXPR*
	|	*ASTERISK-EXPR*
	|	*UPLUS-EXPR*
	|	*UMINUS-EXPR*
	|	*TILDE-EXPR*
	|	*SHRIEK-EXPR* ;
SIZEOF-UNARY-EXPR	=	**sizeof** *unary-expression* ;
SIZEOF-TYPE-EXPR	=	**sizeof (** *type-name* **)** ;
AMPERSAND-EXPR	=	**&** *cast-expression* ;
ASTERISK-EXPR	=	***** *cast-expression*;
UPLUS-EXPR	=	**+** *cast-expression* ;
UMINUS-EXPR	=	**-** *cast-expression* ;
TILDE-EXPR	=	**~** *cast-expression* ;
SHRIEK-EXPR	=	**!** *cast-expression* ;

6.5.3.1 Prefix increment and decrement operators

Designated constructs:

DCRN	Definition	Rationale
6.5.3.1-1	A *PRE-INCREMENT-EXPRESSION* whose *unary-expression* does not have qualified or unqualified real or pointer type or is not a modifiable lvalue.	Such a construct violates a **constraint**.
6.5.3.1-2	A *PRE-DECREMENT-EXPRESSION* whose *unary-expression* does not have qualified or unqualified real or pointer type or is not a modifiable lvalue.	Such a construct violates a **constraint**.
6.5.3.1-3	A *PRE-INCREMENT-EXPRESSION* whose *postfix-expression* has enumerated type.	Some users believe that application of increment and decrement operators to values of enumerated types is a common cause of programming errors and view prohibition of such usage as **defensive programming**.
6.5.3.1-4	A *PRE-DECREMENT-EXPRESSION* whose *postfix-expression* has enumerated type.	Some users believe that application of increment and decrement operators to values of enumerated types is a common cause of programming errors and view prohibition of such usage as **defensive programming**.
6.5.2.4-5	A *PRE-INCREMENT-EXPRESSION* whose *postfix-expression* does not have integer type.	Some users believe that application of increment and decrement operators to values of anything other than integer types is a common cause of programming errors and view prohibition of such usage as **defensive programming**.
6.5.2.4-6	A *PRE-INCREMENT-EXPRESSION* whose *postfix-expression* does not have integer type.	Some users believe that application of increment and decrement operators to values of anything other than integer types is a common cause of programming errors and view prohibition of such usage as **defensive programming**.

6.5.3.2 Address and indirection operators

Designated constructs:

DCRN	Definition	Rationale
6.5.3.2-1	An *AMPERSAND-EXPR* whose *cast-expression* is not a *FUNCTION-DESIGNATOR* or whose value is not the result of a *SUBSCRIPT EXPRESSION* or an *ASTERISK-EXPR*, or is an lvalue that designates an object that is bit-field or is declared with the *storage-class-specifier* `register`.	Such a construct violates a **constraint**.
6.5.3.1-2	An *AMPERSAND-EXPR* whose *cast-expression* denotes the function `main`.	Some users believe that there is no useful purpose in taking the address of main and prefer to ban the practice as a rule of **defensive programming**.

6.5.3.1-3	An *ASTERISK-EXPR* whose *cast-expression* does not have pointer type.	Such a construct violates a **constraint**.

6.5.3.3 Unary arithmetic operators

Designated constructs:

DCRN	Definition	Rationale
6.5.3.3-1	A *UPLUS-EXPR* whose *cast-expression* does not have arithmetic type.	Such a construct violates a **constraint**.
6.5.3.3-2	A *UMINUS-EXPR* whose *cast-expression* does not have arithmetic type.	Such a construct violates a **constraint**.
6.5.3.3-3	A *TILDE-EXPR* whose *cast-expression* does not have integer type.	Such a construct violates a **constraint**.
6.5.3.3-4	A *SHRIEK-EXPR* whose *cast-expression* does not have scalar type or is a constant.	Such a construct violates a **constraint**.
6.5.3.3-5	A *TILDE-EXPR* whose *cast-expression* does not have unsigned type.	The result of applying the tilde operator to a signed operand is **unspecified**.
6.5.3.3-6	A *SHRIEK-EXPR* whose *cast-expression* does not have unsigned type.	The result of applying the tilde operator to a signed operand is **unspecified**.
6.5.3.3-7	A *SHRIEK-EXPR* whose *cast-expression* is not an *EXPLICIT-LOGICAL-EXPR*.	Some users believe that it aids **understandability** if logical operators are applied only to expressions that are of ostensively logical form.
6.5.3.3-8	A *UMINUS-EXPR* whose *cast-expression* does not denote a value of a signed type.	The result of a uminus-expr is the negative of its promoted operand. Some users believe that programmers are prone to make errors by misunderstanding the effects of the entailed promotion on an unsigned operand and therefore choose to ban such constructs in aid of **defensive programming**.
6.5.3.3-9	A *UPLUS-EXPR*.	In many cases a *UPLUS-EXPR* can be replaced by its *cast-expression* without altering the effect of the program. Some users consider that the use of redundant constructs impairs **understandability**.

6.5.3.4 The `sizeof` operator

Designated constructs:

DCRN	Definition	Rationale
6.5.3.4-1	A *SIZEOF-UNARY-EXPR* whose *unary-expression* has function type or an incomplete type or that designates a bit-field.	Such a construct violates a **constraint**.
6.5.3.4-2	A *SIZEOF-UNARY-EXPR* whose result exceeds 65535 (C90 = 32787).	Behaviour is **undefined**.
6.5.3.4-3	A *SIZEOF-TYPE-EXPR* whose result exceeds 65535 (C90 = 32787).	Behaviour is **undefined**.

6.5.3.4-4	A *SIZEOF-UNARY-EXPR* whose *unary-expression* contains a *SIDE EFFECTIVE-OPERATOR*.	Since the operand of `sizeof` is evaluated only if it denotes a variable-length array, side effects of any *SIDE-EFFECTIVE-OPERATOR* in its *unary-expression* may not occur. Some users believe that the occurrence of such a *unary-expression* that does contain a *SIDE-EFFECTIVE-OPERATOR* is likely to indicate an error on the part of the programmer. Accordingly they may wish to ban or control such use in aid of **defensive programming**.
6.5.3.4-5	A *SIZEOF-UNARY-EXPR*.	Some users believe that programmers are prone to make errors by misunderstanding the effects of the `sizeof` operator and there fore choose to ban or control such constructs in aid of **defensive programming**.
6.5.3.4-6	A *SIZEOF-TYPE-EXPR*.	Some users believe that programmers are prone to make errors by misunderstanding the effects of the `sizeof` operator and therefore choose to ban of control such constructs in aid of **defensive programming**.

6.5.4 Cast operators

Orthosyntax:

cast-expression = *unary-expression*
| **(** *type-name* **)** *cast-expression* ;

Parasyntax:

cast-expression = *unary-expression*
| *EXPLICIT-CAST-EXPR* ;

EXPLICIT-CAST-EXPR = **(** *type-name* **)** *cast-expression* ;

Designated constructs:

DCRN	Definition	Rationale
6.5.4-1	An *EXPLICIT-CAST-EXPR* whose *type-name* does not specify the `void` type or a qualified or unqualified scalar type.	Such a construct violates a **constraint**.
6.5.4-2	An *EXPLICIT-CAST-EXPR* that converts a value of const-qualified type to a type that is not const-qualified.	**Undefined** behaviour can result.
6.5.4-3	An *EXPLICIT-CAST-EXPR* that converts a value of one type to a type of stricter alignment.	**Undefined** behaviour can result.
6.5.4-4	An *EXPLICIT-CAST-EXPR* that converts a value of one type to another type in which that value is unrepresentable.	The result may have an **unspecified** value.

6.5.4-5	An *EXPLICIT-CAST-EXPR* whose *cast-expression* has pointer type.	Some users believe that programmers are particularly prone to make errors when casting pointer types. Accordingly they may ban or control such usage in aid of **defensive programming**.
6.5.4-6	An *EXPLICIT-CAST-EXPR* whose behaviour converts a value of one type to the same type.	Such a construct is redundant. Some users believe that redundant constructs should be eliminated in aid of **understandability**.

6.5.5 Multiplicative operators

Orthosyntax:

multiplicative-expression	=	*cast-expression*
	|	*multiplicative-expression* ***** *cast-expression*
	|	*multiplicative-expression* **/** *cast-expression*
	|	*multiplicative-expression* % *cast-expression* ;

Parasyntax:

multiplicative-expression	=	*cast-expression*
	|	*EXPLICIT-MULT-EXPR* ;
EXPLICIT-MULT-EXPR	=	*EXPLICIT-TIMES-EXPR*
	|	*EXPLICIT-DIVIDE-EXPR*
	|	*EXPLICIT-MOD-EXPR* ;
EXPLICIT-TIMES-EXPR	=	*multiplicative-expression* ***** *cast-expression* ;
EXPLICIT-DIVIDE-EXPR	=	*multiplicative-expression* **/** *cast-expression* ;
EXPLICIT-MOD-EXPR	=	*multiplicative-expression* % *cast-expression* ;

Designated constructs:

DCRN	Definition	Rationale
6.5.5-1	An *EXPLICIT-MULT-EXPR* whose *cast-expression* or *multiplicative-expression* does not have arithmetic type.	Such a construct violates a **constraint**.
6.5.5-2	An *EXPLICIT-MOD-EXPR* whose *cast-expression* or *multiplicative-expression* does not have integer type.	Such a construct violates a **constraint**.
6.5.5-3	An *EXPLICIT-DIVIDE-EXPR* whose *cast-expression* denotes a numerical value of zero.	The result is **undefined**.
6.5.5-4	An *EXPLICIT-MOD-EXPR* whose *cast-expression* denotes a numerical value of zero.	The result is **undefined**.
6.5.5-5	An *EXPLICIT-MULT-EXPR* either of whose *cast-expression* or *multiplicative-expression* is an *EXPLICIT-LOGICAL-EXPR*.	Some users believe that mixing arithmetic and logical operators in the same expression impairs the **understandability** of code.

6.5.5-6	An *EXPLICIT-MULT-EXPR* either of whose *cast-expression* or *multiplicative-expression* is an *EXPLICIT-BITWISE-EXPR*.	Some users believe that mixing arithmetic and bitwise operators in the same expression impairs the **understandability** of code.

6.5.6 Additive operators

Orthosyntax:

additive-expression = *multiplicative-expression*
| *additive-expression* **+** *multiplicative-expression*
| *additive-expression* **−** *multiplicative-expression* ;

Parasyntax:

additive-expression = *multiplicative-expression*
| *EXPLICIT-ADDITIVE-EXPR* ;

EXPLICIT-ADDITIVE-EXPR = *EXPLICIT-PLUS-EXPR*
| *EXPLICIT-MINUS-EXPR* ;

EXPLICIT-PLUS-EXPR = *additive-expression* **+** *multiplicative-expression* ;

EXPLICIT-MINUS-EXPR = *additive-expression* **−** *multiplicative-expression* ;

Designated constructs:

DCRN	Definition	Rationale
6.5.6-1	An *EXPLICIT-PLUS-EXPR* for which none of the following holds: (a) both its *additive-expression* or *multiplicative-expression* have arithmetic type, or (b) its *additive-expression* has pointer to object type and its *multiplicative-expression* has integer type, or (c) its *multiplicative-expression* has pointer to object type and its *additive-expression* has integer type.	Such a construct violates a **constraint**.
6.5.6-2	An *EXPLICIT-SUB-EXPR* for which none of the following holds: (a) both its *additive-expression* or *multiplicative-expression* have arithmetic type, or (b) both its *additive-expression* or *multiplicative-expression* have qualified or unqualified versions of compatible types, or (c) its additive-expression has pointer to object type and its multiplicative-expression has integer type.	Such a construct violates a **constraint**.

6.5.6-3	An *EXPLICIT-ADDITIVE-EXPR* whose *additive-expression* or *multiplicative-expression* has pointer to object type but points to an object that is not an element of an array.	**Undefined** behaviour may result.
6.5.6-4	An *EXPLICIT-ADDITIVE-EXPR*: (a) whose *additive-expression* (resp. *multiplicative-expression*) has pointer-to object type and points to or one past the last element of an array, and (b) whose *multiplicative-expression* (resp. *additive-expression*) has integer type, and (c) whose result points to an element or one past the last element of the same array, and (d) for which evaluation would produce an overflow	Behaviour is **undefined**.
6.5.6-5	An *EXPLICIT-ADDITIVE-EXPR* that is the *cast-expression* of an *ASTERISK-EXPR* and whose result points one past the last element of an array and is	Behaviour is **undefined**.
6.5.6-6	An *EXPLICIT-SUB-EXPR* whose *additive-expression* and *multiplicative-expression* both have pointer type but do not point to elements of the same array object or one past the last element of the same array object.	Behaviour is **undefined**.
6.5.6-7	An *EXPLICIT-SUB-EXPR* whose *additive-expression* and *multiplicative-expression* both have pointer type but for which the result of the subtraction is not representable in an object of type **`ptrdiff_t`**.	Behaviour is **undefined**.
6.5.6-8	An *EXPLICIT-SUB-EXPR* whose *additive-expression* and *multiplicative-expression* both have pointer type.	The result type, **`ptrdiff_t`** is **implementation-defined**.
6.5.6-9	An *EXPLICIT-ADDITIVE-EXPR* whose *additive-expression* or *multiplicative-expression* denotes a value of pointer type.	The use of pointer arithmetic can in certain circumstances impair the **analyzability** of code. Also some users believe that programmers are prone to make errors when using pointer arithmetic and therefore ban or control such constructs in aid of **defensive programming**.
6.5.6-10	An *EXPLICIT-ADDITIVE-EXPR* either of whose *additive-expression* or *multiplicative-expression* is an *EXPLICIT-LOGICAL-EXPR*.	Some users believe that mixing arithmetic and logical operators in the same expression impairs the **understandability** of code.
6.5.6-11	An *EXPLICIT-ADDITIVE-EXPR* either of whose *additive-expression* or *multiplicative-expression* is an *EXPLICIT-BITWISE-EXPR*.	Some users believe that mixing arithmetic and bitwise operators in the same expression impairs the **understandability** of code.

6.5.7 Bitwise shift operators

Orthosyntax:

shift-expression = *additive-expression*
| *shift-expression* **<<** *additive-expression*
| *shift-expression* **>>** *additive-expression* ;

Parasyntax:

shift-expression = *additive-expression*
| *EXPLICIT-SHIFT-EXPR* ;

EXPLICIT-SHIFT-EXPR = *EXPLICIT-LSHIFT-EXPR*
| *EXPLICIT-RSHIFT-EXPR* ;

EXPLICIT-LSHIFT-EXPR = *shift-expression* **<<** *additive-expression ;*

EXPLICIT-RSHIFT-EXPR = *shift-expression* **>>** *additive-expression* ;

Designated constructs:

DCRN	Definition	Rationale
6.5.7-1	An *EXPLICIT-SHIFT-EXPR* whose *shift-expression* or *additive-expression* does not have integer type.	Such a construct violates a **constraint**.
6.5.7-2	An *EXPLICIT-SHIFT-EXPR* whose *additive-expression* denotes a negative value.	Behaviour is **undefined**.
6.5.7-3	An *EXPLICIT-SHIFT-EXPR* whose *additive-expression* denotes a value greater than or equal to the width of the promoted value of its *shift-expression*.	Behaviour is **undefined**.
6.5.7-4	An *EXPLICIT-LSHIFT-EXPR* whose *shift-expression* has a signed type and whose result is not representable in its result type.	Behaviour is **undefined**.
6.5.7-5	An *EXPLICIT-RSHIFT-EXPR* whose *shift-expression* has a signed type and denotes a negative value.	The result is **implementation-defined**.
6.5.7-6	An *EXPLICIT-SHIFT-EXPR* whose *shift-expression* does not have unsigned type.	Many users favour restriction of the *shift-expression* to unsigned type as a simple way to avoid both the **undefined** and **implementation-defined** behaviour that might otherwise result.

6.5.8 Relational operators

Orthosyntax:

relational-expr = *shift-expr*
| *relational-expr* **<** *shift-expr*
| *relational-expr* **>** *shift-expr*
| *relational-expr* **<=** *shift-expr*
| *relational-expr* **>=** *shift-expr* ;

Parasyntax:

relational-expression = *shift-expression*
| *EXPLICIT-REL-EXPR* ;

EXPLICIT-REL-EXPR = *EXPLICIT-LT-EXPR*
| *EXPLICIT-GT_EXPR*
| *EXPLICIT-LE-EXPR*
| *EXPLICIT-GE-EXPR* ;

EXPLICIT-LT-EXPR = *relational-expression* **<** *shift-expression* ;

EXPLICIT-GT_EXPR = *relational-expression* **>** *shift-expression* ;

EXPLICIT-LE-EXPR = *relational-expression* **<=** *shift-expression* ;

EXPLICIT-GE-EXPR = *relational-expression* **>=** *shift-expression* ;

Designated constructs:

DCRN	Definition	Rationale
6.5.8-1	An *EXPLICIT-REL-EXPR* for which none of the following holds: (a) both its *relational-expression* or *shift-expression* have real type, (b) both its *relational-expression* or *shift-expression* have pointer types that are pointers to qualified or unqualified version of compatible object types, (c) both its *relational-expression* or *shift-expression* have pointer types that are pointers to qualified or unqualified version of incomplete types.	Such a construct violates a **constraint**.
6.5.8-2	An *EXPLICIT-REL-EXPR* whose *relational-expression* and *shift-expression* both have pointer type but do not both point to the same object or both point one past the last element of the same array object,	Behaviour is **undefined**.
6.5.8-3	An *EXPLICIT-REL-EXPR* whose *relational-expression* or *shift-expression* is an *EXPLICIT-LOGICAL-EXPR*.	Some users believe that mixing relational and logical operators in the same expression impairs the **UNDERSTANDABILITY** of code.
6.5.8-4	An *EXPLICIT-LT-EXPRESSION* whose *shift-expression* denotes a non-negative value and whose *relational-expression* denotes a value of unsigned type.	Such an expression always evaluates to 0 and is likely to be the result of a programming error that may in turn impair the **FUNCTIONALITY** of the code.

6.5.8-5	An *EXPLICIT-REL-EXPR* whose *relational-expression* or *shift-expression* is a *string-literal*.	Some users believe that programmers are prone to make errors using such constructs (mistakenly believing that they gives lexicographical comparison of the strings themselves) and may wish to ban on control them in aid of **defensive programming**.

6.5.9 Equality operators

Orthosyntax:

equality-expression = *relational-expression*
| *equality-expression* **==** *relational-expression*
| *equality-expression* **!=** *relational-expression* ;

Parasyntax:

equality-expression = *relational-expression*
| *EXPLICIT-EQUALITY-EXPR* ;

EXPLICIT-EQUALITY-EXPR | *equality-expression* **==** *relational-expression*
| *equality-expression* **!=** *relational-expression* ;

Designated constructs:

DCRN	Definition	Rationale
6.5.9-1	An *EXPLICIT-EQUALITY-EXPR* for which none of the following holds: (a) its *equality-expression* and *relational-expression* both have arithmetic type, (b) its *equality-expression* and *relational-expression* both have pointer types that are qualified on unqualified versions of compatible types, (c) its *equality-expression* (resp. *relational expression*) denotes a pointer to an object or incomplete type and its *relational-expression* (resp. *equality-expression*) denotes a pointer to a qualified of unqualified version of void. (d) its *equality-expression* (resp. *relational expressio*n) has pointer type and its *relational-expression* (resp. *equality-expression*) denotes a null pointer constant.	Such a construct violates a **constraint**.

6.5.9-2	An *EXPLICIT-EQUALITY-EXPR* whose *equality-expression* and *relational-expression* are such that both have arithmetic types but none of the following holds: (a) both have integer types, (b) both have floating types, (c) both have real types (d) both have imaginary types, *(e)* both have complex types.	Some users believe that programmers are prone to make errors when using equality operators whose operands have different kinds of arithmetic type; accordingly they may wish to ban or control such usage in aid of **defensive programming**.
6.5.9-3	An *EXPLICIT-EQUALITY-EXPR* whose *equality-expression* or *relational-expression* denotes a value of a floating type.	Exact comparison of values of floating type is a well known cause of error in numerical computations and may impair the **FUNCTIONALITY** of code.

6.5.10 Bitwise AND operator

Orthosyntax:

AND-expression = *equality-expression*
| *AND-expression* **&** *equality-expression* ;

Parasyntax:

AND-expression = *equality-expression*
| *EXPLICIT-AND-EXPR* ;

EXPLICIT-AND-EXPR | *AND-expression* **&** *equality-expression* ;

Designated constructs:

DCRN	Definition	Rationale
6.5.10-1	An *EXPLICIT-AND-EXPR* whose *and-expression* and *equality-expression* do not both have integer type.	Such a construct violates a **constraint**.
6.5.10-2	An *EXPLICIT-AND-EXPR* whose *and-expression* and *equality-expression* does not both have unsigned type.	Some users believe that programmers are prone to make errors when using bitwise operators with signed operands; accordingly they may ban or control such usage in aid of **defensive programming**.

6.5.11 Bitwise exclusive OR operator

Orthosyntax:

exclusive-OR-expression = *AND-expression*
| *exclusive-OR-expression* **^** *AND-expression* ;

Parasyntax:

exclusive-OR-expression = *AND-expression*
| *EXPLICIT-XOR-EXPR* ;

EXPLICIT-XOR-EXPR | *exclusive-OR-expression* **^** *AND-expression* ;

Designated constructs:

DCRN	Definition	Rationale
6.5.11-1	An *EXPLICIT-XOR-EXPR* whose *exclusive-or-expression* and *AND-expression* does not both have integer type.	Such a construct violates a **constraint**.
6.5.11-2	An *EXPLICIT-XOR-EXPR* ewhose *exclusive-or-expression* or *AND-expression* does not both have unsigned type.	Some users believe that programmers are prone to make errors when using bitwise operators with signed operands; accordingly they may ban or control such usage in aid of **defensive programming**.

6.5.12 Bitwise inclusive OR operator

Orthosyntax:

inclusive-OR-expression = *exclusive-OR-expression*
| *inclusive-OR-expression* **|** *exclusive-OR-expression* ;

Parasyntax:

inclusive-OR-expression = *exclusive-OR-expression*
| *EXPLICIT-IOR-EXPR* ;

EXPLICIT-IOR-EXPR | *inclusive-OR-expression* **|** *exclusive-OR-expression* ;

Designated constructs:

DCRN	Definition	Rationale
6.5.12-1	An *EXPLICIT-IOR-EXPR* whose *inclusive-OR-expression* or *exclusive-OR-expression* do not both have integer type.	Such a construct violates a **constraint**.
6.5.12-2	An *EXPLICIT-IOR-EXPR* whose *inclusive-OR-expression* or *exclusive-OR-expression* do not both have unsigned type.	Some users believe that programmers are prone to make errors when using bitwise operators with signed operands; accordingly they may ban or control such usage in aid of **defensive programming**.

6.5.13 Logical AND operator

Orthosyntax:

logical-AND-expression = *inclusive-OR-expression*
| *logical-AND-expression* **&&** *inclusive-OR-expression* ;

Parasyntax:

logical-AND-expression = *inclusive-OR-expression*
| *EXPLICIT-LAND-EXPR* ;

EXPLICIT-LAND-EXPR | *logical-AND-expression* **&&** *inclusive-OR-expression* ;

Designated constructs:

DCRN	Definition	Rationale
6.5.13-1	An *EXPLICIT-LAND-EXPR* whose *logical-AND-expression* and *inclusive-OR-expression* do not both have scalar type.	Such a construct violates a **constraint**.
6.5.13-2	An *EXPLICIT-LAND-EXPR* whose *inclusive-OR-expression* contains a *SIDE-EFFECTIVE-OPERATOR*.	The *inclusive-OR-expression* is evaluated only if the *logical-AND-expression* yields true. Some users believe that programmers are prone to forget this partial evaluation and hence make errors if they use DC 6.5.13-2. Accordingly, they may wish to ban or control it in aid of **defensive programming**.
6.5.13-3	An *EXPLICIT-LAND-EXPR* whose *logical-AND-expression* and *inclusive-OR-expression* are not both *EXPLICIT-LOGICAL-EXPR*.	Some users believe that combining logical and non-logical operators in an expression impairs **UNDERSTANDABILITY**.

6.5.14 Logical OR operator

Orthosyntax:

logical-OR-expression = *logical-AND-expression*
| *logical-OR-expression* **||** *logical-AND-expression*

Parasyntax:

logical-OR-expression = *logical-AND-expression*
| *EXPLICIT-LOR-EXPR* ;

EXPLICIT-LOR-EXPR = *logical-OR-expression* **||** *logical-AND-expression* ;

Designated constructs:

DCRN	Definition	Rationale
6.5.14-1	An *EXPLICIT-LOR-EXPR* whose *logical-OR-expression* and *logical-AND-expression* do not have scalar type.	Such a construct violates a **constraint**.
6.5.14-2	An *EXPLICIT-LOR-EXPR* the behaviour of whose *logical-AND-expression* contains a side effect.	The *logical-AND-expression* is evaluated only if the *logical-OR-expression* yields false. Some users believe that programmers are prone to forget this partial evaluation and hence make errors if they use the DC. Accordingly, they may wish to ban or control it in aid of **defensive programming**.

6.5.14-3	An *EXPLICIT-LOR-EXPR* whose *logical-OR-expression* and *logical-AND-expression* are not both *EXPLICIT-LOGICAL-EXPR*.	Some users believe that combining logical and non-logical operators in an expression impairs **UNDERSTANDABILITY**.

6.5.15 Conditional operator

Orthosyntax:

conditional-expression = *logical-OR-expression*
| *logical-OR-expression*
? *expression*
: *conditional-expression* ;

Parasyntax:

conditional-expression = *logical-OR-expression*
| *EXPLICIT-COND-EXPR* ;

EXPLICIT-COND-EXPR = *logical-OR-expression*
? *expression*
: *conditional-expression* ;

Designated constructs:

DCRN	Definition	Rationale
6.5.15-1	An *EXPLICIT-COND-EXPR* whose *logical-OR-expression* does not have scalar type.	Such a construct violates a **constraint**.
6.5.15-2	An *EXPLICIT-COND-EXPR* for whose *expression* and *conditional-expression* none of the following holds: (a) both have arithmetic type, (b) both have the same structure or union type, (c) both have **`void`** type, (d) both have pointer type and point to qualified or unqualified versions of compatible types, (e) one has pointer type and the other is a null pointer constant (f) one has pointer type and points to an object or incomplete type and the other has pointer type and points to a qualified or unqualified version of **`void`**.	Such a construct violates a **constraint**.
6.5.15-3	An *EXPLICIT-COND-EXPR* whose *logical-OR-expression* has a pointer type.	In certain circumstances the use of pointer types impairs the **ANALYSABILITY** of code.
6.5.15-4	An *EXPLICIT-COND-EXPR* whose *expression* and *conditional-expression* do note denote values of the same type.	Some users believe that when the *expression* and *conditional-expression* have different types this impairs the **UNDERSTANDABILITY** of code.

6.5.15-5	An *EXPLICIT-COND-EXPR* either of whose *expression* or *conditional-expression* contains a *SIDE-EFFECTIVE-OPERATOR*.	Some users believe that side effects in the *expression* or *conditional-expression* impair the **UNDERSTANDABILITY** of code.

Note: Banning DC 6.5.15-3 removes the risk that the result of an *EXPLICIT-COND-EXPR* may be modified or accessed after the next sequence point, thereby resulting in **undefined** behaviour.

6.5.16 Assignment operator

Orthosyntax:

assignment-expression	=	*conditional-expression*
	|	*unary-expression assignment-operator*
assignment-expression ;		
assignment-operator	=	**=** | ***=** | **/=** | **%=** | **+=** | **-=**
	|	**<<=** | **>>=** | **&=** | **^=** | **|=** ;

Parasyntax:

assignment-expression	=	*conditional-expression*
	|	*EXPLICIT-ASSIGN-EXPR* ;
EXPLICIT-ASSIGN-EXPR	=	*EXPLICIT-SIMPLE-ASSIGN-EXPR*
	|	*EXPLICIT-MULT-ASSIGN-EXPR*
	|	*EXPLICIT-DIVIDE-ASSIGN-EXPR*
	|	*EXPLICIT-MOD-ASSIGN-EXPR*
	|	*EXPLICIT-PLUS-ASSIGN-EXPR*
	|	*EXPLICIT-MINUS-ASSIGN-EXPR*
	|	*EXPLICIT-LSHIFT-ASSIGN-EXPR*
	|	*EXPLICIT-RSHIFT-ASSIGN-EXPR*
	|	*EXPLICIT-BITWISE-ASSIGN-EXPR* ;
EXPLICIT-SIMPLE-ASSIGN-EXPR	=	*unary-expression* **=** *assignment-expression* ;
EXPLICIT-MULT-ASSIGN-EXPR	=	*unary-expression* ***=** *assignment-expression* ;
EXPLICIT-DIVIDE-ASSIGN-EXPR	=	*unary-expression* **/=** *assignment-expression* ;
EXPLICIT-MOD-ASSIGN-EXPR	=	*unary-expression* **%=** *assignment-expression* ;
EXPLICIT-PLUS-ASSIGN-EXPR	=	*unary-expression* **+=** *assignment-expression* ;
EXPLICIT-MINUS-ASSIGN-EXPR	=	*unary-expression* **-=** *assignment-expression* ;
EXPLICIT-SHIFT-ASSIGN-EXPR	=	*EXPLICIT-LSHIFT-ASSIGN-EXPR*
	|	*EXPLICIT-RSHIFT-ASSIGN-EXPR* ;
EXPLICIT-LSHIFT-ASSIGN-EXPR	=	*unary-expression* **<<=** *assignment-expression* ;
EXPLICIT-RSHIFT-ASSIGN-EXPR	=	*unary-expression* **>>=** *assignment-expression* ;
EXPLICIT-BITWISE-ASSIGN-EXPR	=	*EXPLICIT-AND-ASSIGN-EXPR*
	|	*EXPLICIT-XOR-ASSIGN-EXPR*
	|	*EXPLICIT-IOR-ASSIGN-EXPR* ;

EXPLICIT-AND-ASSIGN-EXPR = *unary-expression* &= *assignment-expression* ;

EXPLICIT-XOR-ASSIGN-EXPR = *unary-expression* ^= *assignment-expression* ;

EXPLICIT-IOR-ASSIGN-EXPR = *unary-expression* |= *assignment-expression* ;

Expanded forms:

EXPLICIT-MULT-ASSIGN-EXPR(α)
=
unary-expression(β) *= *assignment-expression*(γ)
:
expand(α) = β = β * γ ;

EXPLICIT-DIVIDE-ASSIGN-EXPR(α)
=
unary-expression(β) /= *assignment-expression*(γ)
:
expand(α) = β = β / γ ;

EXPLICIT-MOD-ASSIGN-EXPR(α)
=
unary-expression(β) %= *assignment-expression*(γ)
:
expand(α) = β = β % γ ;

EXPLICIT-PLUS-ASSIGN-EXPR(α)
=
unary-expression(β) += *assignment-expression*(γ)
:
expand(α) = β = β + γ ;

EXPLICIT-MINUS-ASSIGN-EXPR(α)
=
unary-expression(β) −= *assignment-expression*(γ)
:
expand(α) = β = β − γ ;

EXPLICIT-LSHIFT-ASSIGN-EXPR(α)
=
unary-expression(β) <<= *assignment-expression*(γ)
:
expand(α) = β = β << γ ;

EXPLICIT-RSHIFT-ASSIGN-EXPR(α)
=

unary-expression(β) **>>=** *assignment-expression*(γ)

:

expand(α) = β **=** β **>>** γ ;

EXPLICIT-AND-ASSIGN-EXPR(α)

=

unary-expression(β) **&=** *assignment-expression*(γ)

:

expand(α) = β **=** β **&** γ ;

EXPLICIT-XOR-ASSIGN-EXPR(α)

=

unary-expression(β) **^=** *assignment-expression*(γ)

:

expand(α) = β **=** β **^** γ ;

EXPLICIT-IOR-ASSIGN-EXPR(α)

=

unary-expression(β) **|=** *assignment-expression*(γ)

:

expand(α) = β **=** β **|** γ ;

Designated constructs:

DCRN	Definition	Rationale
6.5.16-1	An *EXPLICIT-ASSIGN-EXPR* whose *unary-expression* does not denote a modifiable lvalue..	Such a construct violates a **constraint**.
6.5.16-2	An *EXPLICIT-ASSIGN-EXPR* that is any of the following: *(a)* the postfix-expression of a *POST-INCREMENT-EXPRESSION* or a *POST-DECREMENT-EXPRESSION,* (b) the *unary-expression* of a *PRE-INCREMENT-EXPRESSION or a PRE-DECREMENT-EXPRESSION*.	Since such a construct would attempt to modify the result of an *EXPLICIT-ASSIGN-EXPR*, the behaviour is **undefined**.
6.5.16-3	An *EXPLICIT-ASSIGN-EXPR* that is not an *EXPLICIT-SHIFT-ASSIGN-EXPR* and whose *unary-expression* and *assignment-expression* do not have identical types.	Some users believe that programmers are prone to make errors if they mix different types in assignment expressions. Accordingly they may wish to ban or control such usage in aid of **defensive programming**.

6.5.16.1 Simple assignment (NR)

Designated constructs:

DCRN	Definition	Rationale

6.5.16.1-1	An *EXPLICIT-SIMPLE-ASSIGN-EXPR* for which none of the following holds: (a) its *unary-expression* has qualified or unqualified arithmetic type and its *assignment-expression* has arithmetic type, (b) its *unary-expression* has a qualified or unqualified version of a structure or union type compatible with the type of its *assignment-expression*, *(c)* both its *unary-expression* and its *assignment expression* have pointer types that point to qualified or unqualified versions of compatible types and the type pointed to by the *unary-expression* has all the qualifiers of the type pointed to by the *assignment-expression,* (d) its *unary-expression* (resp. *assignment-expression*) has a pointer type that points to an object or incomplete type and its *assignment-expression* (resp. *unary-expression*) has a pointer type that points to a qualified or unqualified version of **`void`**, and the type pointed to by its *unary-expression* has all the qualifiers of the type pointed to by its *assignment-expression*, (e) its *unary-expression* has pointer type and its *assignment-expression* is a null pointer constant, (f) its *unary-expression* has type **`_Bool`** and its *assignment-expression* has pointer type.	Such a construct violates a **constraint**.
6.5.16.1-2	An *EXPLICIT-SIMPLE-ASSIGN-EXPR* such that both of the following hold: (a) both its *unary-expression* and its *assignment-expression* have qualified or unqualified version of compatible types, and (b) the lvalue of its *unary-expression* refers to an object part but not all of which is accessed by its *assignment-expression*.	Behaviour is **undefined**

6.5.16.2 Compound assignment

Designated constructs:

DCRN	Definition	Rationale

6.5.16.2-1	An *EXPLICIT-PLUS-ASSIGN-EXPR* for which none of the following holds: (a) its *unary-expression* has a pointer to object type and its *assignment-expression* has integer type, (b) its *unary-expression* has qualified or unqualified arithmetic type and its *assignment-expression* has arithmetic type.	Such a construct violates a **constraint**.
6.5.16.2-2	An *EXPLICIT-MINUS-ASSIGN-EXPR* for which none of the following holds: (c) its *unary-expression* has a pointer to object type and its *assignment-expression* has integer type, (d) its *unary-expression* has qualified or unqualified arithmetic type and its *assignment-expression* has arithmetic type.	Such a construct violates a **constraint**.
6.5.16.2-3	An *EXPLICIT-MULT-ASSIGN-EXPR* α such that **expand**(α) contains any of the following DCs: **6.5.5-1, 6.5.5-2, 6.5.5-3, 6.5.5-4, 6.5.5-5, 6.5.5-6**	Reasons as for listed DCs respectively.
6.5.16.2-4	An *EXPLICIT-DIVIDE-ASSIGN-EXPR* α such that **expand**(α) contains any of the following DCs: **6.5.5-1, 6.5.5-2, 6.5.5-3, 6.5.5-4, 6.5.5-5, 6.5.5-6**	Reasons as for listed DCs respectively.
6.5.16.2-5	An *EXPLICIT-MOD-ASSIGN-EXPR* α such that **expand**(α) contains any of the following DCs: **6.5.5-1, 6.5.5-2, 6.5.5-3, 6.5.5-4, 6.5.5-5, 6.5.5-6**	Reasons as for listed DCs respectively.
6.5.16.2-6	An *EXPLICIT-LSHIFT-ASSIGN-EXPR* α such that **expand**(α) contains any of the following DCs: **6.5.7-1, 6.5.7-2, 6.5.7-3, 6.5.7-4, 6.5.7-5, 6.5.7-6**	Reasons as for listed DCs respectively.
6.5.16.2-7	An *EXPLICIT-RSHIFT-ASSIGN-EXPR* α such that **expand**(α) contains any of the following DCs: **6.5.7-1, 6.5.7-2, 6.5.7-3, 6.5.7-4, 6.5.7-5, 6.5.7-6**	Reasons as for listed DCs respectively.
6.5.16.2-8	An *EXPLICIT-AND-ASSIGN-EXPR* α such that **expand**(α) contains any of the following DCs: **6.5.10-1, 6.5.10-2**	Reasons as for listed DCs respectively.
6.5.16.2-9	An *EXPLICIT-XOR-ASSIGN-EXPR* α such that **expand**(α) contains any of the following DCs: **6.5.11-1, 6.5.11-2**	Reasons as for listed DCs respectively.
6.5.16.2-10	An *EXPLICIT-IOR-ASSIGN-EXPR* α such that **expand**(α) contains any of the following DCs: **6.5.12-1, 6.5.12-2**	Reasons as for listed DCs respectively.
6.5.16.2-11	An *EXPLICIT-PLUS-ASSIGN-EXPR* whose *unary-expression* does not have the lvalue of an object of integer type.	Some users believe that confining the use of these expression to integer operands promotes the **UNDERSTANDABILITY** of code.

6.5.16.2-12	An *EXPLICIT-MINUS-ASSIGN-EXPR* whose *unary-expression* does not have the lvalue of an object of integer type.	Some users believe that confining the use of these expression to integer operands promotes the **UNDERSTANDABILITY** of code.

6.5.17 Comma operator

Orthosyntax:

```
comma-expression          =     assignment-expression
                          |     expression , assignment-expression ;
```

Parasyntax:

```
comma-expression          =     assignment-expression
                          |     EXPLICIT-COMMA-EXPR ;

EXPLICIT-COMMA-EXPR       =     expression , assignment-expression ;
```

Designated constructs:

DCRN	Definition	Rationale
6.5.17-1	An *EXPLICIT-COMMA-EXPR*.	Some user believe that programmers are prone to make errors when using a *comma-expression* and may wish to ban or control such usage in aid of **defensive programming**.
6.5.17-2	An *EXPLICIT-COMMA-EXPR* that is any of the following: (a) the postfix-expression of a *POST-INCREMENT-EXPRESSION* or a *POST-DECREMENT-EXPRESSION,* (b) the *unary-expression* of a *PRE-INCREMENT-EXPRESSION or a PRE-DECREMENT-EXPRESSION*.	Since such a construct would attempt to modify the result of an *EXPLICIT-COMMA-EXPRESSION*, the behaviour is **undefined**.
6.5.17-3	An *expression* of an *EXPLICIT-COMMA-EXPRESSION* the E-behaviour for whose *expression* has no side-effect.	Since the *expression* has no side effect, it is redundant and the *EXPLICIT-COMMA-EXPR* may be replaced by its *assignment-expression* without effect on the behaviour of the program. Some users believe that elimination of such redundant usage promotes the **UNDERSTANDBILITY** of code.

6.6 Constant expressions

Orthosyntax:

```
constant-expression    =    conditional-expression ;
```

Designated constructs:

DCRN	Definition	Rationale
6.6-1	A *constant-expression* that is not the *unary-expression* of a *SIZEOF-UNARY-EXPR* but that contains any of the following: (a) a *SIDE-EFFECTIVE-OPERATOR*, or (b) a *FUNCTION-CALL-EXPRESSION*, or (c) an *EXPLICIT-COMMA-EXPRESSION*.	Such a construct violates a **constraint**.
6.6-2	A *constant-expression* denoting a value that is not in the range of representable values for its type.	Such a construct violates a **constraint**.

6.7 Declarations

Orthosyntax:

declaration	=	*declaration-specifiers* [*init-declarator-list*] ;
declaration-specifiers	=	*storage-class-specifier* [*declaration-specifiers*]
	|	*type-specifier* [*declaration-specifiers*]
	|	*type-qualifier* [*declaration-specifiers*]
	|	*function-specifier* [*declaration-specifiers*] ;
init-declarator-list	=	*init-declarator*
	|	*init-declarator-list* **,** *init-declarator* ;
init-declarator	=	*declarator*
	|	*declarator* **=** *initializer* ;

Designated constructs:

DCRN	Definition	Rationale
6.7-1	A *declaration* that does not contain an *init-declarator-list*.	Such a construct violates a **constraint**.
6.7-2	A *declaration* of an *identifier* with no linkage where that *declaration* is in the same scope as another declaration of the same *identifier* in the same name space, unless the *identifier* is a tag.	Such a construct violates a **constraint**.
6.7-3	A *declaration* of an *identifier* where that *declaration* in the same scope as another declaration of the same *identifier* in the same name space but the two *declaration* specify types that are not compatible..	Such a construct violates a **constraint**.
6.7-4	A *declaration-specifiers* that contains more than one *storage-class-specifier*.	Such a construct violates a **constraint**.
6.7-5	A *declaration* whose *declaration-specifiers* contain a *function-specifier* but that does not declare an *identifier* for a function.	Such a construct violates a **constraint**.
6.7-6	A *declaration* for which all of the following hold: • its *declaration-specifiers* contain a *storage-class-specifier* other than **`extern`**, and • it declares an *identifier* for a function, • the declared *identifier* has block scope.	Such a construct violates a **constraint**.
6.7-7	A *declaration* whose *declaration-specifiers* contain more than one *STANDARD-TYPE-SPECIFIER-LIST*.	Such a construct may violate a **constraint**.
6.7-8	A *declaration* of an *identifier* such that its type is not complete by the end of the *init-declarator* in which it occurs.	Behaviour is **undefined**.

6.7-9	A *declaration* that declares an object with incomplete type and no linkage.	Behaviour is **undefined**.
6.7-10	An *init-declarator* that does not contain an *initializer*.	Initialization at the point of declaration eliminates the risk of accessing an object whose value is undefined. Some users believe that this practice promotes **RELIABILITY**.
6.7-11	An *init-declarator-list* that has more than one *init-declarator*.	Some users find it convenient to declare one object or function per declaration, thus enabling the line number of the declaration to serve as a an additional means of identifying the object. Insofar as this facilitates easier configuration management, such a practice may promote **MAINTAINABILITY**.
6.7-12	A *declaration* whose *declaration-specifiers* specify the plain `char` type.	It is **implementation-defined** whether plain `char` is a signed or an unsigned type.
6.7-13	A *declaration* whose *declaration-specifiers* specify an extended integer type.	Such types may not be supported by implementations conforming to earlier version of the base language standard and their use impairs **PORTABILITY**.
6.7-14	A *declaration* that is contained in a *BLOCK* and whose *declaration-specifiers* contain the *storage-class-specifier* `typedef`.	Behaviour for such a construct is undefined for implementations conforming to earlier versions of the base language standard, thus imparing **PORTABILITY**.
6.7-15	A *declaration* that is contained in a *BLOCK* and whose *declaration-specifiers* contain the *storage-class-specifier* `extern`.	Behaviour for such a construct is undefined for implementations conforming to earlier versions of the base language standard, thus imparing **PORTABILITY**.
6.7-16	A *declaration* whose *declaration-specifiers* have no *type-specifier*.	When no *type-specifier* is given, the type defaults to `int`. Some users believe that failure to state the type explicitly impairs the **UNDERSTANDABILITY** of code.
6.7-17	A *declaration* whose *declaration-specifiers* specify a floating type.	Some users consider it prudent to ban the use of floating types in critical applications, believing such a ban to promote **RELIABILITY**.
6.7-18	A *declaration-specifiers* containing more than one occurrence of the same *type-qualifier*.	Repetition of a type-qualifier is redundant. Some users believe that elimination of such redundancy promotes the **UNDERSTANDABILITY** of code.
6.7-19	A source file containing a function declaration with the storage class specifier `static` but no definition for the declared function.	Use of such a construct leaves the function without a definition. This is so often a programming error that some users may wish to ban or control it in aid of **defensive programming**.
6.7-20	A source line containing more than one *declaration*.	Some users believe that programmers are prone to make errors when amending declarations if there are more than one per line and may wish to ban or control them in aid of **MAINTAINABILITY**.

6.7.1 Storage-class specifiers

Orthosyntax:

```
storage-class-specifier    =    typedef
                           |    extern
                           |    static
                           |    auto
                           |    register ;
```

Designated constructs:

DCRN	Definition	Rationale
6.7.1-1	A non-standard *storage-class-specifier*.	The semantics of such constructs are **undefined**.
6.7.1-2	The *storage-class-specifier* `register`.	The extent to which a translator takes any notice of `register` is implementation-defined. Hence, some users believe that any *function-specifier* is misleading and impair the **UNDERSTANDABILITY** of code.
6.71-3	The *storage-class-specifier* `auto`.	There is a widespread convention of not using this *storage-class-specifier* and some users consider that using it impairs the **UNDERSTANDABILITY** of code.

6.7.2 Type specifiers

Orthosyntax:

```
type-specifier    =    void
                  |    char
                  |    short
                  |    int
                  |    long
                  |    float
                  |    double
                  |    signed
                  |    unsigned
                  |    _Bool
                  |    _Complex
                  |    _Imaginary
                  |    struct-or-union-specifier
                  |    enum-specifier
                  |    typedef-name ;
```

Parasyntax:

```
STANDARD-TYPE-SPECIFIER-LIST
      =    void                    |    char
      |    signed char             |    unsigned char
      |    short                   |    signed short
      |    short int               |    signed short int
      |    unsigned short          |    unsigned short int
      |    int                     |    signed
|     signed int               |     unsigned
|     unsigned int             |     long
|     signed long              |     long int
|     signed long int          |     unsigned long
|     unsigned long int        |     long long
|     long long int            |     signed long long
|     signed long long int     |     unsigned long long
|     unsigned long long int
|     float
|     double                   |     long double
|     float _Complex           |     float _Imaginary
|     double _Complex          |     double _Imaginary
|     long double _Complex     |     long double _Imaginary
|     _Bool
|     struct-or-union-specifier
|     enum-specifier
|     typedef-name ;
```

Designated constructs:

DCRN	Definition	Rationale
6.7.2-1	A *type-specifier* that is an *enum-specifier*.	The integral type used to represent an enumerated type is **implementation-defined**.
6.7.2-2	A non-standard *type-specifier*.	The semantics of such constructs are **undefined**.
6.7.2-4	The *type-specifier* **`_Complex`**	Implementations are not required to support complex types and their use impairs **PORTABILITITY**
6.7.2-5	The *type-specifier* **`_Imaginary`**	Implementations are not required to support imaginary types and their use impairs **PORTABILITY**.
6.7.2-3	The *type-specifier* **`_Bool`**	Implementations conforming to earlier version of the base language standard may not support **`_Bool`**, hence its use may impair **PORTABILITY**.

6.7.2.1 Structure and union specifiers

Orthosyntax:

struct-or-union-specifier = [*struct-or-union identifier*] **{** *struct-declaration-list* **}**
| *struct-or-union identifier* ;

struct-or-union = **`struct`**
| **`union`** ;

struct-declaration-list = *struct-declaration*
| *struct-declaration-list struct-declaration* ;

struct-declaration = *specifier-qualifier-list struct-declarator-list* ;

specifier-qualifier-list	=	*type-specifier* [*specifier-qualifier-list*]
	|	*type-qualifier* [*specifier-qualifier-list*] ;
struct-declarator-list	=	*struct-declarator*
	|	*struct-declarator-list* **,** *struct-declarator* ;
struct-declarator	=	*declarator*
	|	[*declarator*] **:** *constant-expr* ;

Parasyntax:

struct-or-union-specifier	=	[*struct-or-union SU-IDENTIFIER*] **{** *struct-declaration-list* **}**
	|	*struct-or-union SU-IDENTIFIER* ;
SU-IDENTIFIER	=	*identifier* ;

Note: An *SU-IDENTIFIER* is also referred to as a struct or union tag.

struct-declarator	=	*declarator*
	|	*BIT-FIELD-DECLARATOR* ;
BIT-FIELD-DECLARATOR	=	[*declarator*] **:** *constant-expr* ;

Designated constructs:

DCRN	Definition	Rationale
6.7.2.1-1	A *struct-declaration* whose *specifier-qualifier-list* specifies an incomplete type or a function type unless it specifies an incomplete array type for the last member of a structure that has more than one named member	Such a construct violates a **constraint**.
6.7.2.1-2	A *BIT-FIELD-DECLARATOR* whose *constant-expression* is not an integer constant expression.	Such a construct violates a **constraint**.
6.7.2.1-3	A *BIT-FIELD-DECLARATOR* whose *constant-expression* does not denote a nonnegative value of integer type.	Such a construct violates a **constraint**.
6.7.2.1-4	A *BIT-FIELD-DECLARATOR* whose *constant-expression* does not denote a nonnegative value of integer type whose value does not exceed the number of bits in an object of the type specified in the *specifier-qualifier-list* of its closest-containing *struct-declaration*..	Such a construct violates a **constraint**.
6.7.2.1-5	A *BIT-FIELD-DECLARATOR* whose *constant-expression* denotes the value zero and that does not closest-contain a *declarator*.	Such a construct violates a **constraint**.

6.7.2.1-6	A *BIT-FIELD-DECLARATOR* such that the *specifier-qualifier-list* of its closest-containing *struct-declaration* specifies a type that is not implementation-defined and is other than a qualified version of **`_Bool`**, **`signed int`**, or **`unsigned int`**	Such a construct violates a **constraint**
6.7.2.1-7	A *struct-declarator* that contains no *identifier*.	Behaviour is **undefined**.
6.7.2.1-8	A *struct-or-union-specifier* that has no *struct-declaration-list*.	Behaviour is **undefined**.
6.7.2.1-9	A *specifier-qualifier-list* containing a *storage-class-specifier*.	Some pre-standard compilers tolerated a *storage-class-specifier* in this context but such usage is non-standard and behaviour is **undefined**.
6.7.2.1-10	A *specifier-qualifier-list* that specifies a type other than an object type that is not variably modified.	Behaviour is **undefined**.
6.7.2.1-11	A *BIT-FIELD-DECLARATOR* such that the *specifier-qualifier-list* of its closest-containing *struct-declaration* specifies a type that is implementation-defined.	The semantics of the type are **implementation-defined.**
6.7.2.1-12	A construct whose behaviour may vary according to the packing of bits in a bit-field.	The packing of bits in a bit-field. Is **implementation-defined**.
6.7.2.1-13	A construct whose behaviour may vary according to the order of allocation of bits in a bit-field.	The order of allocation of bits in a bit-field is **implementation-defined**.
6.7.2.1-14	A construct whose behaviour may vary according to the alignment of the addressable storage unit allocated to hold a bit-field.	The alignment of addressable storage units allocated to hold bit-fields is **unspecified**.
6.7.2.1-15	A construct whose behaviour may vary according to the alignment of a member of a structure.	The alignment of members of structures is implementation-defined.
6.7.2.1-16	A *struct-or-union* that is **`union`**.	Some users believe that programmers are prone to make errors when using union types and may wish to ban or control their use in aid of **defensive programming**.
6.7.2.1-17	A *BIT-FIELD-DECLARATOR*.	Some users believe that programmers are prone to make errors when using bit-fields and may wish to ban or control them in aid of **defensive programming**.
6.7.2.1-18	A *BIT-FIELD-DECLARATOR* such that the *specifier-qualifier-list* of its closest-containing *struct-declaration* specifies a type other than **`signed int`** or **`unsigned int`**	Such usage may not be supported by implementations conforming to earlier versions of the base language standard, and its occurrence thus impairs **PORTABILITY**.
6.7.2.1-19	A *BIT-FIELD-DECLARATOR* such that the *specifier-qualifier-list* of its closest-containing *struct-declaration* specifies a type other than **`unsigned int`**	Believing that programmers are less prone to make errors under such a restriction, some users prefer to restrict bit-fields to **`unsigned int`** type in the in aid of **defensive programming**.

6.7.2.1-20	An *SU-IDENTIFIER* whose scope is not the *translation-unit* in which it appears.	Some users believe that declaring tags other than at file scope impairs the **understandability** of code.
6.7.2.2-21	An *SU-IDENTIFIER* whose scope has a non-empty intersection with the scope of a distinct *SU-IDENTIFIER* of the same spelling.	Some users believe that use of non-unique tags impairs the **understandability** of code.

6.7.2.2 Enumeration specifiers

Orthosyntax:

enum-specifier = **`enum`** [*identifier*] **`{`** *enumerator-list* **`}`**
| **`enum`** [*identifier*] **`{`** *enumerator-list* **`,`** **`}`**
| **`enum`** *identifier* ;

enumerator-list = *enumerator*
| *enumerator-list* **`,`** *enumerator* ;

enumerator = *enumeration-constant*
| *enumeration-constant* **=** *constant-expression* ;

Parasyntax:

enum-specifier = **`enum`** [*ENUM-IDENTIFIER*] **`{`** *enumerator-list* **`}`**
| **`enum`** [*ENUM-IDENTIFIER*] **`{`** *enumerator-list* **`,`** **`}`**
| **`enum`** *ENUM- IDENTIFIER* ;

ENUM-IDENTIFIER = *identifier ;*

Note: An *ENUM-IDENTIFIER* is also referred to as a tag.

Designated constructs:

DCRN	Definition	Rationale
6.7.2.2-1	An *enum-specifier* that does not have an *enumerator-list* occurring in a context where the type that it specifies is not complete.	Such a construct violates a **constraint**.
6.7.2.2-2	An *enumerator* whose *constant-expression* is not an integer constant expression whose value is representable as an **`int`**.	Such a construct violates a **constraint**.
6.7.2.2-3	An *enum-specifier*.	It is **implementation-defined** whether an enumerated type is compatible with **`char`**, a signed integer type or an unsigned integer type.
6.7.2.2-4	An *enumerator* whose *constant-expression* does not denote a non-negative value of integral type that does not exceed the value of **`SCHAR_MAX`**.	Reliance on any type other than **`char`** impairs **PORTABILITY**.
6.7.2.2-5	An *enum-specifier* that does not have an *identifier*.	Some users believe that not declaring tags impairs the **UNDERSTANDABILITY** of code.

6.7.2.2-6	An enumerator that has a *constant-expression*.	Some user believe that programmers are prone to make errors when using *constant-expression* in an enumerator. Accordingly they may ban or control such usage in aid **of defensive programming**.
6.7.2.2-7	An *ENUM-IDENTIFIER* whose scope is not the *translation-unit* in which it appears.	Some users believe that declaring tags other than at file scope impairs the **UNDERSTANDABILITY** of code.
6.7.2.2-8	An *ENUM-IDENTIFIER* whose scope has a non-empty intersection with the scope of a distinct *ENUM-IDENTIFIER* of the same spelling.	Some users believe that use of non-unique tags impairs the **understandability** of code.

6.7.2.3 Tags (NR)

Designated constructs:

See 6.7.2.1 and 6.7.2.2.

6.7.3 Type qualifiers

Orthosyntax:

```
type-qualifier      =     const
                    |     restrict
                    |     volatile ;
```

6.7.3.1 Formal definition of restrict (NR)

Designated Constructs:

DCRN	Definition	Rationale
6.7.3.1-1	A *specifier-qualifier-list* that contains **`restrict`** but does not specify a pointer type.	Such a construct violates a **constraint**.
6.7.3.1-2	A construct for which the behaviour attempts to modify an object-defined with a const-qualified type through use of an lvalue with non-const-qualified type.	Behaviour is **undefined**.
6.7.3.1-3	A construct for which the behaviour attempts to modify an object-defined with a volatile-qualified type through use of an lvalue with non-volatile-qualified type.	Behaviour is **undefined**.
6.7.3.1-4	A construct for which the behaviour attempts to access an object that has volatile-qualified type.	What behaviour constitutes such an access is **implementation-defined** and the presence of a construct attempting such access may impair the **ANALYZABILITY** of code.
6.7.3.1-5	The *type-qualifier* **`restrict`**.	This type qualifier may not be supported by implementations conforming to earlier version of the base language standard, hence its use impairs **PORTABILITY**.

6.7.4 Function specifiers

Orthosyntax:

function-specifier = `inline` ;

Designated Constructs:

DCRN	Definition	Rationale
6.7.4-1	The *function-specifier* `inline` appearing in the *specifier-qualifier-list* of a *declaration* of an *identifier* that is not the *identifier* of a function.	Such a construct violates a **constraint**.
6.7.4-2	An inline definition of a function with external linkage that contains a definition of a modifiable object with static storage duration or contains a reference to an identifier with external linkage.	Such a construct violates a **constraint**.
6.7.4-3	An inline definition of a function.	By providing an alternative to an external definition the presence of such a construct may impair the ANALYZABILITY of code, since it is **unspecified** which definition an implementation uses.
6.7.4-4	The *function-specifier* `inline`.	The extent to which an implementation takes any notice of `inline` is **implementation-defined**. Hence, some users believe that any *function-specifier* is misleading and impair the UNDERSTANDABILITY of code

6.7.5 Declarators

Orthosyntax:

declarator = [*pointer*] *direct-declarator* ;

direct-declarator = *identifier*
| **(** *declarator* **)**
| *direct-declarator* **[** [*type-qualifier-list*]
[*assignment-expression*] **]**
| *direct-declarator*
[`static` [*type-qualifier-list*]
assignment-expression **]**
| *direct-declarator* **[** *type-qualifier-list* **`static`**
assignment-expression **]**

| *direct-declarator* **[** [*type-qualifier-list*] * **]**
| *direct-declarator* **(** *parameter-type-list* **)**
| *direct-declarator* **(** [*identifier-list*] **)** ;

Parasyntax:

declarator = *POINTER-DECLARATOR*
| *NON-POINTER-DECLARATOR* ;

POINTER-DECLARATOR = *pointer direct-declarator* ;

NON-POINTER-DECLARATOR = *direct-declarator* ;

direct-declarator = *DD-IDENTIFIER*
| *DEC-IN-PAREN*
| *ARRAY-DECLARATOR*
| *FUNCTION-DECLARATOR* ;

DD-IDENTIFIER = *identifier* ;

DEC-IN-PAREN = **(** *declarator* **)** ;

Designated constructs:

DCRN	Definition	Rationale
6.7.5-1	A *declarator* that has a *pointer*.	The use of pointers can impair the **ANALYZABILITY** of code, for which reason some users may choose to ban them altogether in critical applications.
6.7.5-2	A *declarator*, the scope of whose *DD-IDENTIFIER* is a *compound-statement*, where that *declarator* is closest-contained by a *declaration* whose *declaration-specifiers* contain the *storage-class-specifier* **`extern`**.	Such a construct violates a constraint for implementations conforming to earlier versions of the base language standard and thereby impairs **PORTABILITY**.
6.7.5-3	A *direct-declarator* whose *identifier* appears nowhere else in its scope.	Such a declarator occurring in user-written code indicates a definition that is unused and may be eliminated, thereby reducing the volume of code under maintenance and hence promoting **MAINTAINABILITY**.
6.7.5-4	A *direct-declarator* whose *DD-IDENTIFIER* occurs in the same name space as a *DD-IDENTIFIER* of the same spelling contained by a distinct *direct-declarator*.	Some users believe that use of the same name in different name spaces impairs the **UNDERSTANDABILITY** of code.
6.7.5-5	A *direct-declarator* whose *DD-IDENTIFIER* has a scope that has a non-empty intersection with the scope of a *DD-IDENTIFIER* of the same spelling contained by a distinct *direct-declarator*.	Such a construct entails that the same identifier has been declared twice. Some users believe that programmers are prone to make errors when using multiple declarations of the same identifier and may wish to ban or control such usage in aid of **defensive programming**.

6.7.5.1 Pointer declarator

Orthosyntax:

pointer = ***** [*type-qualifier-list*]
| ***** [*type-qualifier-list*] *pointer* ;

type-qualifier-list = *type-qualifier*
| *type-qualifier-list type-qualifier* ;

Designated constructs:

DCRN	Definition	Rationale

6.7.5.1-1	A *pointer* containing more than two occurrences of `*`.	Some users believe that programmers are prone to make errors when using many levels of indirection and may wish to ban or control such usage in aid of **defensive programming**.
6.7.5-1-2	A *declarator* that is a *POINTER-DECLARATOR* and is closest-contained by a *declaration* whose *declaration-specifiers* contains the *type-qualifier* `const` or the *type-qualifier* `volatile`.	Confusing a constant pointer to a variable value and a variable pointer to a constant value is sufficiently common error that some users may wish to ban or control such usage in aid of **defensive programming**.

6.7.5.2 Array declarators

Parasyntax:

```
ARRAY-DECLARATOR                  =      PLAIN-ARRAY-DECLARATOR
                                  |      STATIC-ARRAY-DECLARATOR
                                  |      UNSPEC-SIZE-ARRAY-DECLARATOR ;

PLAIN-ARRAY-DECLARATOR            |      direct-declarator [ [ type-qualifier-list ]
                                                   [ ARRAY-BOUND ] ] ;
STATIC-ARRAY-DECLARATOR           |      direct-declarator
                                                   [ static [ type-qualifier-list ]
                                                   ARRAY-BOUND ]
                                  |      direct-declarator
                                                   [ type-qualifier-list static
                                                   ARRAY-BOUND ] ;

UNSPEC-SIZE-ARRAY-DECLARATOR      |      direct-declarator
                                                   [ [ type-qualifier-list ] * ] ;
```

ARRAY-BOUND ....
..
=...assignment-expression ;

Designated constructs:

DCRN	Definition	Rationale
6.7.5.2-1	An *ARRAY-BOUND* that does not have integer type.	Such a construct violates a **constraint**.
6.7.5.2-2	An *ARRAY-BOUND* that is a *constant-expression* but does not have a value that exceeds zero..	Such a construct violates a **constraint**.
6.7.5.2-3	An *ARRAY-BOUND* whose value does not exceed zero..	Such a construct violates a **constraint**.
6.7.5.2-4	An *identifier* denoting an object of a variably modified type but that does not have either block or function prototype scope and no linkage.	Such a construct violates a **constraint**.

6.7.5.2-5	An *identifier* denoting an object that has static storage duration and is a variable length array type.	Such a construct violates a **constraint**.
6.7.5.2-6	An *declarator* that is an *ARRAY-DECLARATOR* and is a *declarator* of a *declaration* whose *declaration-specifiers* specify an incomplete type or a function type.	Such a construct violates a **constraint**.
6.7.5.2-7	An *UNSPEC-SIZE-ARRAY-DECLARATOR*.	Use of arrays whose size is not known at translation time impairs the **ANALYZABILITY** of code.
6.7.5.2-8	An *ARRAY-BOUND* that is not a *constant-expression*.	Use of arrays whose size is not known at translation time impairs the **ANALYZABILITY** of code.
6.7.5.2-9	An *ARRAY-DECLARATOR* whose *direct-declarator* is neither a *DD-IDENTIFIER* nor a *DEC-IN-PAREN* whose *declarator* is an *DD-IDENTIFIER r*.	Use of such a construct impairs the **ANALYSABILITY** of code.

6.7.5.3 Function declarators (including prototypes)

Orthosyntax:

```
parameter-type-list            =    parameter-list
                               |    parameter-list ,  ... ;

parameter-list                 =    parameter-declaration
                               |    parameter-list , parameter-declaration ;

parameter-declaration          =    declaration-specifiers declarator
                               |    declaration-specifiers [ abstract-declarator ] ;

identifier-list                =    identifier
                               |    identifier-list , identifier ;
```

Parasyntax:

```
FUNCTION-DECLARATOR            =    FUNCTION-PROTOTYPE
                               |    K-AND-R-FUNCTION-DECLARATOR ;

FUNCTION-PROTOTYPE             =    direct-declarator ( parameter-type-list ) ;

K-AND-R-FUNCTION-DECLARATOR         =      direct-declarator ( [ identifier-list ] ) ;

parameter-declaration          =    PARAM-DEC-SPECIFIERS PARAMETER-DECLARATOR
                               |    PARAM-DEC-SPECIFIERS [ abstract-declarator ]  ;

PARAM-DEC-SPECIFIERS           =    declaration-specifiers  ;

PARAMETER-DECLARATOR           =    declarator  ;

parameter-type-list            =    parameter-list
                               |    ELLIPSIS-PARAMETER-LIST ;

ELLIPSIS-PARAMETER-LIST        =    parameter-list  ,  ...  ;
```

Designated constructs:

DCRN	Definition	Rationale
6.7.5.3-1	An *declarator* that is an *FUNCTION-DECLARATOR* and is a *declarator* of a *declaration* whose *declaration-specifiers* specify an array type or a function type.	Such a construct violates a **constraint**.
6.7.5.3-2	A *parameter-declaration* whose *declaration-specifiers* contain a *storage-class-specifier* other than **`register`**.	Such a construct violates a **constraint**.
6.7.5.3-3	A *K-AND-R-FUNCTION-DECLARATOR* whose *identifier-list* is not contained by the corresponding function definition.	Such a construct violates a **constraint**.
6.7.5.3-4	A *PARAM-DEC-SPECIFIERS* that: (a) is closest-contained by a *FUNCTION-DECLARATOR* that is contained by the *function-definition* of the corresponding function, and that (b) specifies a type that is an incomplete type after adjustment.	Such a construct violates a **constraint**.
6.7.5.3-5	An *ELLIPSIS-PARAMETER-LIST*.	Use of functions that take variable numbers of arguments impairs the **ANALYSABILITY** of code.
6.7.5.3-6	A *PARAMETER-DECLARATOR* that is not an *identifier*.	Use of parameters that have pointer type can impair the **ANALYSABILITY** of code.
6.7.5.3-7	A *FUNCTION-PROTOTYPE* whose *direct-declarator* is neither an *identifier* nor a *DEC-IN-PAREN* whose *declarator* is an *identifier*.	The use of such constructs can impair the **ANALYZABILITY** of code.
6.7.5.3-8	A *K-AND-R-FUNCTION-DECLARATOR*.	The use of such constructs limits the ability of static checking tools to perform type checking, thus impairing the **ANALYZABILITY** of code.
6.7.5.3-9	A *parameter-declaration* whose *PARAM-DEC-SPECIFIERS* specify an incomplete type.	The use of such contructs can severely impair the **ANALYZABILITY** of code.
6.7.5-10	A *declarator* that is a *FUNCTION-DECLARATOR* and is a *declarator* closest-contained by a *declaration* whose *declaration-specifiers* specify a function type, an array type, a struct or union type or an incomplete type other than **`void`**.	Such a construct may not be supported by implementations conforming to earlier versions of the base language standard, thereby impairing **PORTABILITY**.
6.7.5-11	A *parameter-declaration* whose *PARAM-DEC-SPECIFIERS* specify a type that is a function type or a struct or union type..	Such a construct may not be supported by implementations conforming to earlier versions of the base language standard, thereby impairing **PORTABILITY**.

6.7.6 Type names

Orthosyntax:

type-name = *specifier-qualifier-list* [*abstract-declarator*] ;

```
abstract-declarator            =    pointer
                               |    [ pointer ] direct-abstract-declarator  ;

direct-abstract-declarator     =    ( abstract-declarator )
                               |    [ direct-abstract-declarator ]
                                            [ assignment-expression ]
                               |    [ direct-abstract-declarator ] [ * ]
                               |    [ direct-abstract-declarator ]
                                            ( [ parameter-type-list ] )  ;
```

Designated constructs:

DCRN	Definition	Rationale
6.7.6-1	A *type-name* whose *abstract-declarator* closest-contains a *pointer*.	Uncontrolled use of pointer types can impair the ANALYSABILITY of code.

6.7.7 Type definitions

Orthosyntax:

```
typedef-name          =       identifier ;
```

Designated constructs:

DCRN	Definition	Rationale
6.7.7-1	A *typedef-name* that specifies a variably modified type but does not have block scope.	Such a construct violates a **constraint**.
6.7.7-2	A *typedef-name* that specifies a type of unknown size.	Some users believe that programmers are prone to make errors when using such a *typedef-name* and may wish to ban or control such usage in aid of **defensive programming**.
6.7.7-3	An *identifier* that is a *typedef-name* and whose scope is not the *translation-unit* in which it appears.	Some users believe that it impairs the UNDERSTANDABILITY of code if such an identifier does not have a scope that is not the translation-unit in which it appears.

6.7.8 Initialisation

Orthosyntax:

```
initializer                    =    assignment-expr
                               |    { initializer-list }
                               |    { initializer-list ,  } ;

initializer-list               =    [ designation ]  initializer
                               |    initializer-list ,   [ designation ]  initializer ;

designation                    =    designator-list =  ;

designator-list                =    designator
```

```
                              |     designator-list  designator  ;

designator                    =     [  constant-expression  ]
                              |     .  identifier  ;
```

Parasyntax:

```
initializer                   =     assignment-expr
                              |     { initializer-list }
                              |     { COMMA-TERMINATED-INIT-LIST } ;
COMMA-TERMINATED-INIT-LIST    =     { initializer-list ,   } ;

designator                    =     ARRAY-ELEMENT-DESIG
                              |     STRUCT-MEMBER-DESIG  ;

ARRAY-ELEMENT-DESIG           =     [  constant-expression  ]  ;

STRUCT-MEMBER-DESIG  =        .  identifier  ;
```

Designated constructs:

DCRN	Definition	Rationale
6.7.8-1	An *initializer* that attempts to provide a value for an object not contained within the entity being initialized.	Such a construct violates a **constraint**.
6.7.8-2	An *initializer* for an entity that is not one of the following: (a) an array of unknown size, or (b) an object that is not a variable length array type.	Such a construct violates a **constraint**.
6.7.8-3	An *initializer* for an object of unknown size that is not an array object.	Such a construct violates a **constraint**.
6.7.8-4	An *initializer* for an object of static storage duration that contains an *expression* that is neither a *constant-expression* nor a *string-literal*.	Such a construct violates a **constraint**.
6.7.8-5	An *initializer* for an object whose *identifier* has block scope and external or internal linkage.	Such a construct violates a **constraint**.
6.7.8-6	An *ARRAY-ELEMENT-DESIG* for part of a current object that is an array.	Such a construct violates a **constraint**.
6.7.8-7	An *STRUCT-MEMBER-DESIG* for part of a current object that is not a struct or union.	Such a construct violates a **constraint**.
6.7.8-8	An *initializer* for an object of array, struct or union type that has automatic storage duration.	Such a construct may not be supported by some implementations that conform to earlier version of the base language standard, under which their use may result in **undefined behaviour**.

6.7.8-9	An *initializer* in which the numbers, types and sizes of every contained *assignment-expr* do not exactly match those of the object that it initializes.	Such can be highly confusing to readers of programs and is likely to impair the **understandability** of code.
6.7.8-10	A *COMMA-TERMINATED-INIT-LIST*.	Some users deprecate such usage believing it to be poor style and possibly to impair **UNDERSTANDABILITY**.

6.8 Statements and blocks

Orthosyntax:

statement = *labeled-statement*
| *compound-statement*
| *expression-statement*
| *selection-statement*
| *iteration-statement*
| *jump-statement* ;

Designated constructs:

DCRN	Definition	Rationale
6.8-1	A *statement* whose E-behaviour contains no side effect.	Such a styatement may be redundant in which case it can be removed without effect on the behaviour of the program.
6.8-2	A source line containing more than one *statement*.	Some users believe that adhering to one statement per line promotes the **UNDERSTANDABILITY** of code.

6.8.1 Labelled statement

Orthosyntax:

labeled-statement = *identifier* **:** *statement*
| **`case`** *constant-expr* **:** *statement*
| **`default`** **:** *statement* ;

Parasyntax:

labeled-statement = *IDENTIFIER-LABELED-STATEMENT*
| *CASE-LABELED-STATEMENT*
| *DEFAULT-LABELED-STATEMENT* ;

IDENTIFIER-LABELED-STATEMENT = *identifier* **:** *statement* ;

CASE-LABELED-STATEMENT = **`case`** *constant-expr* **:** *statement* ;

DEFAULT-LABELED-STATEMENT = **`default`** **:** *statement* ;

Designated constructs:

DCRN	Definition	Rationale
6.8.1-1	A *CASE-LABELLED-STATEMENT* that is not contained by a *SWITCH-STATEMENT*.	Such a construct violates a **constraint**.
6.8.1-2	A *DEFAULT-LABELLED-STATEMENT* that is not contained by a *SWITCH-STATEMENT*.	Such a construct violates a **constraint**.
6.8.1-3	A *labeled-statement* that contains more than one *labelled-statement*.	Some users consider that giving a statement more than one label may impair the **UNDERSTANDABILITY** of code.

6.8.1-4	An *IDENTIFIER-LABELLED-STATEMENT*.	Such a statement is required only to provide a destination for a *GOTO-STATEMENT*. If the latter are banned, then there is no need for any *IDENTIFIER-LABELLED-STATEMENT*.

6.8.2 Compound statement

Orthosyntax:

compound-statement = **`{`** [*block-item-list*] **`}`** ;

block-item-list = *block-item*
| *block-item-list* *block-item* ;

block-item = *declaration*
| *statement* ;

Designated constructs:

DCRN	Definition	Rationale
6.8.2-1	A *compound-statement* closest-containing a *declaration* and a *statement* such that the *declaration* appears after the *statement*.	Such a construct may not be supported by implementations conforming to earlier version of the base language standard and their use impairs **PORTABILITY**.
6.8.2-2	A *compound-statement* containing more than one *IDENTIFIER-LABELED-STATEMENT* whose *identifier*s have the same spelling.	Such a construct violates a **constraint**.

6.8.3 Expression and null statements

Orthosyntax:

expression-statement = [*expression*] **`;`**

Designated constructs:

DCRN	Definition	Rationale
6.8.3-1	An *expression-statement* that is a *FUNCTION-CALL EXPRESSION* whose *FUNCTION-DESIGNATOR* denotes a function whose return type is not **`void`**.	In this context the value returned by the function is discarded. Some users believe that discarding of function values is associated with programmer error and may wish to ban or control such usage in aid of **defensirve programming**.
6.8.3-2	An *expression-statement* that has no *expression*.	Some users believe that such usage is confusing and impairs **understandability**. Others regard it as a useful **defensive programming** practice in selection statements.

6.8.4 Selection statements

Orthosyntax

selection-statement	=	**if (** *expression* **)** *statement*
	|	**if (** *expression* **)** *statement* **else** *statement*
	|	**switch (** *expression* **)** *statement ;*

Parasyntax

selection-statement	=	*BINARY-SELECTION*
	|	*SWITCH-STMT* ;
BINARY-SELECTION	=	*PLAIN-IF-STMT*
	|	*IF-ELSE-STMT* ;
PLAIN-IF-STMT	=	**if (** *IF-EXPR* **)** *TRUE-STMT* ;
IF-ELSE-STMT	=	**if (** *IF-EXPR* **)** *TRUE-STMT* **else** *FALSE-STMT* ;
IF-EXPR	=	*expression* ;
EXPLICIT-LOGICAL-EXPR	=	*EXPLICIT-REL-EXPR*
	|	*EXPLICIT-EQUALITY-EXPR*
	|	*EXPLICIT-LAND-EXPR*
	|	*EXPLICIT-LOR-EXPR*
	|	**! (** *EXPLICIT-LOGICAL-EXPR* **)** ;
TRUE-STMT	=	*statement* ;
FALSE-STMT	=	*statement* ;
SWITCH-STMT	=	**switch (** *SWITCH-EXPR* **)** *SWITCH-BODY* ;
SWITCH-EXPR	=	*expression* ;
SWITCH-BODY	=	*statement* ;
STRUC-SWITCH-STMNT	=	**switch** (*SWITCH-EXPR*) *STRUC-SWITCH-BODY* ;
STRUC-SWITCH-BODY	=	**{** *CASE-CLAUSES* **;** *DEFAULT-CLAUSE* **}** ;
CASE-CLAUSES	=	*CASE-CLAUSE*
	|	*CASE-CLAUSES* **;** *CASE-CLAUSE* ;
CASE-CLAUSE	=	**case** *constant-expr* **:** *CASE-GROUP* ;
DEFAULT-CLAUSE	=	**default :** *CASE-GROUP* ;
CASE-GROUP	=	**{** *statement-list* **; break }** ;

6.8.4.1 The if statement

Designated constructs:

DCRN	Definition	Rationale
6.8.4.1-1	An *IF-EXPR* that does not have scalar type.	Such a construct violates a **constraint**.

6.8.4.1-2	An *IF-EXPR* that is an *EXPLICIT-SIMPLE-ASSIGNMENT-EXPR*.	The programmer may have written `=` when `==` was intended. This error is sufficiently common that the construct warrants being diagnosed in aid of **defensive programming**.
6.8.4.1-3	An *IF-EXPR* that contains a *SIDE-EFFECTIVE-OPERATOR*.	Some users believe that programmers are prone to make errors when using such a construct. Accordingly they may wish to ban or control them in aid of **defensive programming**.
6.8.4.1-4	An *IF-EXPR* that is *constant-expression* or is deduced to have a value that never changes.	Such constructs are often the result of programming errors and are sufficiently common to warrant being diagnosed in aid of **defensive programming**.
6.8.4.1-5	An *IF-EXPR* that is not an *EXPLICIT-LOGICAL-EXPR*.	Some users believe making logical operations explicit in selection statements promoes **UNDERSTANDABILITY** and is a useful **defensive programming** technique that may help programmers to detect logical errors to during coding.
6.8.4.1-6	A *TRUE-STMT* that is not a *compound-statement*.	Some users consider that prohibition of this construct enhances the **understandability** of code.
6.8.4.1-7	A *FALSE-STMT* that is not a *compound-statement*.	Some users consider that prohibition of this construct enhances the **understandability** of code.
6.8.4.1-8	A *PLAIN-IF-STMT*.	Some users believe that writing `else` cases explicitly is a useful **defensive programming** technique that helps programmers to find logical errors to during coding.
6.8.4.1-9	An *IF-ELSE-STMT* whose *FALSE-STMT* is *a BINARY-SELECTION* that does not begin on the same line as the `else` of the *IF-ELSE-STMT*.	Some users consider that prohibition of this construct enhances the **understandability** of code.

6.8.4.2 The switch statement

Designated constructs:

DCRN	Definition	Rationale
6.8.4.2-1	A *SWITCH-EXPR* that does not have integer type.	Such a construct violates a **constraint**.
6.8.4.2-2	A *SWITCH-STMNT* closest-containing `case` or `default` where either is within the scope of an identifier with a variably-modified type but where the *SWITCH-STMNT* is not itself within the scope of that identifier.	Such a construct violates a **constraint**.
6.8.4.2-3	A *constant-expr* of a *CASE-LABELED-STATEMENT* that is not an integer constant expression.	Such a construct violates a **constraint**.
6.8.4.2-4	A *SWITCH-STMNT* closest-containing two distinct *CASE-LABELED-STATEMENT* whose *constant-expr* have the same value after conversion.	Such a construct violates a **constraint**.

6.8.4.2-5	A *SWITCH-STMNT* closest-containing more than one **`default`**.	Such a construct violates a **constraint**.
6.8.4.2-6	A *SWITCH-EXPR* that is an *EXPLICIT-LOGICAL-EXPR*.	Such constructs are often the result of programming errors and are sufficiently common to warrant being diagnosed in aid of **defensive programming**.
6.8.4.2-7	A *SWITCH-EXPR* that is a *constant-expression* or is deduced to have a value that never changes.	Such constructs are often the result of programming errors and are sufficiently common to warrant being diagnosed in aid of **defensive programming**.

6.8.5 Iteration statements

Orthosyntax:

iteration-statement = **`while`** **`(`** *expression* **`)`** *statement*
| **`do`** *statement* **`while`** **`(`** *expression* **`)`** **`;`**
| **`for`** **`(`** [*expression*] **`;`**
[*expression*] **`;`**
[*expression*] **`)`** *statement*
| **`for`** **`(`** *declaration* [*expression*] **`;`**
[*expression*] **`)`** *statement* ;

Parasyntax:

iteration-statement = *WHILE-STATEMENT*
| *DO-WHILE-STATEMENT*
| *FOR-STATEMENT* ;

WHILE-STATEMENT = **`while`** **`(`** *WHILE-EXPRESSION* **`)`** *BODY* ;

DO-WHILE-STATEMENT = **`do`** *BODY* **`while`** **`(`** *WHILE-EXPRESSION* **`)`** **`;`**

FOR-STATEMENT = *C90-FOR-STATEMENT*
| *C99-FOR-STAMEMENT* ;

C90-FOR-STATEMENT = **`for`** **`(`** [*expression*] **`;`**
[*WHILE-EXPRESSION*] **`;`**
[*expression*] **`)`** *BODY* ;

C99-FOR-STAMEMENT = **`for`** **`(`** *declaration* [*WHILE-EXPRESSION*] **`;`**
[*expression*] **`)`** *BODY* ;

WHILE-EXPRESSION = *expression* ;

BODY = *statement* ;

Designated constructs:

DCRN	Definition	Rationale
6.8.5-1	An *WHILE-EXPRESSION* that does not have scalar type.	Such a construct violates a **constraint**.

6.8.5-2	An *WHILE-EXPRESSION* that does not have arithmetic type.	Some users believe that use of non-arithmetic types impairs the **UNDERSTANDABILITY** of code.
6.8.5-3	An *WHILE-EXPRESSION* that is not an *EXPLICIT-LOGICAL-EXPR*.	Some users believe that not using an *EXPLICIT-LOGICAL-EXPR* impairs the **UNDERSTANDABILITY** of code.
6.8.5-4	A *WHILE-EXPRESSION* that is an *EXPLICIT-SIMPLE-ASSIGNMENT-EXPR*.	Such constructs are often the result of programming errors and are sufficiently common to warrant being diagnosed in aid of **defensive programming**.
6.8.5-5	An *WHILE-EXPRESSION* that is *constant-expr*.	Such constructs are often the result of programming errors and are sufficiently common to warrant being diagnosed in aid of **defensive programming**.
6.8.5-6	An *WHILE-EXPRESSION* that is not a *constant-expr* but is statically deduced to have a constant value.	Such constructs are often the result of programming errors and are sufficiently common to warrant being diagnosed in aid of **defensive programming**.
6.8.5-7	An *WHILE-EXPRESSION* that is a *SIDE-EFFECTIVE-EXPR*.	Some users believe that using a *SIDE-EFFECTIVE-EXPR* impairs the **UNDERSTANDABILITY** of code.
6.8.5-8	A *BODY* that is not a *compound-statement*.	Some users believe that not using a *compound-statement* impairs the **UNDERSTANDABILITY** of code.

Note: A loop for which the *WHILE-EXPRESSION* takes a constant value is sometimes required for implementation of idle-wait states. It is important to ensure that such loops are not removed by code optimisers. If an idle-wait loop is required, the following form may be found useful:

```
{
   volatile int i = 2;

   while ( i != 3)
   {
      i = (i+i) % 7;
   }
}
```

The effect of this construct is to cycle the value of `i` indefinitely through the quadratic residues mod 7. The assignment to `i` has the effect of multiplying it by 2 mod 7 and since 2 is a quadratic residue mod 7, `i` never attains the value 3, which is a non-quadratic residue mod 7. The presence of a side effect on `i` (both by assignment and because `i` is declared `volatile`) is intended to defeat an incautious optimiser that might otherwise attempt to remove the loop. It is believed that few optimisers can make the inferences in elementary number theory required to prove that the loop is infinite. This may not, however, be beyond the power of a dynamic analysis tool.

6.8.5.1 The while statement (NR)

6.8.5.2 The do statement

Designated constructs:

DCRN	Definition	Rationale
6.8.5.2-1	A *DO-WHILE-STATEMENT* whose *BODY* and `while` are not separated by a single space.	Some users believe that a single separating space is a usage that promotes using a the

		UNDERSTANDABILITY of code.

6.8.5.3 The for statement

Designated constructs:

DCRN	Definition	Rationale
6.8.5.3-1	A *declaration* of a *C99-FOR-STATEMENT* that declares an *identifier* for an object that does not either have automatic storage duration or have register storage class.	Such a construct violates a **constraint**.
6.8.5.3-2	A *C99-FOR-STATEMENT*.	The use of such a construct may impair **PORTABILITY** to implementations conforming to earlier version of the base language standard.
6.8.5.3-3	A *FOR-STATEMENT* that does not closest-contain a *WHILE-EXPRESSION*.	Such usage is treated as if the *WHILE-EXPRESSION* had a constant-value (c.f. DCRN 6.8.5-5
6.8.5.3-4	A *FOR-STATEMENT* for whose *BODY* the E-behaviour contains a modifying access to an object and for whose *WHILE-EXPRESSION* the E-behaviour contains any access to the same object.	Some users prefer to modify loop control variables only in the third expression of a for-statement and consider that such usage promotes **UNDERSTANDABILITY**.
6.8.5.3-5	A loop-control variable that has floating type.	Some users consider that use of such variables is prone to error and prefer to ban or control them in aid of **defensive programming**.
6.8.5.3-6	A *FOR-STATEMENT* for which there is more than one loop-control variable.	Some users consider that use of more than one such variables is prone to error and prefer to ban or control them in aid of **defensive programming**.

Note: Since the notion of a loop-control variable is not syntactically defined, diagnostic processors may use heuristic methods to identify such variables and hence their capacity for such identification may exhibit wide variation.

6.8.6 Jump statements

Orthosyntax:

jump-statement = **`goto`** *identifier* **`;`**
| **`continue ;`**
| **`break ;`**
| **`return`** [*expression*] **`;`** ;

Parasyntax:

jump-statement = *GOTO-STATEMENT*
| *CONTINUE-STATEMENT*
| *BREAK-STATEMENT*
| *RETURN-STATEMENT* ;

GOTO-STATEMENT = **`goto`** *identifier* **`;`** ;

CONTINUE-STATEMENT = **`continue ;`** ;

BREAK-STATEMENT = **`break ;`** ;

RETURN-STATEMENT = *PLAIN-RETURN-STMNT*
| *EXPR-RETURN-STMNT* ;

PLAIN-RETURN-STMNT = **`return ;`** ;

EXPR-RETURN-STMNT = **`return`** [*expression*] **`;`** ;

6.8.6.1 The goto statement

Designated constructs:

DCRN	Definition	Rationale
6.8.6.1-1	A *GOTO-STATEMENT* whose *identifier* is not the *identifier* of an *IDENTIFIER-LABELED-STATEMENT* contained in the same *compound-statement* as that *GOTO-STATEMENT*	Such a construct violates a **constraint**.
6.8.6.1-2	A *GOTO-STATEMENT* that is within the scope of an *identifier I* having a variably-modified type but such that its own *identifier* is the *identifier* of an *IDENTIFIER-LABELLED-STATEMENT* that is outside that scope of I.	Such a construct violates a **constraint**.
6.8.6.1-3	A *GOTO-STATEMENT*.	Some users believe that programmers are prone to make errors when using the *GOTO-STATEMENT* and may therefore wish to ban or control its use in aid of **defensive programming**.

6.8.6.2 The continue statement

Designated constructs:

DCRN	Definition	Rationale
6.8.6.2-1	A *CONTINUE-STATEMENT* that is not contained by a *BODY*.	Such a construct violates a **constraint**.
6.8.6.2-2	A *CONTINUE-STATEMENT*.	Some users believe that programmers are prone to make errors when using the *CONTINUE-STATEMENT* and may therefore wish to ban or control its use in aid of **defensive programming**.

6.8.6.3 The break statement

Designated constructs:

DCRN	Definition	Rationale
6.8.6.3-1	A *BREAK-STATEMENT* that is not contained by a *BODY*.	Such a construct violates a **constraint**.

6.8.6.3-2	A *BREAK-STATEMENT* that is contained by the *BODY* of an *ITERATION-STATEMENT*.	Some users believe that programmers are prone to make errors when using the *BREAK-STATEMENT* within loops and may therefore wish to ban or control its use in aid of **defensive programming**.

6.8.6.4 The return statement

Designated constructs:

DCRN	Definition	Rationale
6.8.6.4-1	An *EXPR-RETURN-STATEMENT* contained by the *compound-statement* of the *function-definition* of a function whose return type is `void`.	Such a construct violates a **constraint**.
6.8.6.4-2	An *PLAIN-RETURN-STATEMENT* contained by the *compound-statement* of the *function-definition* of a function whose return type is not `void`.	Such a construct violates a **constraint**.
6.8.6.4-3	A *RETURN-STATEMENT* whose *expression* denotes a value of pointer-type that points to an object whose scope is the *compound-statement* containing that *RETURN-STATEMENT*..	Dereferencing such a returned value will lead to undefined behaviour. Accordingly some users may wish to ban or control use of this construct in aid of **defensive programming**.
6.8.6.4-4	A *RETURN-STATEMENT* whose *expression* does not denote a value of arithmetic type.	Some users believe that programmers are prone to make errors when using such a construct and may therefore wish to ban or control its use in aid of **defensive programming**.
6.8.6.4-5	An *EXPR-RETURN-STATEMENT* whose *expression* does not denote a value of a type identical to the return type of the *function-definition* in whose *compound-statement* it is contained.	Some users believe that programmers are prone to make errors when using such a construct and may therefore wish to ban or control its use in aid of **defensive programming**.

6.9 External definitions

Orthosyntax:

translation-unit	=	*external-declaration*
	|	*translation-unit external-declaration* ;
external-declaration	=	*function-definition*
	|	*declaration*

Designated constructs:

DCRN	Definition	Rationale
6.9-1	An *external-declaration* that contains either of the *storage-class-specifier* `auto` or `register`.	Such a construct violates a **constraint**.
6.9-2	A *translation-unit* containing more than one *external-declaration* that is an external definition for a given *identifier* with internal linkage.	Such a construct violates a **constraint**.
6.9-3	Distinct *declarations* that refer to the same object or function but that specify incompatible types.	Behaviour is **undefined**.
6.9-4	A construct provided to support T-behaviour of assembly code appearing within a *translation-unit*.	Behaviour for such a construct is **implementation-dependent**.
6.9-5	A *translation-unit* containing a construct whose interpretation in C++ differs from the interpretation of a syntactically identical construct in C.	Such a construct impairs the **PORTABILITY** of code between C and C++ implementations.
6.9-6	A source file not that does not contain a *translation-unit*.	In certain circumstances preprocessing of a source file may result in a file that contains no external declarations (e.g. owing to the effects of conditional compilation). Some users like to be warned if this occurs and a diagnostic processor may flag the condition if it arises.

6.9.1 Function definitions

Orthosyntax:

function-definition	=	[*declaration-specifiers*] *declarator* [*declaration-list*] *compound-statement* ;
declaration-list	=	*declaration*
	|	*declaration-list declaration*;

Parasyntax:

function-definition	=	[*declaration-specifiers*] *declarator* [*declaration-list*] *FUNCTION-BLOCK* ;
FUNCTION-BLOCK	=	*compound-statement* ;

Designated constructs:

DCRN	Definition	Rationale

6.9.1-1	A *function-definition* whose declared *identifier* does not have function type.	Such a construct violates a **constraint**.
6.9.1-2	A *function-definition* the return type of whose declared function is neither the void type nor an object type other than an array type.	Such a construct violates a **constraint**.
6.9.1-3	A *function-definition* whose *declaration-specifiers* contain a *storage-class-specifier* other that **`extern`** or **`static`**.	Such a construct violates a **constraint**.
6.9.1-4	A *function-definition* whose *declarator* is a *FUNCTION-PROTOTYPE* and that itself has a *declaration-list*.	Such a construct violates a **constraint**.
6.9.1-5	A *function-definition* whose *declarator* is a *K-AND-R-FUNCTION-DECLARATOR* whose *identifier-list* does not correspond to the *declaration-list* of the *function-definition*.	Such a construct violates a **constraint**.
6.9.1-6	A *FUNCTION-BLOCK* that contains both a *PLAIN-RETURN-STMNT* and an *EXPR-RETURN-STMNT*.	Behaviour for one or the other is **undefined**.
6.9.1-7	A *FUNCTION-BLOCK* that does not contain a *RETURN-STATEMENT*.	For such a construct the possibility exists that the terminating **}** of the function-block may be reached and that the value of the function call will be used in the calling environment. In this occurs, the behaviour is **undefined**.
6.9.1-8	A *function-definition* whose *declarator* does not contain a *FUNCTION-PROTOTYPE*.	The use of such function-definitions impairs the **ANALYSABILITY** of code.
6.9.1-9	A *FUNCTION-BLOCK* that contains more than one *RETURN-STATEMENT*.	Some users believe that adherence to a single-entry, single-exit convention promotes the **UNDERSTANDABILITY** of code.
6.9.1-10	A construct whose E-behaviour may vary according to the layout of storage for function parameters.	The layout of storage for parameters is **unspecified**.
6.9.1-11	A *function-definition* that declares a parameter but whose function block contains no access to that parameter.	Some users believe that the presence of such unused parameters impair the **UNDERSTANDABILITY** of code.

6.9.2 External object definitions

DCRN	**Definition**	**Rationale**
6.9.2-1	A tentative definition of an object that has internal linkage and incomplete type.	Behaviour is **undefined**.

6.10 Preprocessing directives

Orthosyntax:

```
preprocessing-file   =   [ group ] ;

group                =   group-part
                     |   group  group-part ;

group-part           =   [ pp-tokens ] new-line
                     |   if-section
                     |   control-line ;

if-section           =   if-group  [ elif-groups ]  [ else-group ]  endif-line ;

if-group             =   # if constant-expression new-line  [ group ]
                     |   # ifdef identifier new-line  [ group ]
                     |   # ifndef identifier new-line [ group ] ;

elif-groups          =   elif-group
                     |   elif-groups elif-group ;

elif-group           =   # elif constant-expression new-line [ group ] ;

else-group           =   # else new-line [ group ] ;

endif-line           =   # endif new-line ;

control-line         =   # include pp-tokens new-line
                     |   # define identifier replacement-list new-line
                     |   # define identifier  lparen  [ identifier-list ]
                                replacement-list new-line
                     |   # define identifier  lparen   . . . )
                                replacement-list new-line
                     |   # define identifier  lparen  identifier-list   , . . . )
                                replacement-list new-line
                     |   # undef identifier new-line
                     |   # line pp-tokens new-line
                     |   # error [ pp-tokens ]  new-line
                     |   # pragma [ pp-tokens ]  new-line
                     |   #   new-line ;

lparen               =   a left-parentheses without preceding white space ;

replacement-list     =   [ pp-tokens ] ;

pp-tokens            =   preprocessing-token
                     |   pp-tokens preprocessing-token ;

new-line             =   the new-line character ;
```

Parasyntax:

```
control-line            =    INCLUDE-DIRECTIVE
                        |    PLAIN-DEFINE-DIRECTIVE
                        |    FLIKE-DEFINE-DIRECTIVE
                        |    UNDEF-DIRECTIVE
                        |    LINE-DIRECTIVE
                        |    ERROR-DIRECTIVE
                        |    PRAGMA-DIRECTIVE
                        |    NULL-DIRECTIVE ;

DIRECTIVE               =    IF-DIRECTIVE
                        |    IFDEF-DIRECTIVE
                        |    IFNDEF-DIRECTIVE
                        |    ELIF-DIRECTIVE
                        |    ELSE-DIRECTIVE
                        |    ENDIF-DIRECTIVE
                        |    INCLUDE-DIRECTIVE
                        |    PLAIN-DEFINE-DIRECTIVE
                        |    FLIKE-DEFINE-DIRECTIVE
                        |    EMPTY-VAR-FLIKE-DEFINE-DIRECTIVE
                        |    VAR-FLIKE-DEFINE-DIRECTIVE
                        |    UNDEF-DIRECTIVE
                        |    LINE-DIRECTIVE
                        |    ERROR-DIRECTIVE
                        |    PRAGMA-DIRECTIVE
                        |    NULL-DIRECTIVE ;
```

Designated constructs:

DCRN	Definition	Rationale
6.10-1	A *DIRECTIVE* whose opening hash **#** is followed by a white space character.	Such a construct violates a **constraint**. (The **#** will be treated as a **#** preprocessing token)
6.10-2	A directive that contains a white space character other than space or horizontal tab between one preprocessing-token and another.	Such a construct violates a **constraint**.
6.10-3	A non-standard *control-line*.	T-behaviour is **implementation-dependent**.
6.10-4	A non-standard *endif-line*.	T-behaviour is **implementation-dependent**.
6.10-5	A non-standard *if-group*.	T-behaviour is **implementation-dependent**.
6.10-6	A non-standard *elif-group*.	T-behaviour is **implementation-dependent**.
6.10-7	A non-standard *else-group*.	T-behaviour is **implementation-dependent**.
6.10-8	A *DIRECTIVE* whose opening hash **#** does not occur in the first character position of a source line.	Such a construct may not be treated as a directive by pre-standard implementations thereby impairing **PORTABILITY**.

6.10.1 Conditional inclusion

Parasyntax:

if-group = *IF-DIRECTIVE* [*group*] ;
| *IFDEF-DIRECTIVE* [*group*] ;
| *IFNDEF-DIRECTIVE* [*group*] ;

IF-DIRECTIVE = **`# if`** *constant-expression new-line* ;

IFDEF-DIRECTIVE = **`# ifdef`** *identifier new-line* ;

IFNDEF-DIRECTIVE = **`# ifndef`** *identifier new-line* ;

elif-group = *ELIF-DIRECTIVE* [*group*] ;

ELIF-DIRECTIVE = **`# elif`** *constant-expression new-line* ;

else-group = *ELSE-DIRECTIVE* [*group*] ;

ELSE-DIRECTIVE = **`# else`** *new-line* ;

endif-line = *ENDIF-DIRECTIVE* ;

ENDIF-DIRECTIVE = **`# endif`** *new-line* ;

Designated constructs:

DCRN	Definition	Rationale
6.10.1-1	An *IF-DIRECTIVE* , *IFDEF-DIRECTIVE* or *IFNDEF-DIRECTIVE* whose *constant-expression* is not an integer constant expression.	Such a construct violates a **constraint**.
6.10.1-2	An *IF-DIRECTIVE* , *IFDEF-DIRECTIVE* or *IFNDEF-DIRECTIVE* whose *constant-expression* is or expands to one that contains **`defined`** not followed by an *identifier* or (*identifier*).	T-behaviour is **undefined**.
6.10.1-3	A non-standard *if-group* that begins with **`# ifdef`** or **`# ifndef`** in neither case followed by an *identifier*.	T-behaviour is **undefined**.
6.10.1-4	An *IF-DIRECTIVE* , *IFDEF-DIRECTIVE* or *IFNDEF-DIRECTIVE* whose *constant-expression* contains a *character-constant*.	Aspects of T-behaviour are **implementation-defined**.
6.10.1-5	An *ELIF-DIRECTIVE*.	Such a construct may not be supported by pre-standard implementations thereby impairing **PORTABILITY**.
6.10.1-6	An *IF-DIRECTIVE* whose *constant-expression* denotes the value zero.	Some users believe that programmers are prone to write such constructs in error and may wish to ban or control them in aid of **defensive programming**.

6.10.1-7	An *IF-DIRECTIVE*, *IFDEF-DIRECTIVE* or *IFNDEF-DIRECTIVE* for which there is no matching *ELSE-DIRECTIVE*, *ELIF-DIRECTIVE* or *ENDIF-DIRECTIVE*	Some users believe that programmers are prone to write such constructs in error and may wish to ban or control them in aid of **defensive programming**.
6.10.1-8	An *ELSE-DIRECTIVE*, *ELIF-DIRECTIVE* or *ENDIF-DIRECTIVE* for which there is no matching *IF-DIRECTIVE*, *IFDEF-DIRECTIVE* or *IFNDEF-DIRECTIVE*.	Some users believe that programmers are prone to write such constructs in error and may wish to ban or control them in aid of **defensive programming**.

6.10.2 Source file inclusions

Parasyntax:

INCLUDE-DIRECTIVE = **`# include`** *pp-tokens new-line* ;

Designated constructs:

DCRN	Definition	Rationale
6.10.2-1	An *INCLUDE-DIRECTIVE* that does not contain a *header-name*.	T-behaviour is **undefined**.
6.10.2-2	An *INCLUDE-DIRECTIVE* whose first contained *preprocessing-token* is not a *header-name*.	T-behaviour is **undefined**.
6.10.2-3	An *INCLUDE-DIRECTIVE* whose T-behaviour causes inclusion of the file in which it occurs (recursive inclusion).	T-behaviour is **undefined**.
6.10.2-4	An *INCLUDE-DIRECTIVE* whose first contained *preprocessing-token* is a *STD-HEADER-NAME* that is not a *header-name* for a standard library.	Use of non-standard headers impairs **PORTABILITY**.
6.10.2-5	An *INCLUDE-DIRECTIVE* whose first contained *preprocessing-token* is not a *STD-HEADER-NAME*.	Use of user-defined headers impairs **PORTABILITY**.
6.10.2-6	An *INCLUDE-DIRECTIVE* whose T-behaviour contains the expansion of a macro.	Such a construct may not be supported by pre-standard implementations and its presence impairs **PORTABILITY**.
6.10.2-7	An *INCLUDE-DIRECTIVE* containing more than one *preprocessing-token*, only the first of which is a *header-name*.	Such a construct may not be supported by pre-standard implementations and its presence impairs **PORTABILITY**.

6.10.3 Macro replacement

Parasyntax:

DEFINE-DIRECTIVE = *PLAIN-DEFINE-DIRECTIVE*
| *FLIKE-DEFINE-DIRECTIVE*
| *EMPTY-VAR-FLIKE-DEFINE-DIRECTIVE*
| *VAR-FLIKE-DEFINE-DIRECTIVE* ;

PLAIN-DEFINE-DIRECTIVE = **`# define`** *MACRO-NAME* □
replacement-list new-line ;

FLIKE-DEFINE-DIRECTIVE = **`# define`** *MACRO-NAME* < **`(`** [*identifier-list*]
replacement-list new-line ;

EMPTY-VAR-FLIKE-DEFINE-DIRECTIVE = **`# define`** *identifier* < **`( . . . )`**
replacement-list new-line ;

VAR-FLIKE-DEFINE-DIRECTIVE = **`# define`** *identifier* < **`(`** *identifier-list*
`, . . . )` *replacement-list new-line* ;

Note: Use here of the direct concatenation metasymbol **<** obviates the need for the definition of a nonterminal *lparen* defined to be a left-parentheses without preceding white space.

MACRO-NAME = *identifier* ;

PAREN-REPLACEMENT-LIST = **`(`** *replacement-list* **`)`** ;

Designated constructs:

DCRN	Definition	Rationale
6.10.3-1	A *translation-unit* containing both a *PLAIN-DEFINE-DIRECTIVE* and an *FLIKE-DEFINE-DIRECTIVE* such that the *identifier* of one is the same as the *identifier* of the other.	The presence of such constructs violates a **constraint**.
6.10.3-2	Two or more distinct occurrences of an *FLIKE-DEFINE-DIRECTIVE* that define the sameidentifier as a macro but hav different replacement lists.	The presence of such constructs violates a **constraint**.
6.10.3-3	Two or more distinct occurrences of an *EMPTY-VAR-FLIKE-DEFINE-DIRECTIVE* that define the sameidentifier as a macro but hav different replacement lists.	The presence of such constructs violates a **constraint**.
6.10.3-4	Two or more distinct occurrences of a *VAR-FLIKE-DEFINE-DIRECTIVE* that define the sameidentifier as a macro but hav different replacement lists.	The presence of such constructs violates a **constraint**.
6.10.3-5	Two or more distinct occurrences of a *PLAIN-DEFINE-DIRECTIVE* that define the sameidentifier as a macro but hav different replacement lists.	The presence of such constructs violates a **constraint**.
6.10.3-6	A *replacement-list* of a *PLAIN-DEFINE-DIRECTIVE* or an *FLIKE-DEFINE-DIRECTIVE* that contains the *identifier* **`__VA_ARGS_`**.	The presence of **`__VA_ARGS__`** in such a context violates a **constraint**.
6.10.3-7	An *FLIKE-DEFINE-DIRECTIVE* whose *replacement-list* does not contain **`)`** .	Behaviour is **undefined**.

6.10.3-8	A *replacement-list* containing a sequence of *pp-token* that have the syntactic form of a *DIRECTIVE*.	Behaviour is **undefined**.
6.10.3-9	A *DEFINE-DIRECTIVE* that contains a *preprocessing-token* having the same spelling as a *keyword* or is `defined`.	Behaviour is **undefined**.
6.10.3-10	A non standard *DEFINE-DIRECTIVE* that does not contain an *identifier*.	Behaviour is **undefined**.
6.10.3-11	A *DEFINE-DIRECTIVE* that can be replaced (possibly at a different point in a source file by a definition of an object.	A programmer may have used an object-like macro when an object definition could have been used. Use of an object definition can promote **ANALYSABILITY**.
6.10.3-12	A *translation-unit* containing two distinct occurrences of a *PLAIN-DEFINE-DIRECTIVE* such that the *identifiers* of both instances are the same.	Such a construct may lead to undefined behaviour under pre-standard implementations or implementations that conform to earlier version of the base language standard. Its presence therefore impairs **PORTABILITY**.
6.10.3-13	A *translation-unit* containing two distinct occurrences of an *FLIKE-DEFINE-DIRECTIVE* such that the *identifiers* of both instances are the same.	Such a construct may lead to undefined behaviour under pre-standard implementations or implementations that conform to earlier version of the base language standard. Its presence therefore impairs **PORTABILITY**.
6.10.3-14	A *replacement-list* that is not a *PAREN-REPLACEMENT-LIST*.	Some users believe that programmers are prone to make errors when they do not parenthesise replacement lists. Accordingly they may wish to ban or control such usage in aid of **defensive programming**.
6.10.3-15	An *FLIKE-DEFINE-DIRECTIVE* whose *identifier-list* contains distinct occurrences of an *identifier* that have the same spelling.	Some users believe that programmers are prone to make errors when they do not parenthesise replacement lists. Accordingly they may wish to ban or control such usage in aid of **defensive programming**.
6.10.3-16	A macro expansion that causes the generation of a construct containing a *SIDE-EFFECTIVE-OPERATOR*.	Some users believe that programmers are prone to make errors when using such constructs and may wish to ban or control their use in aid of **defensive programming**.
6.10.3-17	A macro expansion that causes the generation of a construct whose E-behaviour contains sequence point.	Some users believe that programmers are prone to make errors when using such constructs and may wish to ban or control their use in aid of **defensive programming**.
6.10.3-18	An *EMPTY-VAR-FLIKE-DEFINE-DIRECTIVE* or a *VAR-FLIKE-DEFINE-DIRECTIVE*	Such a construct may lead to undefined behaviour under pre-standard implementations or implementations that conform to earlier version of the base language standard. Their presence therefore impairs **PORTABILITY**.

6.10.3.1 Argument substitution

Parasyntax:

MACRO-INVOCATION = *MACRO-NAME* [(*INVOCATION-TAIL*];

Designated constructs:

DCRN	Definition	Rationale
6.10.3.1-1	A *MACRO-INVOCATION whose INVOCATION-TAIL* does not begin with an *identifier-list* that contains no fewer identifiers than occur in the identifier-list of its corresponding *FLIKE-DEFINE-DRECITIVE*, . *EMPTY-VAR-FLIKE-DEFINE-DIRECTIVE* or a *VAR-FLIKE-DEFINE-DIRECTIVE.*	Such a construct violates a **constraint**.
6.10.3.1-2	A *MACRO-INVOCATION whose INVOCATION-TAIL* does not begin with an *identifier-list* that contains more identifiers than occur in the identifier-list of its corresponding *FLIKE-DEFINE-DRECITIVE.*	Such a construct violates a **constraint**.
6.10.3.1-3	A *MACRO-INVOCATION* whose *INVOCATION-TAIL* does not begin with an *identifier-list*.	Behaviour is **undefined**.
6.10.3.1-4	A *MACRO-INVOCATION* whose *INVOCATION-TAIL* does not end with a) .	Behaviour is **undefined**.
6.10.3.1-5	A *MACRO-INVOCATION* whose T-behaviour creates a further invocation of the same macro (recursive invocation).	Behaviour is **undefined**.
6.10.3.1-6	A *MACRO-INVOCATION* that is not enclosed in parentheses.	Some users believe that programmers are prone to make errors when using such constructs and may wish toe ban or control them in aid of **defensive programming**.

6.10.3.2 The # operator

Designated constructs:

DCRN	Definition	Rationale
6.10.3.2-1	An occurrence of the # preprocessing token other than immediately before a *pp-token* contained by a *replacement-list*.	Such a construct violates a **constraint**.
6.10.3.2-2	An occurrence of the # preprocessing token whose T-behaviour does not generate a *string-literal*.	Behaviour is **undefined**.
6.10.3.2-3	The # preprocessing operator.	Some users believe that programmers are prone to making errors when using this operator and may wish to ban or control such usage in aid of **defensive programming**.

6.10.3.3 The `##` operator

Designated constructs:

DCRN	Definition	Rationale
6.10.3.3-1	An occurrence of the `##` preprocessing token as the first or last *pp-token* in a *replacement-list*.	Such a construct violates a **constraint**.
6.10.3.3-2	An occurrence of the `##` pre-processing operator whose T-behaviour does not generate a *pp-token*.	Behaviour is **undefined**.
6.10.3.3-3	The `##` preprocessing operator.	Some users believe that programmers are prone to making errors when using this operator and may wish to ban or control such usage in aid of **defensive programming**.

6.10.3.4 Rescanning and further replacement (NR)

6.10.3.5 Scope of macro definitions

Parasyntax:

UNDEF-DIRECTIVE = **`# undef`** *identifier new-line* ;

Designated constructs:

DCRN	Definition	Rationale
6.10.3.5-1	A *UNDEF-DIRECTIVE* that contains a *preprocessing-token* having the same spelling as a *keyword* or is `defined`.	Behaviour is **undefined**.
6.10.3.5-2	A non standard *UNDEF-DIRECTIVE* that does not contain an *identifier*.	Behaviour is **undefined**.
6.10.3.5-3	An *UNDEF-DIRECTIVE*.	Some users believe that programmers are prone to making errors when using such a construct and may wish to ban or control such usage in aid of **defensive programming**.

6.10.4 Line control

Parasyntax:

LINE-DIRECTIVE = **`# line`** *LINE-PP-TOKENS new-line* ;

LINE-PP-TOKENS = *LINE-DIG-SEQ*
| *LINE-DIG-SEQ-SCHAR-SEQ*
| *pp-tokens* ~ *LINE-DIG-SEQ*

| *pp-tokens* ~ *LINE-DIG-SEQ-SCHAR-SEQ*;

LINE-DIG-SEQ = *digit-sequence* ;

LINE-DIG-SEQ-SCHAR-SEQ = *digit-sequence* " [*s-char-sequence*] " *;*

Designated constructs:

DCRN	Definition	Rationale
6.10.4-1	A *LINE-DIG-SEQ-SCHAR-SEQ* whose *s-char-sequence* is not a *character-string-literal*.	Such a construct violates a **constraint**.
6.10.4-2	A *LINE-DIG-SEQ* or *LINE-DIG-SEQ-SCHAR-SEQ* whose *digit-sequence* denotes a value outside the range [1, 2147483647]	Behaviour is **undefined**.
6.10.4-3	*A LINE-PP-TOKENS* that does not result after replacement in a *LINE-DIG-SEQ* or a *LINE-DIG-SEQ-SCHAR-SEQ*.	Behaviour is **undefined**.
6.10.4-4	A *LINE-DIRECTIVE*.	Some users believe that programmers are prone to making errors when using such a construct and may wish to ban or control such usage in aid of **defensive programming**.

6.10.5 Error directive

Parasyntax:

ERROR-DIRECTIVE = **`# error`** [*pp-tokens*] *new-line* ;

Designated constructs:

DCRN	Definition	Rationale
6.10.5-1	An *ERROR-DIRECTIVE*.	Some users believe that programmers are prone to making errors when using such a construct and may wish to ban or control such usage in aid of **defensive programming**.

6.10.6 Pragma directive

Parasyntax:

PRAGMA-DIRECTIVE = *STDC-PRAGMA-DIRECTIVE*
| *PLAIN-PRAGMA-DIRECTIVE* ;

STDC-PRAGMA-DIRECTIVE = **`#pragma STDC`** *STDC-PRAGMA-NAME*
on-off-switch ;

STDC-PRAGMA-NAME = **`FP_CONTRACT`**
| **`FENV_ACCESS`**
| **`CX_LIMITED_RANGE`** ;

```
on-off-switch                 =       ON | OFF | DEFAULT ;

PLAIN-PRAGMA-DIRECTIVE        =       (# pragma [ pp-tokens ] new-line )
                                      ~
                                      STDC-PRAGMA-DIRECTIVE ;
```

Designated constructs:

DCRN	Definition	Rationale
6.10.6-1	A *PLAIN-PRAGMA-DIRECTIVE*.	The effects of such directives are **implementation-dependent**.

6.10.7 Null directive

Parasyntax:

```
NULL-DIRECTIVE                =       #   new-line ;
```

Designated constructs:

DCRN	Definition	Rationale
6.10.7-1	A *NULL-DIRECTIVE*.	Such a directive has no effect and is therefore redundant.

6.10.8 Predefined macro names

Parasyntax:

```
PREDEFINED-MACRO-NAME              =       C90-PREDEFINED-MACRO-NAME
                                   |       C99-PREDEFINED-MACRO-NAME ;

C90-PREDEFINED-MACRO-NAME          =       __DATE__   |__FILE__
      ..........................................................................................
      ..........................................................................................
     |..................................................................................__LINE__ |
     __STDC__
                                   |       __TIME__ ;

C99-PREDEFINED-MACRO-NAME          =       __STDC_HOSTED__
      ..........................................................................................
      ..........................................................................................
     |..........................................................................__STDC_VERSION__
                                   |       __STDC_IEC_559__
                                   |       __STDC_IEC_559_COMPLEX__
                                   |       __STDC_ISO_10646__ ;

UNDEF-DIRECTIVE               =       # undef identifier new-line ;
```

Designated constructs:

DCRN	Definition	Rationale
6.10.8-1	An *UNDEF-DIRECTIVE* whose *identifier* is a *PREDEFINED-MACRO-NAME*.	Behaviour is **undefined**.
6.10.8-2	A *DEFINE-DIRECTIVE* whose *identifier* is a *PREDEFINED-MACRO-NAME*.	Behaviour is **undefined**.

6.10.9 Pragma operator

Parasyntax:

PRAGMA-OPERATOR-EXPRESSION = **`_Pragma (`** *string-literal* **`)`** ;

Designated constructs:

DCRN	Definition	Rationale
6.10.9-1	A *PRAGMA-OPERATOR-EXPRESSION*.	Such constructs may not be supported by implementations conforming to earlier versions of the base language standard and their use impairs **PORTABILITY**.

6.11 Future language directions

6.11.1 Floating types (NR)

6.11.2 Linkages of identifiers (NR)

6.11.3 External names (NR)

6.11.4 Character escape sequences (NR)

6.11.5 Storage-class specifiers (NR)

6.11.6 Function declarators (NR)

6.11.7 Function definitions (NR)

6.11.8 Pragma directives (NR)

6.11.9 Predefined macro names (NR)

7 Library

7.1 Introduction

7.1.1 Definitions of terms (NR)

7.1.2 Standard headers (NR)

7.1.3 Reserved identifiers (NR)

7.1.4 Use of library functions (NR)

7.2 Diagnostics `<assert.h>`

7.2.1 Program diagnostics

7.2.1.1 The assert macro

Designated constructs:

DCRN	Definition	Rationale
7.2.1-1	A *macro-invocation* whose *MACRO-NAME* is **`assert`**.	Behaviour is **implementation-dependent** in freestanding implementations.

7.3 Complex arithmetic <complex.h>

Designated constructs:

DCRN	Definition	Rationale
7.3-1	An *INCLUDE-DIRECTIVE* that causes inclusion of the **`<complex.h>`** header.	The ACCURACY of function provided by this header is implementation-dependent. For critical applications some users may wish to use mathematical libraries for which the accuracy is well characterised.

Note. It may be that some functions provided by the **`<complex.h>`** header of a conforming implementation are of acceptable accuracy while some are not. Accordingly users may wish to control usage at the individual function level. Where this is a possible rationale for other DCRN's in this clause, it is indicated by the abbreviation FSC ACCURACY standing for "Function-specific controls for accuracy".

7.3.1 Introduction (NR)

7.3.2 Conventions (NR)

7.3.3 Branch cuts (NR)

7.3.4 The CX_LIMITED_RANGE pragma

Parasyntax:

CX-LIMITED-RANGE-PRAGMA = **`#pragma STDC CX_LIMITED_RANGE`** *on-off-switch* ;

Designated constructs:

DCRN	Definition	Rationale
7.3.4-1	*A CX-LIMITED-RANGE-PRAGMA*.	Some users of C for numerical applications believe that all but expert numerical programmers are prone to make errors using this pragma and may wish to ban or control its use in aid of **defensive programming**.

7.3.5 Trigonometric functions

Designated constructs:

DCRN	Definition	Rationale
7.3.5.1-1	The *FUNCTION-DESIGNATOR* **`cacos`**	FSC ACCURACY
7.3.5.1-2	The *FUNCTION-DESIGNATOR* **`cacosf`**	FSC ACCURACY
7.3.5.1-3	The *FUNCTION-DESIGNATOR* **`cacosl`**	FSC ACCURACY
7.3.5.2-1	The *FUNCTION-DESIGNATOR* **`casin`**	FSC ACCURACY
7.3.5.2-2	The *FUNCTION-DESIGNATOR* **`casinf`**	FSC ACCURACY
7.3.5.2-3	The *FUNCTION-DESIGNATOR* **`casinl`**	FSC ACCURACY
7.3.5.3-1	The *FUNCTION-DESIGNATOR* **`catan`**	FSC ACCURACY
7.3.5.3-2	The *FUNCTION-DESIGNATOR* **`catanf`**	FSC ACCURACY
7.3.5.3-3	The *FUNCTION-DESIGNATOR* **`catanl`**	FSC ACCURACY
7.3.5.4-1	The *FUNCTION-DESIGNATOR* **`ccos`**	FSC ACCURACY

7.3.5.4-2	The *FUNCTION-DESIGNATOR* `ccosf`	FSC ACCURACY
7.3.5.4-3	The *FUNCTION-DESIGNATOR* `ccosl`	FSC ACCURACY
7.3.5.5-1	The *FUNCTION-DESIGNATOR* `csin`	FSC ACCURACY
7.3.5.5-2	The *FUNCTION-DESIGNATOR* `csinf`	FSC ACCURACY
7.3.5.5-3	The *FUNCTION-DESIGNATOR* `csinl`	FSC ACCURACY
7.3.5.6-1	The *FUNCTION-DESIGNATOR* `ctan`	FSC ACCURACY
7.3.5.6-2	The *FUNCTION-DESIGNATOR* `ctanf`	FSC ACCURACY
7.3.5.6-3	The *FUNCTION-DESIGNATOR* `ctanl`	FSC ACCURACY

7.3.6 Hyperbolic functions

Designated constructs:

DCRN	Definition	Rationale
7.3.6.1-1	The *FUNCTION-DESIGNATOR* `cacosh`	FSC ACCURACY
7.3.6.1-2	The *FUNCTION-DESIGNATOR* `cacoshf`	FSC ACCURACY
7.3.6.1-3	The *FUNCTION-DESIGNATOR* `cacoshl`	FSC ACCURACY
7.3.6.2-1	The *FUNCTION-DESIGNATOR* `casinh`	FSC ACCURACY
7.3.6.2-2	The *FUNCTION-DESIGNATOR* `casinhf`	FSC ACCURACY
7.3.6.2-3	The *FUNCTION-DESIGNATOR* `casinhl`	FSC ACCURACY
7.3.6.3-1	The *FUNCTION-DESIGNATOR* `catanh`	FSC ACCURACY
7.3.6.3-2	The *FUNCTION-DESIGNATOR* `catanhf`	FSC ACCURACY
7.3.6.3-3	The *FUNCTION-DESIGNATOR* `catanhl`	FSC ACCURACY
7.3.6.4-1	The *FUNCTION-DESIGNATOR* `ccosh`	FSC ACCURACY
7.3.6.4-2	The *FUNCTION-DESIGNATOR* `ccoshf`	FSC ACCURACY
7.3.6.4-3	The *FUNCTION-DESIGNATOR* `ccoshl`	FSC ACCURACY
7.3.6.5-1	The *FUNCTION-DESIGNATOR* `csinh`	FSC ACCURACY
7.3.6.5-2	The *FUNCTION-DESIGNATOR* `csinhf`	FSC ACCURACY
7.3.6.5-3	The *FUNCTION-DESIGNATOR* `csinhl`	FSC ACCURACY
7.3.6.6-1	The *FUNCTION-DESIGNATOR* `ctanh`	FSC ACCURACY
7.3.6.6-2	The *FUNCTION-DESIGNATOR* `ctanhf`	FSC ACCURACY
7.3.6.6-3	The *FUNCTION-DESIGNATOR* `ctanhl`	FSC ACCURACY

7.3.7 Exponential and logarithmic functions

Designated constructs:

DCRN	Definition	Rationale
7.3.7.1-1	The *FUNCTION-DESIGNATOR* **`cexp`**	FSC ACCURACY
7.3.7.1-2	The *FUNCTION-DESIGNATOR* **`cexpf`**	FSC ACCURACY
7.3.7.1-3	The *FUNCTION-DESIGNATOR* **`cexpl`**	FSC ACCURACY
7.3.7.2-1	The *FUNCTION-DESIGNATOR* **`clog`**	FSC ACCURACY
7.3.7.2-2	The *FUNCTION-DESIGNATOR* **`clogf`**	FSC ACCURACY
7.3.7.2-3	The *FUNCTION-DESIGNATOR* **`clogl`**	FSC ACCURACY

7.3.8 Power and absolute-value functions

Designated constructs:

DCRN	Definition	Rationale
7.3.8.1-1	The *FUNCTION-DESIGNATOR* **`cabs`**	FSC ACCURACY
7.3.8.1-2	The *FUNCTION-DESIGNATOR* **`cabsf`**	FSC ACCURACY
7.3.8.1-3	The *FUNCTION-DESIGNATOR* **`cabsl`**	FSC ACCURACY
7.3.8.2-1	The *FUNCTION-DESIGNATOR* **`cpow`**	FSC ACCURACY
7.3.8.2-2	The *FUNCTION-DESIGNATOR* **`cpowf`**	FSC ACCURACY
7.3.8.2-3	The *FUNCTION-DESIGNATOR* **`cpowl`**	FSC ACCURACY
7.3.8.3-1	The *FUNCTION-DESIGNATOR* **`csqrt`**	FSC ACCURACY
7.3.8.3-2	The *FUNCTION-DESIGNATOR* **`csqrtf`**	FSC ACCURACY
7.3.8.3-3	The *FUNCTION-DESIGNATOR* **`csqrtl`**	FSC ACCURACY

7.3.9 Manipulation functions

Designated constructs:

DCRN	Definition	Rationale
7.3.9.1-1	The *FUNCTION-DESIGNATOR* **`carg`**	FSC ACCURACY
7.3.9.1-2	The *FUNCTION-DESIGNATOR* **`cargf`**	FSC ACCURACY
7.3.9.1-3	The *FUNCTION-DESIGNATOR* **`cargl`**	FSC ACCURACY
7.3.9.2-1	The *FUNCTION-DESIGNATOR* **`cimag`**	FSC ACCURACY
7.3.9.2-2	The *FUNCTION-DESIGNATOR* **`cimagf`**	FSC ACCURACY
7.3.9.2-3	The *FUNCTION-DESIGNATOR* **`cimagl`**	FSC ACCURACY
7.3.9.3-1	The *FUNCTION-DESIGNATOR* **`conj`**	FSC ACCURACY
7.3.9.3-2	The *FUNCTION-DESIGNATOR* **`conjf`**	FSC ACCURACY
7.3.9.3-3	The *FUNCTION-DESIGNATOR* **`conjl`**	FSC ACCURACY

7.3.9.4-1	The *FUNCTION-DESIGNATOR* `cproj`	FSC ACCURACY
7.3.9.4-2	The *FUNCTION-DESIGNATOR* `cprojf`	FSC ACCURACY
7.3.9.4-3	The *FUNCTION-DESIGNATOR* `cprojl`	FSC ACCURACY
7.3.9.5-1	The *FUNCTION-DESIGNATOR* `creal`	FSC ACCURACY
7.3.9.5-2	The *FUNCTION-DESIGNATOR* `crealf`	FSC ACCURACY
7.3.9.5-3	The *FUNCTION-DESIGNATOR* `creall`	FSC ACCURACY

7.4 Character handling `<ctype.h>`

Designated constructs:

DCRN	Definition	Rationale
7.4-1	An *INCLUDE-DIRECTIVE* that causes inclusion of the **`<ctype.h>`** header.	The functions provided by this header may not exhibit sufficient **ACCURACY** in reflecting the conventions in specific locales. Accordingly some users may wish to use a library that does reflect local conventions.

Note. It may be that some functions provided by the **`<ctype.h>`** header of a conforming implementation do accurately reflect local conventions while some do not. Accordingly users may wish to control usage at the individual function level. Where this is a possible rationale for other DCRN's in this clause, it is indicated by the abbreviation FSC **ACCURACY** standing for "Function-specific controls for accuracy".

7.4.1 Character classification functions

Designated constructs:

DCRN	Definition	Rationale
7.4.1.1-1	The *FUNCTION-DESIGNATOR* **`isalnum`**	FSC **ACCURACY**
7.4.1.2-1	The *FUNCTION-DESIGNATOR* **`isalph`**	FSC **ACCURACY**
7.4.1.3-1	The *FUNCTION-DESIGNATOR* **`isblank`**	FSC **ACCURACY**
7.4.1.4-1	The *FUNCTION-DESIGNATOR* **`iscntrl`**	FSC **ACCURACY**
7.4.1.5-1	The *FUNCTION-DESIGNATOR* **`isdigit`**	FSC **ACCURACY**
7.4.1.6-1	The *FUNCTION-DESIGNATOR* **`isgraph`**	FSC **ACCURACY**
7.4.1.7-1	The *FUNCTION-DESIGNATOR* **`islower`**	FSC **ACCURACY**
7.4.1.8-1	The *FUNCTION-DESIGNATOR* **`isprint`**	FSC **ACCURACY**
7.4.1.9-1	The *FUNCTION-DESIGNATOR* **`ispunct`**	FSC **ACCURACY**
7.4.1.10-1	The *FUNCTION-DESIGNATOR* **`isspace`**	FSC **ACCURACY**
7.4.1.11-1	The *FUNCTION-DESIGNATOR* **`isupper`**	FSC **ACCURACY**
7.4.1.12-1	The *FUNCTION-DESIGNATOR* **`isxdigit`**	FSC **ACCURACY**

7.4.2 Character case mapping function

Designated constructs:

DCRN	Definition	Rationale
7.4.2.1-1	The *FUNCTION-DESIGNATOR* **`tolower`**	FSC **ACCURACY**
7.4.2.2-1	The *FUNCTION-DESIGNATOR* **`toupper`**	FSC **ACCURACY**

7.5 Errors <errno.h>

Designated constructs:

DCRN	Definition	Rationale
7.5-1	An *INCLUDE-DIRECTIVE* that causes inclusion of the **`<errno.h>`** header.	Many aspects of **`errno`** and the values to which it may be set are sufficiently **implementation-dependent** that its use can impair **PORTABILITY**.
7.5-2	The *identifier* **`errno`**.	As for 7.5-1
7.5-3	The *MACRO-NAME* **`errno`**.	As for 7.5-1
7.5-4	The *MACRO-NAME* **`EDOM`**.	As for 7.5-1
7.5-5	The *MACRO-NAME* **`EILSEQ`**.	As for 7.5-1
7.5-6	The *MACRO-NAME* **`ERANGE`**.	As for 7.5-1

7.6 Floating-point environment `<fenv.h>`

Designated constructs:

DCRN	Definition	Rationale
7.6-1	An *INCLUDE-DIRECTIVE* that causes inclusion of the `<fenv.h>` header.	Many aspects of the facilities provided by `<fenv.h>` are **implementation-dependent**. It may also not be supported by implementations conforming to earlier version of the base language standard so its use impairs **PORTABILITY**.
7.6-2	The *typedef-name* `fenv_t`.	As for 7.6-1
7.6-3	The *typedef-name* `fexcept_t`.	As for 7.6-1
7.6-4	The *MACRO-NAME* `FE_DIVBYZERO`.	As for 7.6-1
7.6-5	The *MACRO-NAME* `FE_INEXACT`.	As for 7.6-1
7.6-6	The *MACRO-NAME* `FE_INVALID`.	As for 7.6-1
7.6-7	The *MACRO-NAME* `FE_OVERFLOW`.	As for 7.6-1
7.6-8	The *MACRO-NAME* `FE_UNDERFLOW`.	As for 7.6-1
7.6-9	The *MACRO-NAME* `FE_ALL_EXCEPT`.	As for 7.6-1
7.6-10	The *MACRO-NAME* `FE-DOWNWARD`	As for 7.6-1
7.6-11	The *MACRO-NAME* `FE_TONEAREST`	As for 7.6-1
7.6-12	The *MACRO-NAME* `FE_TOWARDZERO`	As for 7.6-1
7.6-13	The *MACRO-NAME* `FE_UPWARD`	As for 7.6-1
7.6-14	The *MACRO-NAME* `FE_DLF_ENV`	As for 7.6-1

7.6.1 The `FENV_ACCESS` pragma

Parasyntax:

FENV-ACCESS-PRAGMA = `#pragma STDC FENV_ACCESS` *on-off-switch* ;

Designated constructs:

DCRN	Definition	Rationale
7.6.1-1	An *FENV_ACCESS_PRAGMA*.	Some users of C for numerical applications believe that all but expert numerical programmers are prone to make errors using this pragma owing to the degree to which aspects of the floating-point environment are **implementation-dependent**. Such users may wish to ban or control its use in aid of **defensive programming**.

7.6.2 Floating-point exceptions

Designated constructs:

DCRN	Definition	Rationale
7.6.2-1	The *FUNCTION-DESIGNATOR* **`fclearexcept`**	As for 7.6-1
7.6.2-2	The *FUNCTION-DESIGNATOR* **`fegetexceptflag`**	As for 7.6-1
7.6.2-3	The *FUNCTION-DESIGNATOR* **`feraiseexcept`**	As for 7.6-1
7.6.2-4	The *FUNCTION-DESIGNATOR* **`fesetexceptflag`**	As for 7.6-1
7.6.2-5	The *FUNCTION-DESIGNATOR* **`fetestexceptflag`**	As for 7.6-1

7.6.3 Rounding

Designated constructs:

DCRN	Definition	Rationale
7.6.3-1	The *FUNCTION-DESIGNATOR* **`fegetround`**	As for 7.6-1
7.6.3-2	The *FUNCTION-DESIGNATOR* **`fesetround`**	As for 7.6-1

7.6.4 Environment

Designated constructs:

DCRN	Definition	Rationale
7.6.4-1	The *FUNCTION-DESIGNATOR* **`fegetenv`**	As for 7.6-1
7.6.4-2	The *FUNCTION-DESIGNATOR* **`feholdexcept`**	As for 7.6-1
7.6.4-3	The *FUNCTION-DESIGNATOR* **`fesetenv`**	As for 7.6-1
7.6.4-4	The *FUNCTION-DESIGNATOR* **`feupdateenv`**	As for 7.6-1

7.7 Characteristics of floating types `<float.h>` (NR)

This page intentionally left blank.

7.8 Format conversion of integer types `<inttypes.h>`

Designated constructs:

DCRN	Definition	Rationale
7.8-1	An *INCLUDE-DIRECTIVE* that causes inclusion of the **`<inttypes.h>`** header.	The **`<inttypes.h>`** header provides further support for features provided by the **`<stdint.h>`** header and thereby shares many **implementation dependent** characteristic of **`<stdint.h>.`**
7.8-2	The *typedef-name* **`ismaxdiv_t`**.	As for 7.8-1

7.8.1 Macros for format specifiers

Designated constructs:

DCRN	Definition	Rationale
7.8.1-1	The *MACRO-NAME* **`PRId`***N*	As for 7.8-1
7.8.1-2	The *MACRO-NAME* **`PRIdLEAST`***N*	As for 7.8-1
7.8.1-3	The *MACRO-NAME* **`PRIdFAST`***N*	As for 7.8-1
7.8.1-4	The *MACRO-NAME* **`PRIdMAX`**	As for 7.8-1
7.8.1-5	The *MACRO-NAME* **`PRIdPTR`**	As for 7.8-1
7.8.1-6	The *MACRO-NAME* **`PRIi`***N*	As for 7.8-1
7.8.1-7	The *MACRO-NAME* **`PRIiLEAST`***N*	As for 7.8-1
7.8.1-8	The *MACRO-NAME* **`PRIiFAST`***N*	As for 7.8-1
7.8.1-9	The *MACRO-NAME* **`PRIiMAX`**	As for 7.8-1
7.8.1-10	The *MACRO-NAME* **`PRIiPTR`**	As for 7.8-1
7.8.1-11	The *MACRO-NAME* **`PRIo`***N*	As for 7.8-1
7.8.1-12	The *MACRO-NAME* **`PRIoLEAST`***N*	As for 7.8-1
7.8.1-13	The *MACRO-NAME* **`PRIoFAST`***N*	As for 7.8-1
7.8.1-14	The *MACRO-NAME* **`PRIoMAX`**	As for 7.8-1
7.8.1-15	The *MACRO-NAME* **`PRIoPTR`**	As for 7.8-1
7.8.1-16	The *MACRO-NAME* **`PRIu`***N*	As for 7.8-1
7.8.1-17	The *MACRO-NAME* **`PRIuLEAST`***N*	As for 7.8-1
7.8.1-18	The *MACRO-NAME* **`PRIuFAST`***N*	As for 7.8-1
7.8.1-19	The *MACRO-NAME* **`PRIuMAX`**	As for 7.8-1

7.8.1-20	The *MACRO-NAME* **PRIuPTR**	As for 7.8-1
7.8.1-21	The *MACRO-NAME* **PRIx***N*	As for 7.8-1
7.8.1-22	The *MACRO-NAME* **PRIxLEAST***N*	As for 7.8-1
7.8.1-23	The *MACRO-NAME* **PRIxFAST***N*	As for 7.8-1
7.8.1-24	The *MACRO-NAME* **PRIxMAX**	As for 7.8-1
7.8.1-25	The *MACRO-NAME* **PRIxPTR**	As for 7.8-1
7.8.1-26	The *MACRO-NAME* **PRIX***N*	As for 7.8-1
7.8.1-27	The *MACRO-NAME* **PRIXLEAST***N*	As for 7.8-1
7.8.1-28	The *MACRO-NAME* **PRIXFAST***N*	As for 7.8-1
7.8.1-29	The *MACRO-NAME* **PRIXMAX**	As for 7.8-1
7.8.1-30	The *MACRO-NAME* **PRIXPTR**	As for 7.8-1
7.8.1-31	The *MACRO-NAME* **SCNd***N*	As for 7.8-1
7.8.1-32	The *MACRO-NAME* **SCNdLEAST***N*	As for 7.8-1
7.8.1-33	The *MACRO-NAME* **SCNdFAST***N*	As for 7.8-1
7.8.1-34	The *MACRO-NAME* **SCNdMAX**	As for 7.8-1
7.8.1-35	The *MACRO-NAME* **SCNdPTR**	As for 7.8-1
7.8.1-36	The *MACRO-NAME* **SCNi***N*	As for 7.8-1
7.8.1-37	The *MACRO-NAME* **SCNiLEAST***N*	As for 7.8-1
7.8.1-38	The *MACRO-NAME* **SCNiFAST***N*	As for 7.8-1
7.8.1-39	The *MACRO-NAME* **SCNiMAX**	As for 7.8-1
7.8.1-40	The *MACRO-NAME* **SCNiPTR**	As for 7.8-1
7.8.1-41	The *MACRO-NAME* **SCNo***N*	As for 7.8-1
7.8.1-42	The *MACRO-NAME* **SCNoLEAST***N*	As for 7.8-1
7.8.1-43	The *MACRO-NAME* **SCNoFAST***N*	As for 7.8-1
7.8.1-44	The *MACRO-NAME* **SCNoMAX**	As for 7.8-1
7.8.1-45	The *MACRO-NAME* **SCNoPTR**	As for 7.8-1
7.8.1-46	The *MACRO-NAME* **SCNu***N*	As for 7.8-1
7.8.1-47	The *MACRO-NAME* **SCNuLEAST***N*	As for 7.8-1
7.8.1-48	The *MACRO-NAME* **SCNuFAST***N*	As for 7.8-1

7.8.1-49	The *MACRO-NAME* **SCNuMAX**	As for 7.8-1
7.8.1-50	The *MACRO-NAME* **SCNuPTR**	As for 7.8-1
7.8.1-51	The *MACRO-NAME* **SCNx***N*	As for 7.8-1
7.8.1-52	The *MACRO-NAME* **SCNxLEAST***N*	As for 7.8-1
7.8.1-53	The *MACRO-NAME* **SCNxFAST***N*	As for 7.8-1
7.8.1-54	The *MACRO-NAME* **SCNxMAX**	As for 7.8-1
7.8.1-55	The *MACRO-NAME* **SCNxPTR**	As for 7.8-1

7.8.2 Functions for greatest-width integer types

Designated constructs:

DCRN	Definition	Rationale
7.8.2-1	The *FUNCTION-DESIGNATOR* **bimaxabs**	As for 7.8-1
7.8.2-2	The *FUNCTION-DESIGNATOR* **imaxdiv**	As for 7.8-1
7.8.2-3	The *FUNCTION-DESIGNATOR* **strtoimax**	As for 7.8-1
7.8.2-4	The *FUNCTION-DESIGNATOR* **strtoumax**	As for 7.8-1
7.8.2-5	The *FUNCTION-DESIGNATOR* **wcstoimax**	As for 7.8-1
7.8.2-6	The *FUNCTION-DESIGNATOR* **wcstoumax**	As for 7.8-1

7.9 Alternative spellings `<iso646.h>`

Designated constructs:

DCRN	Definition	Rationale
7.9-1	An *INCLUDE-DIRECTIVE* that causes inclusion of the **`<iso646.h>`** header.	This header may not be supported by implementation conforming to earlier version of the base language standard, thereby impairing **PORTABILITY**.
7.9-2	The *MACRO-NAME* **`and`**	As for 7.9-1
7.9-3	The *MACRO-NAME* **`and-eq`**	As for 7.9-1
7.9-4	The *MACRO-NAME* **`bitand`**	As for 7.9-1
7.9-5	The *MACRO-NAME* **`bitor`**	As for 7.9-1
7.9-6	The *MACRO-NAME* **`compl`**	As for 7.9-1
7.9-7	The *MACRO-NAME* **`not`**	As for 7.9-1
7.9-8	The *MACRO-NAME* **`not_eq`**	As for 7.9-1
7.9-9	The *MACRO-NAME* **`or`**	As for 7.9-1
7.9-10	The *MACRO-NAME* **`or_eq`**	As for 7.9-1
7.9-11	The *MACRO-NAME* **`xor`**	As for 7.9-1
7.9-12	The *MACRO-NAME* **`xor_eq`**	As for 7.9-1

This page intentionally left blank.

7.11 Localisation

Designated constructs:

DCRN	Definition	Rationale
7.11-1	An *INCLUDE-DIRECTIVE* that causes inclusion of the **`<locale.h>`** header.	Most aspects of locales are **implementation-dependent**.

7.11.1 Locale control

Designated constructs:

DCRN	Definition	Rationale
7.11.1.1-1	The *FUNCTION-DESIGNATOR* **`setlocale`**.	As for 7.11-1

7.11.2 Numeric formatting convention enquiry

7.11.2.1 The localeconv function

Designated constructs:

DCRN	Definition	Rationale
7.11.2.1-1	The *FUNCTION-DESIGNATOR* **`localeconv`**.	As for 7.11-1

7.12 Mathematics <math.h>

Designated constructs:

DCRN	Definition	Rationale
7.12-1	An *INCLUDE-DIRECTIVE* that causes inclusion of the **`<math.h>`** header.	Some of the provisions of C99 make certain aspects of the mathematical functions significantly **implementation-dependent**. Further, the mathematical functions and macros provided by any particular implementation do not necessarily exhibit sufficient **ACCURACY** for critical applications.
7.12-4	The *type-name* **`float_t`**.	As for 7.12-1
7.12-5	The *type-name* **`double_t`**.	As for 7.12-1
7.12-6	The *MACRO-NAME* **`HUGE_VAL`**	As for 7.12-1
7.12-7	The *MACRO-NAME* **`HUGE_VALF`**	As for 7.12-1
7.12-8	The *MACRO-NAME* **`HUGE_VALL`**	As for 7.12-1
7.12-9	The *MACRO-NAME* **`INFINITY`**	As for 7.12-1
7.12-10	The *MACRO-NAME* **`NAN`**	As for 7.12-1
7.12-11	The *MACRO-NAME* **`FP_INFINITE`**	As for 7.12-1
7.12-12	The *MACRO-NAME* **`FP_NAN`**	As for 7.12-1
7.12-13	The *MACRO-NAME* **`FP_NORMAL`**	As for 7.12-1
7.12-14	The *MACRO-NAME* **`FP_SUBNORMAL`**	As for 7.12-1
7.12-15	The *MACRO-NAME* **`FP_ZERO`**	As for 7.12-1
7.12-16	The *MACRO-NAME* **`FP_FAST_FMA`**	As for 7.12-1
7.12-17	The *MACRO-NAME* **`FP_FAST_FMAF`**	As for 7.12-1
7.12-18	The *MACRO-NAME* **`FP_FAST_FMAL`**	As for 7.12-1
7.12-19	The *MACRO-NAME* **`FP_ILOGB0`**	As for 7.12-1
7.12-20	The *MACRO-NAME* **`FP_ILOGBNAN`**	As for 7.12-1
7.12-21	The *MACRO-NAME* **`MATH_ERRNO`**	As for 7.12-1
7.12-22	The *MACRO-NAME* **`MATH_ERREXCEPT`**	As for 7.12-1
7.12-23	The *MACRO-NAME* **`math_errhandling`**	As for 7.12-1

7.12.1 Treatment of error conditions (NR)

7.12.2 The `FP_CONTRACT` pragma

Parasyntax:

FP-CONTRACT-PRAGMA = **`#pragma STDC FP_CONTRACT`** *on-off-switch* ;

Designated constructs:

DCRN	Definition	Rationale
7.12.2-1	An *FP_CONTRACT_PRAGMA*.	As for 7.12-1

7.12.3 Classification macros

Designated constructs:

DCRN	Definition	Rationale
7.12.3.1-1	The *MACRO-NAME* `fpclassify`	As for 7.12-1
7.12.3.2-1	The *MACRO-NAME* `isfinite`	As for 7.12-1
7.12.3.3-1	The *MACRO-NAME* `isint`	As for 7.12-1
7.12.3.4-1	The *MACRO-NAME* `isnan`	As for 7.12-1
7.12.3.5-1	The *MACRO-NAME* `isnormal`	As for 7.12-1
7.12.3.6-1	The *MACRO-NAME* `signbit`	As for 7.12-1

7.12.4 Trigonometric functions

Designated constructs:

DCRN	Definition	Rationale
7.12.4.1-1	The *FUNCTION-DESIGNATOR* `acos`	As for 7.12-1
7.12.4.1-2	The *FUNCTION-DESIGNATOR* `acosf`	As for 7.12-1
7.12.4.1-3	The *FUNCTION-DESIGNATOR* `acosl`	As for 7.12-1
7.12.4.2-1	The *FUNCTION-DESIGNATOR* `asin`	As for 7.12-1
7.12.4.2-2	The *FUNCTION-DESIGNATOR* `asinf`	As for 7.12-1
7.12.4.2-3	The *FUNCTION-DESIGNATOR* `asinl`	As for 7.12-1
7.12.4.3-1	The *FUNCTION-DESIGNATOR* `atan`	As for 7.12-1
7.12.4.3-2	The *FUNCTION-DESIGNATOR* `atanf`	As for 7.12-1
7.12.4.3-3	The *FUNCTION-DESIGNATOR* `atanl`	As for 7.12-1
7.12.4.4-1	The *FUNCTION-DESIGNATOR* `atan2`	As for 7.12-1
7.12.4.4-2	The *FUNCTION-DESIGNATOR* `atan2f`	As for 7.12-1
7.12.4.4-3	The *FUNCTION-DESIGNATOR* `atan2l`	As for 7.12-1
7.12.4.5-1	The *FUNCTION-DESIGNATOR* `cos`	As for 7.12-1
7.12.4.5-2	The *FUNCTION-DESIGNATOR* `cosf`	As for 7.12-1
7.12.4.5-3	The *FUNCTION-DESIGNATOR* `cosl`	As for 7.12-1
7.12.4.6-1	The *FUNCTION-DESIGNATOR* `sin`	As for 7.12-1
7.12.4.6-2	The *FUNCTION-DESIGNATOR* `sinf`	As for 7.12-1
7.12.4.6-3	The *FUNCTION-DESIGNATOR* `sinl`	As for 7.12-1
7.12.4.7-1	The *FUNCTION-DESIGNATOR* `tan`	As for 7.12-1
7.12.4.7-2	The *FUNCTION-DESIGNATOR* `tanf`	As for 7.12-1
7.12.4.7-3	The *FUNCTION-DESIGNATOR* `tanl`	As for 7.12-1

7.12.5 Hyperbolic functions

Designated constructs:

DCRN	Definition	Rationale
7.12.5.1-1	The *FUNCTION-DESIGNATOR* **`acosh`**	As for 7.12-1
7.12.5.1-2	The *FUNCTION-DESIGNATOR* **`acoshf`**	As for 7.12-1
7.12.5.1-3	The *FUNCTION-DESIGNATOR* **`acoshl`**	As for 7.12-1
7.12.5.2-1	The *FUNCTION-DESIGNATOR* **`asinh`**	As for 7.12-1
7.12.5.2-2	The *FUNCTION-DESIGNATOR* **`asinhf`**	As for 7.12-1
7.12.5.2-3	The *FUNCTION-DESIGNATOR* **`asinhl`**	As for 7.12-1
7.12.5.3-1	The *FUNCTION-DESIGNATOR* **`atanh`**	As for 7.12-1
7.12.5.3-2	The *FUNCTION-DESIGNATOR* **`atanhf`**	As for 7.12-1
7.12.5.3-3	The *FUNCTION-DESIGNATOR* **`atanhl`**	As for 7.12-1
7.12.5.4-1	The *FUNCTION-DESIGNATOR* **`cosh`**	As for 7.12-1
7.12.5.4-2	The *FUNCTION-DESIGNATOR* **`coshf`**	As for 7.12-1
7.12.5.4-3	The *FUNCTION-DESIGNATOR* **`coshl`**	As for 7.12-1
7.12.5.5-1	The *FUNCTION-DESIGNATOR* **`sinh`**	As for 7.12-1
7.12.5.5-2	The *FUNCTION-DESIGNATOR* **`sinhf`**	As for 7.12-1
7.12.5.5-3	The *FUNCTION-DESIGNATOR* **`sinhl`**	As for 7.12-1
7.12.5.6-1	The *FUNCTION-DESIGNATOR* **`tanh`**	As for 7.12-1
7.12.5.6-2	The *FUNCTION-DESIGNATOR* **`tanhf`**	As for 7.12-1
7.12.5.6-3	The *FUNCTION-DESIGNATOR* **`tanhl`**	As for 7.12-1

7.12.6 Exponential and logarithmic functions

Designated constructs:

DCRN	Definition	Rationale
7.12.6.1-1	The *FUNCTION-DESIGNATOR* **`exp`**	As for 7.12-1
7.12.6.1-2	The *FUNCTION-DESIGNATOR* **`expf`**	As for 7.12-1
7.12.6.1-3	The *FUNCTION-DESIGNATOR* **`expl`**	As for 7.12-1
7.12.6.2-1	The *FUNCTION-DESIGNATOR* **`exp2`**	As for 7.12-1
7.12.6.2-2	The *FUNCTION-DESIGNATOR* **`exp2f`**	As for 7.12-1
7.12.6.2-3	The *FUNCTION-DESIGNATOR* **`exp2l`**	As for 7.12-1
7.12.6.3-1	The *FUNCTION-DESIGNATOR* **`expm1`**	As for 7.12-1
7.12.6.3-2	The *FUNCTION-DESIGNATOR* **`expm1f`**	As for 7.12-1
7.12.6.3-3	The *FUNCTION-DESIGNATOR* **`expm1l`**	As for 7.12-1
7.12.6.4-1	The *FUNCTION-DESIGNATOR* **`frexp`**	As for 7.12-1

7.12.6.4-2	The *FUNCTION-DESIGNATOR* **frexpf**	As for 7.12-1
7.12.6.4-3	The *FUNCTION-DESIGNATOR* **frexpl**	As for 7.12-1
7.12.6.5-1	The *FUNCTION-DESIGNATOR* **ilogb**	As for 7.12-1
7.12.6.5-2	The *FUNCTION-DESIGNATOR* **ilogbf**	As for 7.12-1
7.12.6.5-3	The *FUNCTION-DESIGNATOR* **ilogbl**	As for 7.12-1
7.12.6.6-1	The *FUNCTION-DESIGNATOR* **ldexp**	As for 7.12-1
7.12.6.6-2	The *FUNCTION-DESIGNATOR* **ldexpf**	As for 7.12-1
7.12.6.6-3	The *FUNCTION-DESIGNATOR* **ldexpl**	As for 7.12-1
7.12.6.7-1	The *FUNCTION-DESIGNATOR* **log**	As for 7.12-1
7.12.6.7-2	The *FUNCTION-DESIGNATOR* **logf**	As for 7.12-1
7.12.6.7-3	The *FUNCTION-DESIGNATOR* **logl**	As for 7.12-1
7.12.6.8-1	The *FUNCTION-DESIGNATOR* **log10**	As for 7.12-1
7.12.6.8-2	The *FUNCTION-DESIGNATOR* **log10f**	As for 7.12-1
7.12.6.8-3	The *FUNCTION-DESIGNATOR* **log10l**	As for 7.12-1
7.12.6.9-1	The *FUNCTION-DESIGNATOR* **log1p**	As for 7.12-1
7.12.6.9-2	The *FUNCTION-DESIGNATOR* **log1pf**	As for 7.12-1
7.12.6.9-3	The *FUNCTION-DESIGNATOR* **log1pl**	As for 7.12-1
7.12.6.10-1	The *FUNCTION-DESIGNATOR* **log2**	As for 7.12-1
7.12.6.10-2	The *FUNCTION-DESIGNATOR* **log2f**	As for 7.12-1
7.12.6.10-3	The *FUNCTION-DESIGNATOR* **log2l**	As for 7.12-1
7.12.6.11-1	The *FUNCTION-DESIGNATOR* **logb**	As for 7.12-1
7.12.6.11-2	The *FUNCTION-DESIGNATOR* **logbf**	As for 7.12-1
7.12.6.11-3	The *FUNCTION-DESIGNATOR* **logbl**	As for 7.12-1
7.12.6.12-1	The *FUNCTION-DESIGNATOR* **modf**	As for 7.12-1
7.12.6.12-2	The *FUNCTION-DESIGNATOR* **modff**	As for 7.12-1
7.12.6.12-3	The *FUNCTION-DESIGNATOR* **modfl**	As for 7.12-1
7.12.6.13-1	The *FUNCTION-DESIGNATOR* **scalbn**	As for 7.12-1
7.12.6.13-2	The *FUNCTION-DESIGNATOR* **scalbnf**	As for 7.12-1
7.12.6.13-3	The *FUNCTION-DESIGNATOR* **scalbnl**	As for 7.12-1
7.12.6.13-4	The *FUNCTION-DESIGNATOR* **scalbln**	As for 7.12-1
7.12.6.13-5	The *FUNCTION-DESIGNATOR* **scalblnf**	As for 7.12-1
7.12.6.13-6	The *FUNCTION-DESIGNATOR* **scalblnl**	As for 7.12-1

7.12.7 Power and absolute value functions

Designated constructs:

DCRN	**Definition**	**Rationale**
7.12.7.1-1	The *FUNCTION-DESIGNATOR* **`cbrt`**	As for 7.12-1
7.12.7.1-2	The *FUNCTION-DESIGNATOR* **`cbrtf`**	As for 7.12-1
7.12.7.1-3	The *FUNCTION-DESIGNATOR* **`cbrtl`**	As for 7.12-1
7.12.7.2-1	The *FUNCTION-DESIGNATOR* **`fabs`**	As for 7.12-1
7.12.7.2-2	The *FUNCTION-DESIGNATOR* **`fabsf`**	As for 7.12-1
7.12.7.2-3	The *FUNCTION-DESIGNATOR* **`fabsl`**	As for 7.12-1
7.12.7.3-1	The *FUNCTION-DESIGNATOR* **`hypot`**	As for 7.12-1
7.12.7.3-2	The *FUNCTION-DESIGNATOR* **`hypotf`**	As for 7.12-1
7.12.7.3-3	The *FUNCTION-DESIGNATOR* **`hypotl`**	As for 7.12-1
7.12.7.4-1	The *FUNCTION-DESIGNATOR* **`pow`**	As for 7.12-1
7.12.7.4-2	The *FUNCTION-DESIGNATOR* **`powf`**	As for 7.12-1
7.12.7.4-3	The *FUNCTION-DESIGNATOR* **`powl`**	As for 7.12-1
7.12.7.5-1	The *FUNCTION-DESIGNATOR* **`sqrt`**	As for 7.12-1
7.12.7.5-2	The *FUNCTION-DESIGNATOR* **`sqrtf`**	As for 7.12-1
7.12.7.5-3	The *FUNCTION-DESIGNATOR* **`sqrtl`**	As for 7.12-1

7.12.8 Error and gamma functions

Designated constructs:

DCRN	**Definition**	**Rationale**
7.12.8.1-1	The *FUNCTION-DESIGNATOR* **`erf`**	As for 7.12-1
7.12.8.1-2	The *FUNCTION-DESIGNATOR* **`erff`**	As for 7.12-1
7.12.8.1-3	The *FUNCTION-DESIGNATOR* **`erfl`**	As for 7.12-1
7.12.8.2-1	The *FUNCTION-DESIGNATOR* **`erfc`**	As for 7.12-1
7.12.8.2-2	The *FUNCTION-DESIGNATOR* **`erfcf`**	As for 7.12-1
7.12.8.2-3	The *FUNCTION-DESIGNATOR* **`erfcl`**	As for 7.12-1
7.12.8.3-1	The *FUNCTION-DESIGNATOR* **`lgamma`**	As for 7.12-1
7.12.8.3-2	The *FUNCTION-DESIGNATOR* **`lgammaf`**	As for 7.12-1
7.12.8.3-3	The *FUNCTION-DESIGNATOR* **`lgammal`**	As for 7.12-1
7.12.8.4-1	The *FUNCTION-DESIGNATOR* **`tgamma`**	As for 7.12-1
7.12.8.4-2	The *FUNCTION-DESIGNATOR* **`tgammaf`**	As for 7.12-1
7.12.8.4-3	The *FUNCTION-DESIGNATOR* **`tgammal`**	As for 7.12-1

7.12.9 Nearest integer functions

Designated constructs:

DCRN	Definition	Rationale
7.12.9.1-1	The *FUNCTION-DESIGNATOR* **`ceil`**	As for 7.12-1
7.12.9.1-2	The *FUNCTION-DESIGNATOR* **`ceilf`**	As for 7.12-1
7.12.9.1-3	The *FUNCTION-DESIGNATOR* **`ceill`**	As for 7.12-1
7.12.9.2-1	The *FUNCTION-DESIGNATOR* **`floor`**	As for 7.12-1
7.12.9.2-2	The *FUNCTION-DESIGNATOR* **`floorf`**	As for 7.12-1
7.12.9.2-3	The *FUNCTION-DESIGNATOR* **`floorl`**	As for 7.12-1
7.12.9.3-1	The *FUNCTION-DESIGNATOR* **`nearbyint`**	As for 7.12-1
7.12.9.3-2	The *FUNCTION-DESIGNATOR* **`nearbyintf`**	As for 7.12-1
7.12.9.3-3	The *FUNCTION-DESIGNATOR* **`nearbyintl`**	As for 7.12-1
7.12.9.4-1	The *FUNCTION-DESIGNATOR* **`rint`**	As for 7.12-1
7.12.9.4-2	The *FUNCTION-DESIGNATOR* **`rintf`**	As for 7.12-1
7.12.9.4-3	The *FUNCTION-DESIGNATOR* **`rintl`**	As for 7.12-1
7.12.9.5-1	The *FUNCTION-DESIGNATOR* **`lrint`**	As for 7.12-1
7.12.9.5-2	The *FUNCTION-DESIGNATOR* **`lrintf`**	As for 7.12-1
7.12.9.5-3	The *FUNCTION-DESIGNATOR* **`lrintl`**	As for 7.12-1
7.12.9.5-4	The *FUNCTION-DESIGNATOR* **`llrint`**	As for 7.12-1
7.12.9.5-5	The *FUNCTION-DESIGNATOR* **`llrintf`**	As for 7.12-1
7.12.9.5-6	The *FUNCTION-DESIGNATOR* **`llrintl`**	As for 7.12-1
7.12.9.6-1	The *FUNCTION-DESIGNATOR* **`round`**	As for 7.12-1
7.12.9.6-2	The *FUNCTION-DESIGNATOR* **`roundf`**	As for 7.12-1
7.12.9.6-3	The *FUNCTION-DESIGNATOR* **`roundl`**	As for 7.12-1
7.12.9.7 -1	The *FUNCTION-DESIGNATOR* **`lround`**	As for 7.12-1
7.12.9.7-2	The *FUNCTION-DESIGNATOR* **`lroundf`**	As for 7.12-1
7.12.9.7-3	The *FUNCTION-DESIGNATOR* **`lroundl`**	As for 7.12-1
7.12.9.7–4	The *FUNCTION-DESIGNATOR* **`llround`**	As for 7.12-1
7.12.9.7-5	The *FUNCTION-DESIGNATOR* **`llroundf`**	As for 7.12-1
7.12.9.7-6	The *FUNCTION-DESIGNATOR* **`llroundl`**	As for 7.12-1
7.12.9.8-1	The *FUNCTION-DESIGNATOR* **`trunc`**	As for 7.12-1
7.12.9.8-2	The *FUNCTION-DESIGNATOR* **`truncf`**	As for 7.12-1
7.12.9.8-3	The *FUNCTION-DESIGNATOR* **`truncl`**	As for 7.12-1

7.12.10 Remainder functions

Designated constructs:

DCRN	Definition	Rationale
7.12.10.1-1	The *FUNCTION-DESIGNATOR* **`fmod`**	As for 7.12-1
7.12.10.1-2	The *FUNCTION-DESIGNATOR* **`fmodf`**	As for 7.12-1
7.12.10.1-3	The *FUNCTION-DESIGNATOR* **`fmodl`**	As for 7.12-1
7.12.10.2-1	The *FUNCTION-DESIGNATOR* **`remainder`**	As for 7.12-1
7.12.10.2-2	The *FUNCTION-DESIGNATOR* **`remainderf`**	As for 7.12-1
7.12.10.2-3	The *FUNCTION-DESIGNATOR* **`remainderl`**	As for 7.12-1
7.12.10.3-1	The *FUNCTION-DESIGNATOR* **`remquo`**	As for 7.12-1
7.12.10.3-2	The *FUNCTION-DESIGNATOR* **`remquof`**	As for 7.12-1
7.12.10.3-3	The *FUNCTION-DESIGNATOR* **`remquol`**	As for 7.12-1

7.12.11 Manipulation functions

Designated constructs:

DCRN	Definition	Rationale
7.12.11.1-1	The *FUNCTION-DESIGNATOR* **`copysign`**	As for 7.12-1
7.12.11.1-2	The *FUNCTION-DESIGNATOR* **`copysignf`**	As for 7.12-1
7.12.11.1-3	The *FUNCTION-DESIGNATOR* **`copysignl`**	As for 7.12-1
7.12.11.2-1	The *FUNCTION-DESIGNATOR* **`nan`**	As for 7.12-1
7.12.11.2-2	The *FUNCTION-DESIGNATOR* **`nanf`**	As for 7.12-1
7.12.11.2-3	The *FUNCTION-DESIGNATOR* **`nanl`**	As for 7.12-1
7.12.11.3-1	The *FUNCTION-DESIGNATOR* **`nextafter`**	As for 7.12-1
7.12.11.3-2	The *FUNCTION-DESIGNATOR* **`nextafterf`**	As for 7.12-1
7.12.11.3-3	The *FUNCTION-DESIGNATOR* **`nextafterl`**	As for 7.12-1
7.12.11.4-1	The *FUNCTION-DESIGNATOR* **`nexttoward`**	As for 7.12-1
7.12.11.4-2	The *FUNCTION-DESIGNATOR* **`nexttowardf`**	As for 7.12-1
7.12.11.4-3	The *FUNCTION-DESIGNATOR* **`nexttowardl`**	As for 7.12-1

7.12.12 Maximum, minimum and positive difference functions

Designated constructs:

DCRN	Definition	Rationale
7.12.12.1-1	The *FUNCTION-DESIGNATOR* **`fdim`**	As for 7.12-1
7.12.12.1-2	The *FUNCTION-DESIGNATOR* **`fdimf`**	As for 7.12-1
7.12.12.1-3	The *FUNCTION-DESIGNATOR* **`fdiml`**	As for 7.12-1
7.12.12.2-1	The *FUNCTION-DESIGNATOR* **`fmax`**	As for 7.12-1
7.12.12.2-2	The *FUNCTION-DESIGNATOR* **`fmaxf`**	As for 7.12-1
7.12.12.2-3	The *FUNCTION-DESIGNATOR* **`fmaxl`**	As for 7.12-1
7.12.12.2-1	The *FUNCTION-DESIGNATOR* **`fmin`**	As for 7.12-1
7.12.12.2-2	The *FUNCTION-DESIGNATOR* **`fminf`**	As for 7.12-1
7.12.12.2-3	The *FUNCTION-DESIGNATOR* **`fminl`**	As for 7.12-1

7.12.13 Floating multiply-add

Designated constructs:

DCRN	Definition	Rationale
7.12.13.1-1	The *FUNCTION-DESIGNATOR* **`fma`**	As for 7.12-1
7.12.13.1-2	The *FUNCTION-DESIGNATOR* **`fmaf`**	As for 7.12-1
7.12.13.1-3	The *FUNCTION-DESIGNATOR* **`fmal`**	As for 7.12-1

7.12.14 Comparison macros

Designated constructs:

DCRN	Definition	Rationale
7.12.14.1-1	The *MACRO-NAME* **`isgreater`**	As for 7.12-1
7.12.14.2-1	The *MACRO-NAME* **`isgreaterequal`**	As for 7.12-1
7.12.14.3-1	The *MACRO-NAME* **`isless`**	As for 7.12-1
7.12.14.4-1	The *MACRO-NAME* **`islessequal`**	As for 7.12-1
7.12.14.5-1	The *MACRO-NAME* **`islessgreater`**	As for 7.12-1
7.12.14.6-1	The *MACRO-NAME* **`isunordered`**	As for 7.12-1

7.13 Nonlocal jumps <setjmp.h>

Designated constructs:

DCRN	Definition	Rationale
7.13-1	An *INCLUDE-DIRECTIVE* that causes inclusion of the **`<setjmp.h>`** header.	Many aspects of the facilities of **`<setjmp.h>`** are associated with **undefined** behaviour or can impairs the **ANALYSABILITY** of code.
7.13-2	The *typedef-name* **`jmpbuf`**.	As for 7.13-1.

7.13.1 Save calling environment

Designated constructs:

DCRN	Definition	Rationale
7.13.1-1	A *MACRO-INVOCATION* whose *MACRO-NAME* is **`setjmp`** but whose expansion does not occur as: • an *IF-EXPR* or a *WHILE-EXPR*, or • one operand of a *RELATIONAL-EXPR* or *EQUALITY-EXPR* that is an *IF-EXPR* or a *WHILE-EXPR* and where the other operand is an integer constant expression, or • the operand of a unary ! operator whose closest-containing *unary-expression* is an *IF-EXPR* or a *WHILE-EXPR*, • an *expression-statament*.	Behaviour is **undefined**.
7.13.1-2	The *MACRO-NAME* **`setjmp`**.	As for 7.13-1 (**ANALYSABILITY**)
7.13.1-3	A *FUNCTION-DESIGNATOR* that denotes **`setjmp`** implemented as a function..	As for 7.13-1 (**ANALYSABILITY**)

7.13.2 Restore calling environment

Designated constructs:

DCRN	Definition	Rationale
7.13.2-1	The *FUNCTION-DESIGNATOR* **`longjmp`**.	As for 7.13-1

7.14 Signal handling functions <signal.h>

Designated constructs:

DCRN	Definition	Rationale
7.14-1	An *INCLUDE-DIRECTIVE* that causes inclusion of the*e* `<signal.h>`. header.	Many aspects of signals are **implementation-dependent.**.
7.14-2	The *MACRO-NAME* `SIG_DFL`	As for 7.14-1.
7.14-3	The *MACRO-NAME* `SIG_ERR`	As for 7.14-1.
7.14-4	The *MACRO-NAME* `SIG_IGN`	As for 7.14-1.
7.14-5	The *MACRO-NAME* `SIGABRT`	As for 7.14-1.
7.14-6	The *MACRO-NAME* `SIGFPE`	As for 7.14-1.
7.14-7	The *MACRO-NAME* `SIGILL`	As for 7.14-1.
7.14-8	The *MACRO-NAME* `SIGINT`	As for 7.14-1.
7.14-9	The *MACRO-NAME* `SIGSEG`	As for 7.14-1.

7.14.1 Specify signal handling

Designated constructs:

DCRN	Definition	Rationale
7.14.1-1	The *FUNCTION-DESIGNATOR* `signal`.	As for 7.14-1.

7.14.2 Send signal

Designated constructs:

7.14.2-1	The *FUNCTION-DESIGNATOR* `raise`.	As for 7.14-1.

7.15 Variable arguments <stdarg.h>

Designated constructs:

DCRN	Definition	Rationale
7.15-1	A *INCLUDE-DIRECTIVE* that causes inclusion of the **`<stdarg.h>`**. header.	Many aspects of variable arguments are **implementation-dependent** and their use impairs the **ANALYSABILITY** of code.
7.15-4	The *typedef-name* **`va_list`**.	As for 7.15-1.

7.15.1 Variable argument list access macros

7.15.1.1 The `va_arg` macro

Designated constructs:

DCRN	Definition	Rationale
7.15.1.1-1	The *MACRO-NAME* **`va_arg`**	As for 7.15-1.
7.15.1.1-2	A construct that denotes **`va_arg`** implemented as an external object.	As for 7.15-1.

7.15.1.2 The `va_copy` macro

Designated constructs:

DCRN	Definition	Rationale
7.15.1.2-1	The *MACRO-NAME* **`va_copy`**	As for 7.15-1.
7.15.1.2-2	A construct that denotes **`va_copy`** implemented as an external object.	As for 7.15-1.

7.15.1.3 The `va_end` macro

Designated constructs:

DCRN	Definition	Rationale
7.15.1.3-1	The *MACRO-NAME* **`va_end`**.	As for 7.15-1.

7.15.1.4 The `va_start` macro

Designated constructs:

DCRN	Definition	Rationale
7.15.1.4-1	The *MACRO-NAME* **`va_start`**.	As for 7.15-1.

7.16 Boolean type and values <stdbool.h>

Designated constructs:

DCRN	Definition	Rationale
7.16-1	An *INCLUDE-DIRECTIVE* that causes inclusion of the **`<stdbool.h>`** header.	This header and its facilities may not be supported by implementations conforming to earlier version of the base language standard thereby impairing **PORTABILITY**.
7.16-2	The *MACRO-NAME* **`bool`**	As for 7.16-1
7.16-3	The *MACRO-NAME* **`true`**	As for 7.16-1
7.16-4	The *MACRO-NAME* **`false`**	As for 7.16-1
7.16-5	The *MACRO-NAME* **`__bool_true_false_are_defined`**	As for 7.16-1

7.17 Common definitions <stddef.h>

Designated constructs:

DCRN	Definition	Rationale
7.17-1	An *INCLUDE-DIRECTIVE* that causes inclusion of the **`<stddef.h>`** header.	See note below.
7.17-2	The *type-name* **`ptrdiff_t`**.	**ANALYSABILITY** (implied by rationale against use of pointer arithmetic).
7.17-3	The *type-name* **`size_t`**.	**Defensive programming** (implied by similar rationale for the **`sizeof`** operator).
7.17-4	The *type-name* **`wchar_t`**.	Implied by rationale for **implementation-dependent** aspects of wide characters.
7.17-5	The *MACRO-NAME* **`NULL`**.	See note below.
7.17-6	The *MACRO-NAME* **`offsetof`**.	**Defensive programming.**

Note: The **`<stddef.h>`** header provides very few facilities. Depending on the application there may be reason to control the use of all such facilities with the exception **`NULL`** macro. Accordingly some users may prefer to provide their own definition of **`NULL`** and ban inclusion of **`<stddef.h>.`**

7.18 Integer types <stdint.h>

Designated constructs:

DCRN	Definition	Rationale
7.18-1	An *INCLUDE-DIRECTIVE* that causes the inclusion of the **`<stdint.h>`**. header.	Many aspects of the the types provided by **`<stdint.h>`** are **implementation-dependent**.

7.18.1 Integer types

Designated constructs:

DCRN	Definition	Rationale
7.18.1.1-1	The *identifier* **`int`***N***`_t`**	As for 7.18-1
7.18.1.1-2	The *identifier* **`uint`***N***`_t`**	As for 7.18-1
7.18.1.2-1	The *identifier* **`int_least`***N***`_t`** (not otherwise specified)	As for 7.18-1
7.18.1.2-2	The *identifier* **`int_least8_t`**	As for 7.18-1
7.18.1.2-3	The *identifier* **`int_least16_t`**	As for 7.18-1
7.18.1.2-4	The *identifier* **`int_least32_t`**	As for 7.18-1
7.18.1.2-5	The *identifier* **`int_least64_t`**	As for 7.18-1
7.18.1.2-6	The *identifier* **`uint_least`***N***`_t`** (not otherwise specified)	As for 7.18-1
7.18.1.2-7	The *identifier* **`uint_least8_t`**	As for 7.18-1
7.18.1.2-8	The *identifier* **`uint_least16_t`**	As for 7.18-1
7.18.1.2-9	The *identifier* **`uint_least32_t`**	As for 7.18-1
7.18.1.2-10	The *identifier* **`uint_least64_t`**	As for 7.18-1
7.18.1.3-1	The *identifier* **`int_fast`***N***`_t`** (not otherwise specified)	As for 7.18-1
7.18.1.3-2	The *identifier* **`int_fast8_t`**	As for 7.18-1
7.18.1.3-3	The *identifier* **`int_fast16_t`**	As for 7.18-1
7.18.1.3-4	The *identifier* **`int_fast32_t`**	As for 7.18-1
7.18.1.3-5	The *identifier* **`int_fast64_t`**	As for 7.18-1
7.18.1.3-6	The *identifier* **`uint_fast`***N***`_t`** (not otherwise specified)	As for 7.18-1
7.18.1.3-7	The *identifier* **`uint_fast8_t`**	As for 7.18-1
7.18.1.3-8	The *identifier* **`uint_fast16_t`**	As for 7.18-1
7.18.1.3-9	The *identifier* **`uint_fast32_t`**	As for 7.18-1

7.18.1.3-10	The *identifier* **uint_fast64_t**	As for 7.18-1
7.18.1.4-1	The *identifier* **intptr_t**	As for 7.18-1
7.18.1.4-2	The *identifier* **uintptr_t**	As for 7.18-1
7.18.1.5-1	The *identifier* **intmax_t**	As for 7.18-1
7.18.1.5-2	The *identifier* **uintmax_t**	As for 7.18-1

7.18.2 Limits of specified-width integer types

Designated constructs:

DCRN	Definition	Rationale
7.18.2.1-1	The *MACRO-NAME* **INT*N*_MIN**	As for 7.18-1
7.18.2.1-2	The *MACRO-NAME* **INT*N*_MAX**	As for 7.18-1
7.18.2.1-3	The *MACRO-NAME* **UINT*N*_MAX**	As for 7.18-1
7.18.2.2-1	The *MACRO-NAME* **INT_LEAST*N*_MIN**	As for 7.18-1
7.18.2.2-2	The *MACRO-NAME* **INT_LEAST*N*_MAX**	As for 7.18-1
7.18.2.2-3	The *MACRO-NAME* **UINT_LEAST*N*_MAX**	As for 7.18-1
7.18.2.3-1	The *MACRO-NAME* **INT_FAST*N*_MIN**	As for 7.18-1
7.18.2.3-2	The *MACRO-NAME* **INT_FAST*N*_MAX**	As for 7.18-1
7.18.2.3-3	The *MACRO-NAME* **UINT_FAST*N*_MAX**	As for 7.18-1
7.18.2.4-1	The *MACRO-NAME* **INTPTR_MIN**	As for 7.18-1
7.18.2.4-2	The *MACRO-NAME* **INTPTR_MAX**	As for 7.18-1
7.18.2.4-3	The *MACRO-NAME* **UINTPTR_MAX**	As for 7.18-1
7.18.2.5-1	The *MACRO-NAME* **INTMAX_MIN**	As for 7.18-1
7.18.2.5-2	The *MACRO-NAME* **INTMAX_MAX**	As for 7.18-1
7.18.2.5-3	The *MACRO-NAME* **UINTMAX_MAX**	As for 7.18-1

7.18.3 Limits of other integer types

Designated constructs:

DCRN	Definition	Rationale
7.18.3-1	The *MACRO-NAME* **`PTRDIFF_MIN`**	As for 7.18-1
7.18.3-2	The *MACRO-NAME* **`PTRDIFF_MAX`**	As for 7.18-1
7.18.3-3	The *MACRO-NAME* **`SIG_ATOMIC_MIN`**	As for 7.18-1
7.18.3-4	The *MACRO-NAME* **`SIG_ATOMIC_MAX`**	As for 7.18-1
7.18.3-5	The *MACRO-NAME* **`SIZE_MAX`**	As for 7.18-1
7.18.3-6	The *MACRO-NAME* **`WCHAR_MIN`**	As for 7.18-1
7.18.3-6	The *MACRO-NAME* **`WCHAR_MAX`**	As for 7.18-1
7.18.3-6	The *MACRO-NAME* **`WINT_MIN`**	As for 7.18-1
7.18.3-6	The *MACRO-NAME* **`WINT_MAX`**	As for 7.18-1

7.18.4 Macros for integer constants

Designated constructs:

DCRN	Definition	Rationale
7.18.4.1-1	The *MACRO-NAME* **`INT`***N***`_C`**	As for 7.18-1
7.18.4.1-2	The *MACRO-NAME* **`UINT`***N***`_C`**	As for 7.18-1
7.18.4.2-1	The *MACRO-NAME* **`INTMAX_C`**	As for 7.18-1
7.18.4.2-2	The *MACRO-NAME* **`UINTMAX_C`**	As for 7.18-1

7.19 Input/output <stdio.h>

Designated constructs:

DCRN	Definition	Rationale
7.19-1	An include-directive that causes inclusion of the `<stdio.h>` header.	Many aspects of input and output are **implementation-dependent**.

7.19.1 Introduction

Designated constructs:

DCRN	Definition	Rationale
7.19.1-1	The *typedef-name* `FILE`.	As for 7.19-1
7.19.1-2	The *typedef-name* `fpos_t`.	As for 7.19-1
7.19.1-3	The *MACRO-NAME* `_IOFBF`	As for 7.19-1
7.19.1-4	The *MACRO-NAME* `_IOLBF`	As for 7.19-1
7.19.1-5	The *MACRO-NAME* `_IONBF`	As for 7.19-1
7.19.1-6	The *MACRO-NAME* `BUFSIZ`	As for 7.19-1
7.19.1-7	The *MACRO-NAME* `EOF`	As for 7.19-1
7.19.1-8	The *MACRO-NAME* `FOPEN_MAX`	As for 7.19-1
7.19.1-9	The *MACRO-NAME* `FILENAME_MAX`	As for 7.19-1
7.19.1-10	The *MACRO-NAME* `L_tmpnam`	As for 7.19-1
7.19.1-11	The *MACRO-NAME* `SEEK_CUR`	As for 7.19-1
7.19.1-12	The *MACRO-NAME* `SEEK_END`	As for 7.19-1
7.19.1-13	The *MACRO-NAME* `SEEK_SET`	As for 7.19-1
7.19.1-14	The *MACRO-NAME* `TMP_MAX`	As for 7.19-1
7.19.1-15	The *MACRO-NAME* `stderr`	As for 7.19-1
7.19.1-16	The *MACRO-NAME* `stdin`	As for 7.19-1
7.19.1-17	The *MACRO-NAME* `stdout`	As for 7.19-1
7.19.1-18	A construct whose E-behaviour contains an access to part of an object of type `FILE`.	Effects are **implementation-dependent** and can be unpredictable.
7.19.1-19	A construct that attempts to copy an object of type `FILE`.	Effects are **implementation-dependent** and can be unpredictable.
7.19.1-20	A *FUNCTION-CALL-EXPRESSION* for which the evaluation of an argument that denotes a file contains a side effect.	Effects are **implementation-dependent**.

7.19.2 Streams (NR)

7.19.3 Files (NR)

7.19.4 Operations on files

Designated constructs:

DCRN	Definition	Rationale
7.19.4.1-1	The *FUNCTION-DESIGNATOR* **`remove`**	As for 7.19-1
7.19.4.1-2	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* is **`remove`** and that attempts to remove a file that is open.	Behaviour is **implementation-defined**.
7.19.4.2-1	The *FUNCTION-DESIGNATOR* **`rename`**	As for 7.19-1
7.19.4.2-2	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* is **`remame`** and that attempts to rename a file to that of a file that already exists.	Behaviour is **implementation-defined**.
7.19.4.3-1	The *FUNCTION-DESIGNATOR* **`tmpfile`**	As for 7.19-1
7.19.4.4-1	The *FUNCTION-DESIGNATOR* **`tmpnam`**	As for 7.19-1

7.19.5 File access functions

Designated constructs:

DCRN	Definition	Rationale
7.19.5-1	The *FUNCTION-DESIGNATOR* **`fclose`**	As for 7.19-1
7.19.5-2	The *FUNCTION-DESIGNATOR* **`fflush`**	As for 7.19-1
7.19.5-3	The *FUNCTION-DESIGNATOR* **`fopen`**	As for 7.19-1
7.19.5-4	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* is **`fopen`** and that attempts to open a file when eight files are already open.	Behaviour is **implementation-defined**.
7.19.5-5	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* is **`fopen`** and that attempts to open a file in append mode.	Aspects of writing in append mode are **implementation-dependent.**
7.19.5-6	A non-standard mode string.	Behaviour is **undefined**.
7.19.5-7	The *FUNCTION-DESIGNATOR* **`freopen`**	As for 7.19-1
7.19.5-8	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* is **`freopen`** and that attempts to reopen a file in mode other than that in	The effects of re-opening with a different mode are **implementation-defined.**

	which it was previously opened.	
7.19.5-9	The *FUNCTION-DESIGNATOR* **`setbuf`**	As for 7.19-1
7.19.5-10	The *FUNCTION-DESIGNATOR* **`setvbuf`**	As for 7.19-1
7.19.5-11	A *FUNCTION-CALL-EXPRESSION* that is applied to a wide-oriented stream but whose *FUNCTION-DESIGNATOR* denotes a byte-oriented function.	Behaviour is **undefined**.
7.19.5-12	A *FUNCTION-CALL-EXPRESSION* that is applied to a byte-oriented stream but whose *FUNCTION-DESIGNATOR* denotes a wide-oriented function.	Behaviour is **undefined**.

7.19.6 Formatted input/output functions

Designated constructs:

DCRN	Definition	Rationale
7.19.6-	A format non-standard conversion specifier.	Behaviour is **undefined**.
7.19.6-	A format string containing a non-standard combination of conversion specifiers and flags.	Behaviour is **undefined**.
7.19.6-	A multibyte format string that does not both start and end in the initial shift state.	Such a construct violates a **constraint**.
7.19.6-	An occurrence of the backspace character within a format string.	Behaviour on a display device may be **unspecified**.
7.19.6-	An occurrence of: the horizontal tab character within a format string.	Behaviour on a display device may be **unspecified**.
7.19.6-	A construct whose execution causes a printable character to be written when the active position is at the final position of a line.	Behaviour on a display device may be **unspecified**.
7.19.6-	An occurrence of: the vertical tab character within a format string.	Behaviour on a display device may be **unspecified**.
7.19.6-	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* denotes a formatted I/O function and that has no *argument-expression-list*.	As for 7.19-1
7.19.6-	A format string that denotes a null string.	**Defensive programming.**
7.19.6-	A format string in which white space characters immediately precede a new-line character.	Effects on writing are **unspecified**.
7.19.6-	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* denotes a formatted I/O function for which the conversion specifiers in the format string and the numbers and types of arguments do not correspond.	Behaviour is **undefined**.

7.19.6-	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* denotes a formatted I/O function and that attempts to write a text line whose length exceeds 254 characters.	Behaviour is **implementation-defined**.
7.19.6-	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* denotes a formatted read function that attempts to assign values to overlapping objects.	Behaviour is **undefined**.
7.19.6-	A scanset specifier in which the same character occurs more than once.	The repeated character is redundant.
7.19.6-	A scanset specifier containing the – character in which the value of the character preceding - exceeds that of the character that follows.	Behaviour is **undefined**.
7.19.6-	The *FUNCTION-DESIGNATOR* **`fprintf`**	As for 7.19-1
7.19.6-	The *FUNCTION-DESIGNATOR* **`fscanf`**	As for 7.19-1
7.19.6-	The *FUNCTION-DESIGNATOR* **`printf`**	As for 7.19-1
7.19.6-	The *FUNCTION-DESIGNATOR* **`scanf`**	As for 7.19-1
7.19.6-	The *FUNCTION-DESIGNATOR* **`snprintf`**	As for 7.19-1
7.19.6-	The *FUNCTION-DESIGNATOR* **`sprintf`**	As for 7.19-1
7.19.6-	The *FUNCTION-DESIGNATOR* **`sprintf`**	As for 7.19-1
7.19.6-	The *FUNCTION-DESIGNATOR* **`vfprintf`**	As for 7.19-1
7.19.6-	The *FUNCTION-DESIGNATOR* **`vfscanf`**	As for 7.19-1
7.19.6-	The *FUNCTION-DESIGNATOR* **`vprintf`**	As for 7.19-1
7.19.6-	The *FUNCTION-DESIGNATOR* **`vscanf`**	As for 7.19-1
7.19.6-	The *FUNCTION-DESIGNATOR* **`vsnprintf`**	As for 7.19-1
7.19.6-	The *FUNCTION-DESIGNATOR* **`vsprintf`**	As for 7.19-1
7.19.6-	The *FUNCTION-DESIGNATOR* **`vsscanf`**	As for 7.19-1

7.19.7 Character input/output functions

Designated constructs:

DCRN	Definition	Rationale
7.19.7.1-1	The *FUNCTION-DESIGNATOR* **`fgetc`**	As for 7.19-1
7.19.7.2-1	The *FUNCTION-DESIGNATOR* **`fgets`**	As for 7.19-1
7.19.7.3-1	The *FUNCTION-DESIGNATOR* **`fputc`**	As for 7.19-1

7.19.7.4-1	The *FUNCTION-DESIGNATOR* `fputs`	As for 7.19-1
7.19.7.5-1	The *FUNCTION-DESIGNATOR* `getc`	As for 7.19-1
7.19.7.6-1	The *FUNCTION-DESIGNATOR* `getchar`	As for 7.19-1
7.19.7.7-1	The *FUNCTION-DESIGNATOR* `gets`	As for 7.19-1
7.19.7.8-1	The *FUNCTION-DESIGNATOR* `putc`	As for 7.19-1
7.19.7.9-1	The *FUNCTION-DESIGNATOR* `putchar`	As for 7.19-1
7.19.7.10-1	The *FUNCTION-DESIGNATOR* `puts`	As for 7.19-1
7.19.7.11-1	The *FUNCTION-DESIGNATOR* `ungetc`	As for 7.19-1

7.19.8 Direct input/output functions

Designated constructs:

DCRN	**Definition**	**Rationale**
7.19.8.1-1	The *FUNCTION-DESIGNATOR* `fread`	As for 7.19-1
7.19.8.2-1	The *FUNCTION-DESIGNATOR* `fwrite`	As for 7.19-1

7.19.9 File positioning functions

Designated constructs:

DCRN	**Definition**	**Rationale**
7.19.9.1-1	The *FUNCTION-DESIGNATOR* `fgetpos`	As for 7.19-1
7.19.9.2-1	The *FUNCTION-DESIGNATOR* `fseek`	As for 7.19-1
7.19.9.2-2	A *FUNCTION-CALL-EXPRESSION* whose *FUNCTION-DESIGNATOR* denotes the fseek function and that attempts to position to `SEEK_END`.	Effects are **undefined.**
7.19.9.3-1	The *FUNCTION-DESIGNATOR* `fsetpos`	As for 7.19-1
7.19.9.4-1	The *FUNCTION-DESIGNATOR* `ftell`	As for 7.19-1
7.19.9.5-1	The *FUNCTION-DESIGNATOR* `rewind`	As for 7.19-1

7.19.10 Error-handling functions

Designated constructs:

DCRN	**Definition**	**Rationale**
7.19.10.1-1	The *FUNCTION-DESIGNATOR* `clearer`	As for 7.19-1
7.19.10.2-1	The *FUNCTION-DESIGNATOR* `feof`	As for 7.19-1

7.19.10.3-1	The *FUNCTION-DESIGNATOR* **`ferror`**	As for 7.19-1
7.19.10.4-1	The *FUNCTION-DESIGNATOR* **`perror`**	As for 7.19-1

7.20 General utilities <stdlib.h>

Designated constructs:

DCRN	Definition	Rationale
7.20-1	An *INCLUDE-DIRECTIVE* that cuases inclusion of the **`<stdlib.h>`** header.	Most features provided by this header have characteristics that impair one or more non-functional attributes.
7.20-4	The *typedef-name* **`div_t`**.	By implication from 7.20.6.2-1, 7.20.6.2-2
7.20-5	The *typedef-name* **`ldiv_t`**.	By implication from 7.20.6.2-1, 7.20.6.2-2
7.20-6	The *typedef-name* **`lldiv_t`**.	By implication from 7.20.6.2-3
7.20-7	The *MACRO-NAME* **`EXIT_FAILURE`**	By implication from 7.20.4-3 and 7.20.4-4
7.20-8	The *MACRO-NAME* **`EXIT_SUCCESS`**	By implication from 7.20.4-3 and 7.20.4-4
7.20-9	The *MACRO-NAME* **`RAND_MAX`**	By implication from 7.20.2-1.
7.20-10	The *MACRO-NAME* **`MB_CUR_MAX`**	Support for multibyte characters is **implementation-dependent**.

7.20.1 Numeric conversion functions

Designated constructs:

DCRN	Definition	Rationale
7.20.1.1-1	The *FUNCTION-DESIGNATOR* **`atof`**	Since none of these functions is bounded they all carry the risk of buffer overrun and thereby potentially impair **SECURITY**.
7.20.1.2-1	The *FUNCTION-DESIGNATOR* **`atoi`**	
7.20.1.2-2	The *FUNCTION-DESIGNATOR* **`atoll`**	
7.20.1.2-3	The *FUNCTION-DESIGNATOR* **`atoll`**	
7.20.1.3-1	The *FUNCTION-DESIGNATOR* **`strtod`**	
7.20.1.3-2	The *FUNCTION-DESIGNATOR* **`strtof`**	
7.20.1.3-3	The *FUNCTION-DESIGNATOR* **`strtold`**	
7.20.1.4-1	The *FUNCTION-DESIGNATOR* **`strtol`**	
7.20.1.4-2	The *FUNCTION-DESIGNATOR* **`strtoll`**	
7.20.1.4-3	The *FUNCTION-DESIGNATOR* **`strtoul`**	
7.20.1.4-4	The *FUNCTION-DESIGNATOR* **`strtoull`**	

7.20.2 Pseudo-random sequence generation functions

Designated constructs:

DCRN	Definition	Rationale
7.20.2-1	The *FUNCTION-DESIGNATOR* `rand`	The **FUNCTIONALITY** of `rand` may not be fit for purpose in critical applications.
7.20.2-2	The *FUNCTION-DESIGNATOR* `srand`	As for 7.20.2-1 by implication.

7.20.3 Memory management functions

Designated constructs:

DCRN	Definition	Rationale
7.20.3-1	The *FUNCTION-DESIGNATOR* `calloc`	Use of dynamically allocated memory can impair the **ANALYSABILITY** of code.
7.20.3-2	The *FUNCTION-DESIGNATOR* `free`	As for 7.20.3-1
7.20.3-3	The *FUNCTION-DESIGNATOR* `malloc`	As for 7.20.3-1
7.20.3-4	The *FUNCTION-DESIGNATOR* `realloc`	As for 7.20.3-1

7.20.4 Communication with the environment

Designated constructs:

DCRN	Definition	Rationale
7.20.4-1	The *FUNCTION-DESIGNATOR* `abort`	Communication with the environment is **implementation-dependent**.
7.20.4-2	The *FUNCTION-DESIGNATOR* `atexit`	As for 7.20.4-1
7.20.4.3	The *FUNCTION-DESIGNATOR* `exit`	As for 7.20.4-1
7.20.4-4	The *FUNCTION-DESIGNATOR* `_Exit`	As for 7.20.4-1
7.20.4-5	The *FUNCTION-DESIGNATOR* `getenv`	As for 7.20.4-1
7.20.4-6	The *FUNCTION-DESIGNATOR* `system`	As for 7.20.4-1

7.20.5 Searching and sorting utilities

Designated constructs:

DCRN	Definition	Rationale
7.20.5-1	The *FUNCTION-DESIGNATOR* `bsearch`	If two elements of the searched array compare as equal, wich element is matched is **unspecified**.
7.20.5-2	The *FUNCTION-DESIGNATOR* `qsort`	If two elements compare as equal, their order in the resulting sorted array is **unspecified**.

7.20.6 Integer arithmetic functions

7.20.6.1 The `abs`, `labs` and `llabs` functions.

Designated constructs:

DCRN	Definition	Rationale
7.20.6.1-1	The *FUNCTION-DESIGNATOR* **`abs`**	**TIME BEHAVIOUR:** Absolute value functions are used very extensively in numerical software where efficiency is at a premium. The implementation of such functions as provided by a conforming implementation may not fast enough for all requirements and users may wish to control their use accordingly.
7.20.6.1-2	The *FUNCTION-DESIGNATOR* **`labs`**	
7.20.6.1-2	The *FUNCTION-DESIGNATOR* **`llabs`**	

7.20.6.2 The `div`, `ldiv` and `lldiv` functions.

Designated constructs:

DCRN	Definition	Rationale
7.20.6.2-1	The *FUNCTION-DESIGNATOR* **`div`**	Aspects of **`div`** and **`ldiv`** are **implementation-defined** for implementations conforming to earlier version of the base language standard, this impairing **PORTABILITY**.
7.20.6.2-2	The *FUNCTION-DESIGNATOR* **`ldiv`**	
7.20.6.2-3	The *FUNCTION-DESIGNATOR* **`lldiv`**	The **`lldiv`** function may not be supported by implementations conforming to earlier version of the base language standard, this impairing **PORTABILITY**.

7.20.7 Multibyte/wide character conversion functions

Designated constructs:

DCRN	Definition	Rationale
7.20.7-1	The *FUNCTION-DESIGNATOR* **`mblen`**	Support for wide and multibyte characters is **implementation-dependent**.
7.20.7-2	The *FUNCTION-DESIGNATOR* **`mbtowc`**	As for 7.20.7-1
7.20.7-3	The *FUNCTION-DESIGNATOR* **`wctomb`**	As for 7.20.7-1

7.20.8 Multibyte/wide string conversion functions

Designated constructs:

DCRN	Definition	Rationale
7.20.8-1	The *FUNCTION-DESIGNATOR* **`mbstowcs`**	Support for wide and multibyte characters is **implementation-dependent**.
7.20.8-2	The *FUNCTION-DESIGNATOR* **`wcstombs`**	As for 7.20.8-1

7.21 String handling <string.h>

Designated constructs:

DCRN	Definition	Rationale
7.21-1	An *INCLUDE-DIRECTIVE* that cuases inclusion of the **`<string.h>`**. header.	Many aspects of string handling are **implementation-dependent** or may impair **SECURITY**.

7.21.1 String function conventions (NR)

7.21.2 Copying functions

Designated constructs:

DCRN	Definition	Rationale
7.21.2.1-1	The *FUNCTION-DESIGNATOR* **`memcpy`**	Behaviour is **implementation-dependent** and is not bounded thus impairing **SECURITY**.
7.21.2.2-1	The *FUNCTION-DESIGNATOR* **`memmove`**	Behaviour is bounded but may rely on memory management functions thus potentially impairing **SECURITY**.
7.21.2.3-1	The *FUNCTION-DESIGNATOR* **`strcpy`**	Behaviour is **implementation-dependent** and is not bounded thus impairing **SECURITY**.
7.21.2.4-1	The *FUNCTION-DESIGNATOR* **`strncpy`**	Behaviour is **implementation dependent**.

Note: Implementations of string copying functions may rely on memory management functions. See 7.10.3.

7.21.3 Concatenation functions

Designated constructs:

DCRN	Definition	Rationale
7.21.3.1-1	The *FUNCTION-DESIGNATOR* **`strcat`**	Behaviour is **implementation-dependent** and is not bounded thus impairing **SECURITY**.
7.21.3.2-1	The *FUNCTION-DESIGNATOR* **`strncat`**	Behaviour is bounded but may rely on memory management functions thus potentially impairing **SECURITY**.

Note: Implementations of string concatenation functions may rely on memory management functions. See also 7.10.3.

7.21.4 Comparison functions

Designated constructs:

DCRN	Definition	Rationale
7.21.4.1-1	The *FUNCTION-DESIGNATOR* **`memcmp`**	Behaviour is not bounded thereby impairing **SECURITY**.
7.21.4.2-1	The *FUNCTION-DESIGNATOR* **`strcmp`**	Behaviour is not bounded thereby impairing **SECURITY**.
7.21.4.3-1	The *FUNCTION-DESIGNATOR* **`strcoll`**	The **`strcoll`** function is **locale-dependent**.
7.21.4.4-1	The *FUNCTION-DESIGNATOR* **`strncmp`**	Other things being equal the **`strncmp`** function should be preferred to the **`memcmp`** function because of stronger type checking.
7.21.4.5-1	The *FUNCTION-DESIGNATOR* **`strxfrm`**	The **`strxfrm`** function is **locale-dependent**.

7.21.5 Search functions

Designated constructs:

DCRN	Definition	Rationale
7.21.5.1-1	The *FUNCTION-DESIGNATOR* **`memchr`**	The use of void parameters means that the **`memchr`** function is not type-safe and its use impairs **ANALYZABILITY**.
7.21.5.2-1	The *FUNCTION-DESIGNATOR* **`strchr`**	Behaviour is not bounded thereby potentially impairing **SECURITY**.
7.21.5.3-1	The *FUNCTION-DESIGNATOR* **`strcspn`**	Behaviour is not bounded thereby potentially impairing **SECURITY**.
7.21.5.4-1	The *FUNCTION-DESIGNATOR* **`strpbrk`**	Behaviour is not bounded thereby potentially impairing **SECURITY**.
7.21.5.5-1	The *FUNCTION-DESIGNATOR* **`strrchr`**	Behaviour is not bounded thereby potentially impairing **SECURITY**.
7.21.5.6-1	The *FUNCTION-DESIGNATOR* **`strspn`**	Behaviour is not bounded thereby potentially impairing **SECURITY**.
7.21.5.7-1	The *FUNCTION-DESIGNATOR* **`strstr`**	Behaviour is not bounded thereby potentially impairing **SECURITY**.
7.21.5.8-1	The *FUNCTION-DESIGNATOR* **`strtok`**	Behaviour is not bounded thereby potentially impairing **SECURITY**.

7.21.6 Miscellaneous functions

Designated constructs:

DCRN	Definition	Rationale
7.21.6.1-1	The *FUNCTION-DESIGNATOR* **`memset`**	The use of void parameters means that the **`memchr`** function is not type-safe and its use impairs **ANALYZABILITY**.
7.21.6.2-1	The *FUNCTION-DESIGNATOR* **`strerror`**	The **`strerror`** function is **implementation-dependent**.
7.21.6.3-1	The *FUNCTION-DESIGNATOR* **`strlen`**	Behaviour is not bounded thereby potentially impairing **SECURITY**.

7.22 Type-generic math <tgmath.h>

Designated constructs:

DCRN	Definition	Rationale
7.22-1	An *INCLUDE-DIRECTIVE* that causes inclusion of the **`<tgmath.h>`** header.	Several aspects of mathematical functions are **implementation-defined** and mathematical functions may not exhibit sufficient **ACCURACY** for critical numerical applications.
7.22-2	The *MACRO-NAME* **`acos`**	As for 7.22-1
7.22-3	The *MACRO-NAME* **`asin`**	As for 7.22-1
7.22-4	The *MACRO-NAME* **`atan`**	As for 7.22-1
7.22-5	The *MACRO-NAME* **`acosh`**	As for 7.22-1
7.22-6	The *MACRO-NAME* **`asinh`**	As for 7.22-1
7.22-7	The *MACRO-NAME* **`atanh`**	As for 7.22-1
7.22-8	The *MACRO-NAME* **`cos`**	As for 7.22-1
7.22-9	The *MACRO-NAME* **`sin`**	As for 7.22-1
7.22-10	The *MACRO-NAME* **`tan`**	As for 7.22-1
7.22-11	The *MACRO-NAME* **`cosh`**	As for 7.22-1
7.22-12	The *MACRO-NAME* **`sinh`**	As for 7.22-1
7.22-13	The *MACRO-NAME* **`tanh`**	As for 7.22-1
7.22-14	The *MACRO-NAME* **`exp`**	As for 7.22-1
7.22-15	The *MACRO-NAME* **`log`**	As for 7.22-1
7.22-16	The *MACRO-NAME* **`pow`**	As for 7.22-1
7.22-17	The *MACRO-NAME* **`sqrt`**	As for 7.22-1
7.22-18	The *MACRO-NAME* **`fabs.`**	As for 7.22-1
7.22-19	The *MACRO-NAME* **`atan2`**	As for 7.22-1
7.22-20	The *MACRO-NAME* **`cbrt`**	As for 7.22-1
7.22-21	The *MACRO-NAME* **`ceil`**	As for 7.22-1
7.22-22	The *MACRO-NAME* **`copysign`**	As for 7.22-1
7.22-23	The *MACRO-NAME* **`erf`**	As for 7.22-1
7.22-24	The *MACRO-NAME* **`exp2`**	As for 7.22-1

7.22-25	The *MACRO-NAME* **`expml`**	As for 7.22-1
7.22-26	The *MACRO-NAME* **`fdim`**	As for 7.22-1
7.22-27	The *MACRO-NAME* **`floor`**	As for 7.22-1
7.22-28	The *MACRO-NAME* **`fma`**	As for 7.22-1
7.22-29	The *MACRO-NAME* **`fmax`**	As for 7.22-1
7.22-30	The *MACRO-NAME* **`fmin`**	As for 7.22-1
7.22-31	The *MACRO-NAME* **`fmod`**	As for 7.22-1
7.22-32	The *MACRO-NAME* **`frexp`**	As for 7.22-1
7.22-33	The *MACRO-NAME* **`hypot`**	As for 7.22-1
7.22-34	The *MACRO-NAME* **`ilogb`**	As for 7.22-1
7.22-35	The *MACRO-NAME* **`ldexp`**	As for 7.22-1
7.22-36	The *MACRO-NAME* **`lgamma`**	As for 7.22-1
7.22-37	The *MACRO-NAME* **`llrint`**	As for 7.22-1
7.22-38	The *MACRO-NAME* **`llround`**	As for 7.22-1
7.22-39	The *MACRO-NAME* **`log10`**	As for 7.22-1
7.22-40	The *MACRO-NAME* **`log1p`**	As for 7.22-1
7.22-41	The *MACRO-NAME* **`log2`**	As for 7.22-1
7.22-42	The *MACRO-NAME* **`logb`**	As for 7.22-1
7.22-43	The *MACRO-NAME* **`lrint`**	As for 7.22-1
7.22-44	The *MACRO-NAME* **`lround`**	As for 7.22-1
7.22-45	The *MACRO-NAME* **`nearbyint`**	As for 7.22-1
7.22-46	The *MACRO-NAME* **`nextafter`**	As for 7.22-1
7.22-47	The *MACRO-NAME* **`nexttoward`**	As for 7.22-1
7.22-48	The *MACRO-NAME* **`remainder`**	As for 7.22-1
7.22-49	The *MACRO-NAME* **`remquo`**	As for 7.22-1
7.22-50	The *MACRO-NAME* **`rint`**	As for 7.22-1
7.22-51	The *MACRO-NAME* **`round`**	As for 7.22-1
7.22-52	The *MACRO-NAME* **`scalbn`**	As for 7.22-1
7.22-53	The *MACRO-NAME* **`scalbln`**	As for 7.22-1

7.22-54	The *MACRO-NAME* **`tgamma`**	As for 7.22-1
7.22-55	The *MACRO-NAME* **`trunc`**	As for 7.22-1

7.23 Date and time <time.h>

Designated constructs:

DCRN	Definition	Rationale
7.23-1	An *INCLUDE-DIRECTIVE* that causes inclusion of the **`<time.h>`** header.	Time measurement is **implementation-dependent.**

7.23.1 Components of time

Designated constructs:

DCRN	Definition	Rationale
7.23-1	The *MACRO-NAME* **`CLOCKS_PER_SEC`**	As for 7.23-1
7.23-2	The *typedef-name* **`clock_t`**	As for 7.23-1
7.23-3	The *typedef-name* **`time_t`**	As for 7.23-1
7.23-4	The *struct-or-union-specifier* **`struct tm`**	As for 7.23-1

7.23.2 Time manipulation functions

Designated constructs:

DCRN	Definition	Rationale
7.23.2.1-1	The *FUNCTION-DESIGNATOR* **`clock`**	As for 7.23-1
7.23.2.2-1	The *FUNCTION-DESIGNATOR* **`difftime`**	As for 7.23-1
7.23.2.3-1	The *FUNCTION-DESIGNATOR* **`mktime`**	As for 7.23-1
7.23.2.4-1	The *FUNCTION-DESIGNATOR* **`time`**	As for 7.23-1

7.23.3 Time conversion functions

Designated constructs:

DCRN	Definition	Rationale
7.23.3.1-1	The *FUNCTION-DESIGNATOR* **`asctime`**	As for 7.23-1
7.23.3.2-1	The *FUNCTION-DESIGNATOR* **`ctime`**	As for 7.23-1
7.23.3.3-1	The *FUNCTION-DESIGNATOR* **`gmtime`**	As for 7.23-1
7.23.3.4-1	The *FUNCTION-DESIGNATOR* **`localtime`**	As for 7.23-1
7.23.3.5-1	The *FUNCTION-DESIGNATOR* **`strftime`**	

7.24 Extended multibyte and wide character utilities `<wchar.h>`

Designated constructs:

DCRN	Definition	Rationale
7.24-1	An *INCLUDE-DIRECTIVE* that cuases inclusion of the **`<wchar.h>`**. header.	Wide character support is **implementation-dependent**.

7.24.1 Introduction

Designated constructs:

DCRN	Definition	Rationale
7.24.1-1	The *typedef-name* **`mbstate_t`**	As for 7.24-1
7.24.1-2	The *typedef-name* **`wint_t`**	As for 7.24-1
7.24.1-3	The *MACRO-NAME* **`weof`**	As for 7.24-1

7.24.2 Formatted wide character input/output functions

Designated constructs:

DCRN	Definition	Rationale
7.24.2-1	The *FUNCTION-DESIGNATOR* **`fwprintf`**	As for 7.24-1
7.24.2-2	The *FUNCTION-DESIGNATOR* **`fwscanf`**	As for 7.24-1
7.24.2-3	The *FUNCTION-DESIGNATOR* **`swprintf`**	As for 7.24-1
7.24.2-4	The *FUNCTION-DESIGNATOR* **`swscanf`**	As for 7.24-1
7.24.2-5	The *FUNCTION-DESIGNATOR* **`vfwprintf`**	As for 7.24-1
7.24.2-6	The *FUNCTION-DESIGNATOR* **`vfwscanf`**	As for 7.24-1
7.24.2-7	The *FUNCTION-DESIGNATOR* **`vswprintf`**	As for 7.24-1
7.24.2-8	The *FUNCTION-DESIGNATOR* **`vswscanf`**	As for 7.24-1
7.24.2-9	The *FUNCTION-DESIGNATOR* **`vwprintf`**	As for 7.24-1
7.24.2-10	The *FUNCTION-DESIGNATOR* **`vwscanf`**	As for 7.24-1
7.24.2-11	The *FUNCTION-DESIGNATOR* **`wprintf`**	As for 7.24-1
7.24.2-12	The *FUNCTION-DESIGNATOR* **`wscanf`**	As for 7.24-1

7.24.3 Wide character input/output functions

Designated constructs:

DCRN	Definition	Rationale
7.24.3-1	The *FUNCTION-DESIGNATOR* **`fgetwc`**	As for 7.24-1
7.24.3-2	The *FUNCTION-DESIGNATOR* **`fgetws`**	As for 7.24-1
7.24.3-3	The *FUNCTION-DESIGNATOR* **`fputwc`**	As for 7.24-1
7.24.3-4	The *FUNCTION-DESIGNATOR* **`fputws`**	As for 7.24-1
7.24.3-5	The *FUNCTION-DESIGNATOR* **`fwide`**	As for 7.24-1
7.24.3-6	The *FUNCTION-DESIGNATOR* **`getwc`**	As for 7.24-1
7.24.3-7	The *FUNCTION-DESIGNATOR* **`getwchar`**	As for 7.24-1
7.24.3-8	The *FUNCTION-DESIGNATOR* **`putwc`**	As for 7.24-1
7.24.3-9	The *FUNCTION-DESIGNATOR* **`putwchar`**	As for 7.24-1
7.24.3-10	The *FUNCTION-DESIGNATOR* **`ungetwc`**	As for 7.24-1

7.24.4 General wide string utilities

7.24.4.1 Wide string numeric conversion functions

Designated constructs:

DCRN	Definition	Rationale
7.24.4.1-1	The *FUNCTION-DESIGNATOR* **`wcstod`**	As for 7.24-1
7.24.4.1-2	The *FUNCTION-DESIGNATOR* **`wcstof`**	As for 7.24-1
7.24.4.1-3	The *FUNCTION-DESIGNATOR* **`wcstold`**	As for 7.24-1
7.24.4.1-4	The *FUNCTION-DESIGNATOR* **`wcstol`**	As for 7.24-1
7.24.4.1-5	The *FUNCTION-DESIGNATOR* **`wcstoll`**	As for 7.24-1
7.24.4.1-6	The *FUNCTION-DESIGNATOR* **`wcstoul`**	As for 7.24-1
7.24.4.1-7	The *FUNCTION-DESIGNATOR* **`wcstoull`**	As for 7.24-1

7.24.4.2 Wide string copying functions

Designated constructs:

DCRN	Definition	Rationale
7.24.4.2-1	The *FUNCTION-DESIGNATOR* **`wcscpy`**	As for 7.24-1
7.24.4.2-2	The *FUNCTION-DESIGNATOR* **`wcsncpy`**	As for 7.24-1
7.24.4.2-3	The *FUNCTION-DESIGNATOR* **`wmemcpy`**	As for 7.24-1
7.24.4.2-4	The *FUNCTION-DESIGNATOR* **`wmemmove`**	As for 7.24-1

7.24.4.3 Wide string concatenation functions

Designated constructs:

DCRN	Definition	Rationale
7.24.4.3-1	The *FUNCTION-DESIGNATOR* **`wcscat`**	As for 7.24-1
7.24.4.3-2	The *FUNCTION-DESIGNATOR* **`wcsncat`**	As for 7.24-1

7.24.4.4 Wide string comparison functions

Designated constructs:

DCRN	Definition	Rationale
7.24.4.4-1	The *FUNCTION-DESIGNATOR* **`wcscmp`**	As for 7.24-1
7.24.4.4-2	The *FUNCTION-DESIGNATOR* **`wcscoll`**	As for 7.24-1
7.24.4.4-3	The *FUNCTION-DESIGNATOR* **`wcsncmp`**	As for 7.24-1
7.24.4.4-4	The *FUNCTION-DESIGNATOR* **`wcsxfrm`**	As for 7.24-1
7.24.4.4-5	The *FUNCTION-DESIGNATOR* **`wmemcmp`**	As for 7.24-1

7.24.4.5 Wide string search functions

Designated constructs:

DCRN	Definition	Rationale
7.24.4.5-1	The *FUNCTION-DESIGNATOR* **`wcschr`**	As for 7.24-1
7.24.4.5-2	The *FUNCTION-DESIGNATOR* **`wcscspn`**	As for 7.24-1
7.24.4.5-3	The *FUNCTION-DESIGNATOR* **`wcsrchr`**	As for 7.24-1
7.24.4.5-4	The *FUNCTION-DESIGNATOR* **`wcsspn`**	As for 7.24-1
7.24.4.5-5	The *FUNCTION-DESIGNATOR* **`wcsstr`**	As for 7.24-1
7.24.4.5-6	The *FUNCTION-DESIGNATOR* **`wcstok`**	As for 7.24-1
7.24.4.5-7	The *FUNCTION-DESIGNATOR* **`wmemchr`**	As for 7.24-1

7.24.4.6 Miscellaneous functions

Designated constructs:

DCRN	Definition	Rationale
7.24.4.6-1	The *FUNCTION-DESIGNATOR* **`wcslen`**	As for 7.24-1
7.24.4.6-2	The *FUNCTION-DESIGNATOR* **`wmemset`**	As for 7.24-1

7.24.5 Wide character time conversion functions

Designated constructs:

DCRN	Definition	Rationale
7.24.5-1	The *FUNCTION-DESIGNATOR* **`wcsftime`**	As for 7.24-1

7.24.6 Extended multibyte/wide character conversion utilities

7.24.6.1 Single byte/wide character conversion utilities

Designated constructs:

DCRN	Definition	Rationale
7.24.6.1-1	The *FUNCTION-DESIGNATOR* **`btowc`**	As for 7.24-1
7.24.6.1-2	The *FUNCTION-DESIGNATOR* **`wctob`**	As for 7.24-1

7.24.6.2 Conversion state functions

Designated constructs:

DCRN	Definition	Rationale
7.24.6.2-1	The *FUNCTION-DESIGNATOR* **`mbsinit`**	As for 7.24-1

7.24.6.3 Restartable multibyte/wide character conversion functions

Designated constructs:

DCRN	Definition	Rationale
7.24.6.3-1	The *FUNCTION-DESIGNATOR* **`mbrlen`**	As for 7.24-1
7.24.6.3-2	The *FUNCTION-DESIGNATOR* **`mbrtowc`**	As for 7.24-1
7.24.6.3-3	The *FUNCTION-DESIGNATOR* **`wcrtomb`**	As for 7.24-1

7.24.6.4 Restartable multibyte/wide string conversion functions

Designated constructs:

DCRN	Definition	Rationale
7.24.6.4-1	The *FUNCTION-DESIGNATOR* **`mbsrtombs`**	As for 7.24-1
7.24.6.4-2	The *FUNCTION-DESIGNATOR* **`wcsrtombs`**	As for 7.24-1

7.25 Wide character classification functions <wctype.h>

Designated constructs:

DCRN	Definition	Rationale
7.25-1	An *INCLUDE-DIRECTIVE* that causes inclusion of the **`<wctype.h>`** header.	Support for wide characters is **implementation-dependent**.

7.25.1 Introduction

Designated constructs:

DCRN	Definition	Rationale
7.25.1-1	The *typedef-name* **`wctrans_t`**	As for 7.25-1
7.25.1-2	The *typedef-name* **`wctype_t`**	As for 7.25-1

7.25.2 Wide character classification utilities

7.25.2.1 Wide character classification functions

Designated constructs:

DCRN	Definition	Rationale
7.25.2.1.1-1	The *FUNCTION-DESIGNATOR* **`iswalnum`**	As for 7.25-1
7.25.2.1.2-1	The *FUNCTION-DESIGNATOR* **`iswalpha`**	As for 7.25-1
7.25.2.1.3-1	The *FUNCTION-DESIGNATOR* **`iswblank`**	As for 7.25-1
7.25.2.1.4-1	The *FUNCTION-DESIGNATOR* **`iswcntrl`**	As for 7.25-1
7.25.2.1.5-1	The *FUNCTION-DESIGNATOR* **`iswdigit`**	As for 7.25-1
7.25.2.1.6-1	The *FUNCTION-DESIGNATOR* **`iswgraph`**	As for 7.25-1
7.25.2.1.7-1	The *FUNCTION-DESIGNATOR* **`iswlower`**	As for 7.25-1
7.25.2.1.8-1	The *FUNCTION-DESIGNATOR* **`iswprint`**	As for 7.25-1
7.25.2.1.9-1	The *FUNCTION-DESIGNATOR* **`iswpunct`**	As for 7.25-1
7.25.2.1.10-1	The *FUNCTION-DESIGNATOR* **`iswspace`**	As for 7.25-1
7.25.2.1.11-1	The *FUNCTION-DESIGNATOR* **`iswupper`**	As for 7.25-1
7.25.2.1.12-1	The *FUNCTION-DESIGNATOR* **`iswxdigit`**	As for 7.25-1

7.25.2.2 Extensible wide character classification functions

Designated constructs:

DCRN	Definition	Rationale
7.25.2.2.1-1	The *FUNCTION-DESIGNATOR* `iswctype`	As for 7.25-1
7.25.2.2.2-1	The *FUNCTION-DESIGNATOR* `wctype`	As for 7.25-1

7.25.3 Wide character case mapping utilities

7.25.3.1 Wide character case mapping functions

Designated constructs:

DCRN	Definition	Rationale
7.25.3.1.1-1	The *FUNCTION-DESIGNATOR* `towlower`	As for 7.25-1
7.25.3.1.2-1	The *FUNCTION-DESIGNATOR* `towupper`	As for 7.25-1

7.25.3.2 Extensible wide character case mapping functions

Designated constructs:

DCRN	Definition	Rationale
7.25.3.2.1-1	The *FUNCTION-DESIGNATOR* `towctrans`	As for 7.25-1
7.25.3.2.2-1	The *FUNCTION-DESIGNATOR* `wctrans`	As for 7.25-1

7.26 Future library directions

7.26.1 Complex arithmetic `<complex.h>`

Designated constructs:

An *identifier* that is any of the following:		
DCRN	***Identifier***	**Rationale**
7.26.1-1	`cerf`	This name may be added to the declarations in the **`<complex.h>`** header. By avoiding its use in user-written code, users reduce the risk that programs will behave differently under implementations that comply with future revisions of the language standard. **PORTABILITY**
7.26.1-2	`cerff`	As for 7.26.1-1
7.26.1-3	`cerfl`	As for 7.26.1-1
7.26.1-4	`cerfc`	As for 7.26.1-1
7.26.1-5	`cerfcf`	As for 7.26.1-1
7.26.1-6	`cerfcl`	As for 7.26.1-1
7.26.1-7	`cexp2`	As for 7.26.1-1
7.26.1-8	`cexp2f`	As for 7.26.1-1
7.26.1-9	`cexp2l`	As for 7.26.1-1
7.26.1-10	`cexpm1`	As for 7.26.1-1
7.26.1-11	`cexpm1f`	As for 7.26.1-1
7.26.1-12	`cexpm1l`	As for 7.26.1-1
7.26.1-13	`clog10`	As for 7.26.1-1
7.26.1-14	`clog10f`	As for 7.26.1-1
7.26.1-15	`clog10l`	As for 7.26.1-1
7.26.1-16	`clog1p`	As for 7.26.1-1
7.26.1-17	`clog1pf`	As for 7.26.1-1
7.26.1-18	`clog1pl`	As for 7.26.1-1
7.26.1-19	`clog2`	As for 7.26.1-1
7.26.1-20	`clog2f`	As for 7.26.1-1
7.26.1-21	`clog2l`	As for 7.26.1-1
7.26.1-22	`clgamma`	As for 7.26.1-1
7.26.1-23	`clgammaf`	As for 7.26.1-1
7.26.1-24	`clgammal`	As for 7.26.1-1
7.26.1-25	`ctgamma`	As for 7.26.1-1
7.26.1-26	`ctgammaf`	As for 7.26.1-1
7.26.1-27	`ctgammal`	As for 7.26.1-1

7.26.2 Character handling `<ctype.h>`

Designated constructs:

DCRN	Definition	Rationale
7.26.2-1	An *identifier* that begins with `is` or `to` followed by a lowercase letter.	Function names that begin in this manner may be added to the `<ctype.h>` header. By avoiding use of the specified identifiers in user-written code, users reduce the risk that programs will behave differently under implementations that comply with future revisions of the language standard. (**PORTABILITY**)

Note: Since, similar functions whose names begin in a similar manner may also be added to the `<wctype.h>` header (7.26.13), DCRN 7.26.2-1 serves for both cases.

7.26.3 Errors `<errno.h>`

Designated constructs:

DCRN	Definition	Rationale
7.26.3-1	An *identifier* that begins with `E` and a digit or `E` and an uppercase letter.	Macro names that begin in this manner may be added to the `<errno.h>` header. By avoiding use of the specified identifiers in user-written code, users reduce the risk that programs will behave differently under implementations that comply with future revisions of the language standard. (**PORTABILITY**)

7.26.4 Format conversion of integer types `<inttypes.h>`

Designated constructs:

DCRN	Definition	Rationale
7.26.4-1	An *identifier* that begins with `PRI` or `SCN` followed by any lowercase letter or `X`.	Macros names that begin in this manner may be added to the `<inttypes.h>` header. By avoiding use of the specified identifiers in user-written code, users reduce the risk that programs will behave differently under implementations that comply with future revisions of the language standard. (**PORTABILITY**)

7.26.5 Localisation `<locale.h>`

Designated constructs:

DCRN	Definition	Rationale
7.26.5-1	An *identifier* that begins with `LC_` followed by an uppercase letter.	Macro names that begin in this manner may be added to the `<locale.h>` header. By avoiding use of the specified identifiers in user-written code, users reduce the risk that programs will behave differently under implementations that comply with future revisions of the language standard. (**PORTABILITY**)

7.26.6 Signal handling `<signal.h>`

Designated constructs:

DCRN	Definition	Rationale
7.26.6-1	An *identifier* that begins with **`SIG`** or **`SIG_`** followed by an uppercase letter.	Macro names that begin in this manner may be added to the **`<locale.h>`** header. By avoiding use of the specified identifiers in user-written code, users reduce the risk that programs will behave differently under implementations that comply with future revisions of the language standard. (**PORTABILITY**)

7.26.7 Boolean types and values `<stdbool.h>`

Designated constructs:

DCRN	Definition	Rationale
7.26.7-1	Any of the *MACRO-NAME* **`bool`**, **`true`** or **`false`**.	The ability to define and perhaps then redefine the macros **`bool`**, **`true`** and **`false`** is an obsolescent feature. Avoidance of constructs that effect such definitions or redefinitions reduces the risk that a program will behave differently under implementations that comply with future revisions of the standard. (**PORTABILITY**)

7.26.8 Integer types `<stdint.h>`

Designated constructs:

DCRN	Definition	Rationale
7.26.8-1	An *identifier* that begins with **`int`** or **`uint`** and ends in **`_t`**.	Typedef names that begin and end in this manner may be added to the **`<stdint.h>`** header. By avoiding use of the specified identifiers in user-written code, users reduce the risk that programs will behave differently under implementations that comply with future revisions of the language standard. (**PORTABILITY**)
7.26.8-2	An *identifier* that begins with **`INT`** or **`UINT`** and ends with **`_MAX`**, **`_MIN`** or **`_C`**.	Macro names that begin and end in this manner may be added to the **`<stdint.h>`** header. By avoiding use of the specified identifiers in user-written code, users reduce the risk that programs will behave differently under implementations that comply with future revisions of the language standard. (**PORTABILITY**)

7.26.9 Input/output `<stdio.h>`

Designated constructs:

DCRN	Definition	Rationale
7.26.9-1	A *FUNCTION-DESIGNATOR* denoting the **`ungetc`** function at a point where the file position indicator is zero.	Such usage has been designated an obsolescent feature. Its occurrence in user-written code increases the risk that a program may fail under implementations that conform to future revisions of the language standard. (**PORTABILITY**)

7.26.10 General utilities `<stdlib.h>`

Designated constructs:

DCRN	Definition	Rationale
7.26.10-1	An *identifier* that begins with **`str`**, followed by a lowercase letter.	Function names that begin in this manner may be added to the **`<stdlib.h>`** header. By avoiding use of the specified identifiers in user-written code, users reduce the risk that programs will behave differently under implementations that comply with future revisions of the language standard. **(PORTABILITY)**

Note: Since, similar functions whose names begin in a similar manner may also be added to the **`<string.h>`** header, DCRN 7.26.10-1 serves for both cases.

7.26.11 String handling `<string.h>`

Designated constructs:

DCRN	Definition	Rationale
7.26.11-1	An *identifier* that begins with **`mem`** followed by a lowercase letter.	Function names that begin in this manner may be added to the **`<string.h>`** header. By avoiding use of the specified identifiers in user-written code, users reduce the risk that programs will behave differently under implementations that comply with future revisions of the language standard. **(PORTABILITY)**

7.26.12 Extended multibyte and wide character utilities `<wchar.h>`

Designated constructs:

DCRN	Definition	Rationale
7.26.12-1	An *identifier* that begins with **`wcs`**, followed by a lowercase letter.	Function names that begin in this manner may be added to the **`<wchar.h>`** header. By avoiding use of the specified identifiers in user-written code, users reduce the risk that programs will behave differently under implementations that comply with future revisions of the language standard. **(PORTABILITY)**

Note: Since, similar functions whose names begin in a similar manner may also be added to the **`<string.h>`** header, DCRN 7.26.12-1 serves for both cases.

7.26.13 Wide character classification and mapping utilities `<wctype.h>`

Designated constructs:

See 7.26.2.

8 Annex A – Orthosyntax and Parasyntax Summary

8.1 Lexical grammar

8.1.1 Lexical elements

Orthosyntax:

token = *keyword*
| *identifier*
| *constant*
| *string-literal*
| *punctuator* ;

preprocessing-token = *header-name*
| *identifier*
| *pp-number*
| *character-constant*
| *string-literal*
| *operator*
| *punctuator*
| each non-white-space character that cannot be one of the above ;

Parasyntax:

LETTER = *identifier-nondigit* \ _ ;

WORD-TOKEN = *LETTER*
| *WORD-TOKEN* < *LETTER* ;

8.1.2 Keywords

Orthosyntax:

keyword = **auto** | **break** | **case** | **char** | **const** | **continue** |
default | **do** | **double** | **else** | **enum** | **extern** |
float | **for** | **goto** | **if** | **inline** | **int** | **long** |
register | **restrict** | **return** | **short** | **signed** |
sizeof | **static** | **struct** | **switch** | **typedef** |
union | **unsigned** | **void** | **volatile** | **while** | **_Bool** |
_Complex | **_Imaginary** ;

8.1.3 Identifiers

Orthosyntax:

identifier = *identifier-nondigit*
| *identifier* < *identifier-nondigit*
| *identifier* < *digit*

identifier-nondigit = **_** | **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**
| **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**

```
digit                       =     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ;
```

8.1.4 Universal character names

Orthosyntax:

```
universal-character-name    =     \u < hex-quad
                            |     \U < hex-quad  ;

hex-quad                    =     hexadecimal-digit < hexadecimal-digit <
                                            hexadecimal-digit < hexadecimal-digit  ;
```

8.1.5 Constants

Orthosyntax:

```
constant       =     floating-constant
               |     integer-constant
               |     enumeration-constant
               |     character-constant  ;
```

Orthosyntax:

```
integer-constant      =     decimal-constant < [ integer-suffix ]
                      |     octal-constant < [ integer-suffix ]
                      |     hexadecimal-constant < [ integer-suffix ] ;

decimal-constant      =     nonzero-digit
                      |     decimal-constant < digit ;

octal-constant        =     0
                      |     octal-constant < octal-digit ;

hexadecimal-constant  =     hexadecimal-prefix < hexadecimal-digit
                      |     hexadecimal-constant < hexadecimal-digit ;

hexadecimal-constant  =     0x | 0X ;

nonzero-digit         =     1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ;

octal-digit           =     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 ;

hexadecimal-digit     =     1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
                      |     a | b | c | d | e | f
                      |     A | B | C | D | E | F ;

integer-suffix        =     unsigned-suffix < [ long-suffix ]
                      |     unsigned-suffix < long-long suffix
                      |     long-suffix < [ unsigned-suffix ]
                      |     long-long-suffix < [ unsigned-suffix ]  ;

unsigned-suffix       =     u | U ;

long-suffix           =     l | L ;
```

```
long-long-suffix        =       ll | LL ;
```

Orthosyntax:

```
floating-constant               =       decimal-floating-constant
                                |       hexadecimal-floating-constant ;

decimal-floating-constant       =       fractional-constant
                                                < [ exponent-part ] < [ floating-suffix ]
                                |       digit-sequence < exponent-part < [ floating-suffix ] ;

hexadecimal-floating-constant   =       hexadecimal-prefix
                                                <  hexadecimal-fractional-constant
                                                <  binary-exponent-part
                                                <  [ floating-suffix ]
                                |       hexadecimal-prefix
                                                <  hexadecimal-digit-sequence
                                                <  binary-exponent-part
                                                <  [ floating-suffix ] ;

fractional-constant             =       [ digit-sequence ]  <  .  <  digit-sequence
                                |       digit-sequence ;

exponent-part                   =       e  <  [ sign ]  <  digit-sequence
                                |       E  <  [ sign ]  <  digit-sequence ;

sign                            =       +  |  - ;

digit-sequence                  =       digit
                                |       digit-sequence  <  digit ;
hexadecimal-fractional-constant         =       [ hexadecimal-digit-sequence ]  <  .
                                                        <  hexadecimal-digit-sequence
                                        |       hexadecimal-digit-sequence  <  . ;

binary-exponent-part                    =       p  <  [ sign ]  <  digit-sequence
                                        |       P  <  [ sign ]  <  digit-sequence  ;

hexadecimal-digit-sequence      =       hexadecimal-digit
                                |       hexadecimal-digit-sequence  <  hexadecimal-digit ;

floating-suffix                         =       f | l | F | L ;
```

Orthosyntax:

```
enumeration-constant    =       identifier ;
```

Orthosyntax:

```
character-constant              =       `  <  c-char-sequence  <  '  ;
                                |       L  <  '  <  c-char-sequence  <  ' ;

character-constant              =       '  <  c-char-sequence  <  '
                                |       L  <  '  <  c-char-sequence  <  ' ;
```

```
c-char-sequence                =    c-char
                               |    c-char-sequence < c-char ;

c-char                         =    escape-sequence
                               |    any member of the source character set except the
                                    single-quote ', backslash \, or new-line character ;

escape-sequence                =    simple-escape-sequence
                               |    octal-escape-sequence
                               |    hexadecimal-escape-sequence
                               |    universal-character-name ;

simple-escape-sequence         =    \' | \" | \? | \\ | \a | \b
                               |    \f | \n | \r | \t | \v  ;

octal-escape-sequence          =    \ < octal-digit
                               |    \ < octal-digit < octal-digit
                               |    \ < octal-digit < octal-digit < octal-digit ;

hexadecimal-escape-sequence    =    \x < hexadecimal-digit
                               |    hexadecimal-escape-sequence < hexadecimal-digit ;
```

Parasyntax:

```
character-constant             =    INTEGER-CHARACTER-CONSTANT
                               |    WIDE-CHARACTER-CONSTANT  ;

INTEGER-CHARACTER-CONSTANT     =    ` < c-char-sequence < ' ;

WIDE-CHARACTER-CONSTANT        =    L < ' < c-char-sequence < ' ;

VALUE-ESCAPE-SEQUENCE          =    escape-sequence
                               &    OCT-OR-HEX-ESCAPE-SEQUENCE ;

OCT-OR-HEX-ESCAPE-SEQUENCE     =    \ < OCTAL-ESC-DIGITS
                               |    \ < HEXADECIMAL-ESC-DIGITS ;

OCTAL-ESC-DIGITS               =    octal-digit
                               |    octal-digit < octal-digit
                               |    octal-digit < octal-digit < octal-digit ;

HEXADECIMAL-ESC-DIGITS         =    hexadecimal-digit
                               |    HEXADECIMAL-ESC-DIGITS < hexadecimal-digit  ;
```

8.1.6 String literals

Orthosyntax:

```
string-literal          =    " < [ s-char-sequence ] < "
                        |    L" < [ s-char-sequence ] < " ;

s-char-sequence         =    s-char
```

| *s-char-sequence* < *s-char* ;

s-char = *escape-sequence*
| any member of the source character set except the double-quote ", backslash \, or new-line character ;

Parasyntax:

CHARACTER-STRING-LITERAL = " < [*s-char-sequence*] < " ;

WIDE-STRING-LITERAL = L" < [*s-char-sequence*] < " ;

8.1.7 Punctuators

Orthosyntax:

punctuator = [|] | (|) | { | } | . | -> | ++ | -- | & | * | + | -
| ~ | ! | / | % | << | >> | < | > | <= | >= | == | ^ | | | &&
| || | ? | : | ; | ... | = | *= | /= | %= | += | -= | <<=
| >>= | &= | ^= | |= | , | # | ## | <: | :> | <% | %> | %:
| %:%: ;

Parasyntax:

SUBSTITUTE-PUNCTUATOR = <: | :> | <% | %> | %: | %:%: ;

8.1.8 Header names

Orthosyntax:

header-name = < < *h-char-sequence* < >
| " < *q-char-sequence* < " ;

h-char-sequence = *h-char*
| *h-char-sequence* < *h-char* ;

h-char = any member of the source character set except the new-line character and >

q-char-sequence = *q-char*
| *q-char-sequence* < *q-char*

q-char = any member of the source character set except the new-line character and "

Parasyntax:

STD-HEADER-NAME = < < *STD-HU-CHAR-SEQUENCE* < > ;

USER-HEADER-NAME = " < *STD-HU-CHAR-SEQUENCE* < " ;

STD-HU-CHAR-SEQUENCE = *STD-HU-BEFORE-PERIOD* < . < *LETTER* ;

STD-HU-BEFORE-PERIOD = *STD-HU-CHAR* & *LETTER*
| *STD-HU-BEFORE-PERIOD* < *STD-HU-CHAR* ;

STD-HU-CHAR = *LETTER*

```
                    |      digit ;
```

8.1.9 Preprocessing numbers

Orthosyntax:

```
pp-number     =     digit
              |     . < digit
              |     pp-number < digit
              |     pp-number < nondigit
              |     pp-number < e < sign
              |     pp-number < E < sign
              |     pp-number < p < sign
              |     pp-number < P < sign
              |     pp-number < . ;
```

Parasyntax:

```
ALL-DIGIT-PP-NUMBER   =     digit
                      |     ALL-DIGIT-PP-NUMBER < digit ;
```

8.2 Phrase structure grammar

8.2.1 Expressions

Parasyntax:

```
SIDE-EFFECTIVE-OPERATOR    =     ++ | -- | == | *= | /= | %=  | += |
                                 -= | <<= | >>= | &= | ^= | |=
```

Orthosyntax:

```
primary-expr  =     identifier
              |     constant
              |     string-literal
              |     ( expression )
```

Orthosyntax:

```
postfix-expr  =     primary-expr
              |     postfix-expr [ expression ]
              |     postfix-expr ( [ argument-expression-list ] )
              |     postfix-expr   identifier
              |     postfix-expr -> identifier
              |     postfix-expr ++
              |     postfix-expr -- ;
```

```
argument-expression-list:
              assignment-expr
              argument-expression-list , assignment-expr
```

Parasyntax:

```
postfix-expr  =     primary-expr
              |     SUBSCRIPT-EXPRESSION
```

```
                          |         FUNCTION-CALL-EXPRESSION
                          |         DIRECT-ACCESS-EXPRESSION
                          |         INDIRECT-ACCESS-EXPRESSION
                          |         POST-INCREMENT-EXPRESSION
                          |         POST-DECREMENT-EXPRESSION ;

SUBSCRIPT-EXPRESSION          =         postfix-expr [ expression ] ;

FUNCTION-CALL-EXPRESSION      =         postfix-expr ( [ argument-expression-list ] ) ;

DIRECT-ACCESS-EXPRESSION      =         postfix-expr    identifier ;

INDIRECT-ACCESS-EXPRESSION    =         postfix-expr -> identifier ;

POST-INCREMENT-EXPRESSION     =         postfix-expr ++ ;

POST-DECREMENT-EXPRESSION     =         postfix-expr -- ;

argument-expression-list      =         ARGUMENT
                              |         argument-expression-list , ARGUMENT ;

ARGUMENT                      =         assignment-expr ;
```

Orthosyntax:

```
unary-expr              =       postfix-expr
                        |       ++ unary-expr
                        |       -- unary-expr
                        |       unary-operator cast-expr
                        |       sizeof unary-expr
                        |       sizeof ( type-name ) ;

unary-operator          =       & | * | + | - | ~ | ! ;
```

Parasyntax:

```
unary-expr                    =         postfix-expr
                              |         PRE-INCREMENT-EXPRESSION
                              |         PRE-DECREMENT-EXPRESSION
                              |         UNARY-OP-EXPR
                              |         SIZEOF-UNARY-EXPR
                              |         SIZEOF-TYPE-NAME ;

PRE-INCREMENT-EXPRESSION      =         ++ unary-expr ;

PRE-DECREMENT-EXPRESSION      =         -- unary-expr ;

UNARY-OP-EXPR                 =         AMPERSAND-EXPR
                              |         ASTERISK-EXPR
                              |         UPLUS-EXPR
                              |         UMINUS-EXPR
                              |         TILDE-EXPR
                              |         SHRIEK-EXPR ;
```

```
SIZEOF-UNARY-EXPR              =       sizeof unary-expr ;

SIZEOF-TYPE-EXPR               =       sizeof ( type-name ) ;

AMPERSAND-EXPR                 =       & cast-expr ;

ASTERISK-EXPR                  =       * cast-expr ;

UPLUS-EXPR                     =       + cast-expr ;

UMINUS-EXPR                    =       - cast-expr ;

TILDE-EXPR                     =       ~ cast-expr ;

SHRIEK-EXPR                    =       ! cast-expr ;
```

Orthosyntax:

```
cast-expr               =       unary-expr
                        |       ( type-name ) cast-expr  ;
```

Parasyntax:

```
cast-expr               =       unary-expr
                        |       EXPLICIT-CAST-EXPR ;

EXPLICIT-CAST-EXPR      =       ( type-name ) cast-expr  ;
```

Orthosyntax:

```
multiplicative-expr     =       cast-expr
                        |       multiplicative-expr * cast-expr
                        |       multiplicative-expr / cast-expr
                        |       multiplicative-expr % cast-expr  ;
```

Parasyntax:

```
EXPLICIT-MULT-EXPR      =       multiplicative-expr * cast-expr
                        |       EXPLICIT-DIVIDE-EXPR  ;

EXPLICIT-DIVIDE-EXPR    =       multiplicative-expr / cast-expr
                        |       multiplicative-expr % cast-expr  ;
```

Orthosyntax:

```
additive-expr                  =       multiplicative-expr
                               |       additive-expr + multiplicative-expr
                               |       additive-expr - multiplicative-expr  ;
```

Parasyntax:

```
additive-expr                  =       multiplicative-expr
                               |       EXPLICIT-ADDITIVE-EXPR ;
```

```
EXPLICIT-ADDITIVE-EXPR      =    EXPLICIT-PLUS-EXPR
                            |    EXPLICIT-MINUS-EXPR ;

EXPLICIT-PLUS-EXPR          =    additive-expr + multiplicative-expr ;

EXPLICIT-MINUS-EXPR         =    additive-expr – multiplicative-expr ;
```

Orthosyntax:

```
shift-expr     =    additive-expr
               |    shift-expr << additive-expr
               |    shift-expr >> additive-expr
```

Orthosyntax:

```
relational-expr      =    shift-expr
                     |    relational-expr <  shift-expr
                     |    relational-expr >  shift-expr
                     |    relational-expr <= shift-expr
                     |    relational-expr >= shift-expr ;
```

Parasyntax:

```
relational-expr      =    shift-expr
                     |    EXPLICIT-REL-EXPR ;

EXPLICIT-REL-EXPR    =    EXPLICIT- LT-EXPR
                     |    relational-expr >  shift-expr
                     |    relational-expr <= shift-expr
                     |    relational-expr >= shift-expr ;

EXPLICIT- LT-EXPR    =    relational-expr <  shift-expr ;
```

Orthosyntax:

```
equality-expr               =    relational-expr
                            |    equality-expr == relational-expr
                            |    equality-expr != relational-expr ;
```

Parasyntax:

```
equality-expr               =    relational-expr
                            |    EXPLICIT-EQUALITY-EXPR ;

EXPLICIT-EQUALITY-EXPR      |    equality-expr == relational-expr
                            |    equality-expr != relational-expr ;
```

Orthosyntax:

```
AND-expr             =    equality-expr
                     |    AND-expr & equality-expr ;
```

Parasyntax:

```
AND-expr              =   equality-expr
                      |   EXPLICIT-AND-EXPR ;

EXPLICIT-AND-EXPR     |   AND-expr & equality-expr ;
```

Orthosyntax:

```
exclusive-OR-expr          =   AND-expr
                           |   exclusive-OR-expr ^ AND-expr ;
```

Parasyntax:

```
exclusive-OR-expr          =   AND-expr
                           |   EXPLICIT-XOR-EXPR ;

EXPLICIT-XOR-EXPR          |   exclusive-OR-expr ^ AND-expr ;
```

Orthosyntax:

```
inclusive-OR-expr          =   exclusive-OR-expr
                           |   inclusive-OR-expr | exclusive-OR-expr ;
```

Parasyntax:

```
inclusive-OR-expr          =   exclusive-OR-expr
                           |   EXPLICIT-IOR-EXPR ;

EXPLICIT-IOR-EXPR          |   inclusive-OR-expr | exclusive-OR-expr ;
```

Orthosyntax:

```
logical-AND-expr           =   inclusive-OR-expr
                           |   logical-AND-expr && inclusive-OR-expr ;
```

Parasyntax:

```
logical-AND-expr           =   inclusive-OR-expr
                           |   EXPLICIT-LAND-EXPR ;

EXPLICIT-LAND-EXPR         |   logical-AND-expr && inclusive-OR-expr ;
```

Orthosyntax:

```
logical-OR-expr            =   logical-AND-expr
                           |   logical-OR-expr || logical-AND-expr
```

Parasyntax:

```
logical-OR-expr            =   logical-AND-expr
                           |   EXPLICIT-LOR-EXPR ;

EXPLICIT-LOR-EXPR          =   logical-OR-expr || logical-AND-expr ;
```

Orthosyntax:

```
conditional-expr                 =      logical-OR-expr
                                 |      logical-OR-expr ? expr : conditional-expr ;
```

Parasyntax:

```
conditional-expr                 =      logical-OR-expr
                                 |      EXPLICIT-COND-EXPR ;

EXPLICIT-COND-EXPR               =      logical-OR-expr ? expr : conditional-expr ;
```

Orthosyntax:

```
assignment-expr                  =      conditional-expr
                                 |      unary-expr assignment-operator assignment-expr ;

assignment-operator              =      = | *= | /= | %= | += | -=
                                 |      <<= | >>= | &= | ^= | |= ;
```

Parasyntax:

```
assignment-expr                          =      conditional-expr
                                         |      EXPLICIT-ASSIGNMENT-EXPR ;

EXPLICIT-ASSIGNMENT-EXPR                 =      EXPLICIT-SIMPLE-ASSIGNMENT-EXPR
                                         |      EXPLICIT-MULT-ASSIGNMENT-EXPR
                                         |      EXPLICIT-DIVIDE-ASSIGNMENT-EXPR
                                         |      EXPLICIT-MOD-ASSIGNMENT-EXPR
                                         |      EXPLICIT-PLUS-ASSIGNMENT-EXPR
                                         |      EXPLICIT-MINUS-ASSIGNMENT-EXPR
                                         |      EXPLICIT-SHIFT-ASSIGNMENT-EXPR
                                         |      EXPLICIT-SHIFT-ASSIGNMENT-EXPR
                                         |      EXPLICIT-BITWISE-ASSIGNMENT-EXPR ;

EXPLICIT-SIMPLE-ASSIGNMENT-EXPR          =      unary-expr = assignment-expr ;

EXPLICIT-MULT-ASSIGNMENT-EXPR            =      unary-expr *= assignment-expr ;

EXPLICIT-DIVIDE-ASSIGNMENT-EXPR          =      unary-expr /= assignment-expr ;

EXPLICIT-MOD-ASSIGNMENT-EXPR             =      unary-expr %= assignment-expr ;

EXPLICIT-PLUS-ASSIGNMENT-EXPR            =      unary-expr += assignment-expr ;

EXPLICIT-MINUS-ASSIGNMENT-EXPR           =      unary-expr -= assignment-expr ;

EXPLICIT-SHIFT-ASSIGNMENT-EXPR           =      EXPLICIT-LSHIFT-ASSIGNMENT-EXPR
                                         |      EXPLICIT-RSHIFT-ASSIGNMENT-EXPR ;

EXPLICIT-LSHIFT-ASSIGNMENT-EXPR          =      unary-expr <<= assignment-expr ;

EXPLICIT-RSHIFT-ASSIGNMENT-EXPR          =      unary-expr >>= assignment-expr ;

EXPLICIT-BITWISE-ASSIGNMENT-EXPR         =      EXPLICIT-AND-ASSIGNMENT-EXPR
                                         |      EXPLICIT-XOR-ASSIGNMENT-EXPR
```

| *EXPLICIT-IOR-ASSIGNMENT-EXPR* ;

EXPLICIT-AND-ASSIGNMENT-EXPR = *unary-expr* **&=** *assignment-expr* ;

EXPLICIT-XOR-ASSIGNMENT-EXPR = *unary-expr* **^=** *assignment-expr* ;

EXPLICIT-IOR-ASSIGNMENT-EXPR = *unary-expr* **|=** *assignment-expr* ;

Orthosyntax:

comma-expression = *assignment-expr*
| *expression* **,** *assignment-expr* ;

Parasyntax:

comma-expression = *assignment-expr*
| *EXPLICIT-COMMA-EXPRESSION* ;

EXPLICIT-COMMA-EXPRESSION= *expression* **,** *assignment-expr* ;

Orthosyntax:

constant-expr = *conditional-expr* ;

8.2.2 Declarations

Orthosyntax:

declaration = *declaration-specifiers* [*init-declarator-list*] ;

declaration-specifiers = *storage-class-specifier* [*declaration-specifiers*]
| *type-specifier* [*declaration-specifiers*]
| *type-qualifier* [*declaration-specifiers*] ;

init-declarator-list = *init-declarator*
| *init-declarator-list* **,** *init-declarator* ;

init-declarator = *declarator*
| *declarator* **=** *initializer* ;

Orthosyntax:

storage-class-specifier = **typedef**
| **extern**
| **static**
| **auto**
| **register** ;

Orthosyntax:

type-specifier = **void**
| **char**
| **short**
| **int**
| **long**

```
                              |       float
                              |       double
                              |       signed
                              |       unsigned
                              |       _Bool
                              |       _Complex
                              |       _Imaginary
                              |       struct-or-union-specifier
                              |       enum-specifier
                              |       typedef-name ;
```

Orthosyntax:

```
struct-or-union-specifier   = [ struct-or-union identifier ] { struct-declaration-list }
                            | struct-or-union identifier  ;

struct-or-union               =       struct
                              |       union ;

struct-declaration-list       =       struct-declaration
                              |       struct-declaration-list struct-declaration ;

struct-declaration            =       specifier-qualifier-list struct-declarator-list ;

specifier-qualifier-list      =       type-specifier [ specifier-qualifier-list ]
                              |       type-qualifier [ specifier-qualifier-list ] ;

struct-declarator-list        =       struct-declarator
                              |       struct-declarator-list , struct-declarator ;

struct-declarator             =       declarator
                              |       [ declarator ] : constant-expr ;
```

Parasyntax:

```
struct-or-union-specifier   = [ struct-or-union SU-IDENTIFIER ] { struct-declaration-list }
                            | struct-or-union SU-IDENTIFIER  ;

SU-IDENTIFIER                 =       identifier  ;

struct-declarator             =       declarator
                              |       BIT-FIELD-DECLARATOR ;

BIT-FIELD-DECLARATOR          =       [ declarator ] : constant-expr ;
```

Orthosyntax:

```
enum-specifier        =       enum [ identifier ]  { enumerator-list }
                      |       enum [ identifier ]  { enumerator-list  ,  }
                      |       enum identifier ;

enumerator-list       =       enumerator
                      |       enumerator-list , enumerator ;
```

```
enumerator                =       enumeration-constant
                          |       enumeration-constant = constant-expression ;
```

Parasyntax:

```
enum-specifier            =       enum [ ENUM-IDENTIFIER ]  { enumerator-list }
                          |       enum [ ENUM-IDENTIFIER ]  { enumerator-list  ,  }
                          |       enum ENUM- IDENTIFIER ;

ENUM-IDENTIFIER           =       identifier ;
```

Orthosyntax:

```
type-qualifier            =       const
                          |       restrict
                          |       volatile ;
```

Orthosyntax:

```
function-specifier        =       inline ;
```

Orthosyntax:

```
declarator                        =       [ pointer ] direct-declarator ;

direct-declarator                 =       identifier
                                  |       ( declarator )
                                  |       direct-declarator [ [ constant-expr ] ]
                                  |       direct-declarator ( parameter-type-list )
                                  |       direct-declarator ( [ identifier-list ] ) ;
```

Parasyntax:

```
declarator                        =       POINTER-DECLARATOR
                                  |       NON-POINTER-DECLARATOR  ;

POINTER-DECLARATOR                =       pointer  direct-declarator ;

NON-POINTER-DECLARATOR            =       direct-declarator ;

direct-declarator                 =       DD-IDENTIFIER
                                  |       DEC-IN-PAREN
                                  |       ARRAY-DECLARATOR
                                  |       FUNCTION-DECLARATOR ;

DD-IDENTIFIER                     =       identifier  ;

DEC-IN-PAREN                      =       ( declarator ) ;

ARRAY-DECLARATOR                  =       direct-declarator ARRAY-BOUND ;

ARRAY-BOUND                       =       [ [ constant-expr ] ] ;

FUNCTION-DECLARATOR               =       FUNCTION-PROTOTYPE
                                  |       K-AND-R-FUNCTION-DECLARATOR ;
```

```
FUNCTION-PROTOTYPE     =     direct-declarator ( parameter-type-list ) ;

K-AND-R-FUNCTION-DECLARATOR     =     direct-declarator ( [ identifier-list ] ) ;
```

Orthosyntax:

```
pointer                =     * [ type-qualifier-list ]
                       |     * [ type-qualifier-list ] pointer ;

type-qualifier-list    =     type-qualifier
                       |     type-qualifier-list type-qualifier ;
```

Orthosyntax:

```
parameter-type-list          =     parameter-list
                             |     parameter-list ,  ... ;

parameter-list               =     parameter-declaration
                             |     parameter-list , parameter-declaration ;

parameter-declaration        =     declaration-specifiers declarator
                             |     declaration-specifiers [ abstract-declarator ] ;

identifier-list              =     identifier
                             |     identifier-list , identifier ;
```

Parasyntax:

```
parameter-declaration        =     declaration-specifiers PARAMETER-DECLARATOR
                             |     declaration-specifiers [ abstract-declarator ] ;

PARAMETER-DECLARATOR         =     declarator ;
```

Orthosyntax:

```
type-name                  =  specifier-qualifier-list [ abstract-declarator ] ;

abstract-declarator        =  pointer
                           |  [ pointer ] direct-abstract-declarator ;

direct-abstract-declarator =  ( abstract-declarator )
                           |  [ direct-abstract-declarator ] [ [ constant-expression ] ]
                           |  [ direct-abstract-declarator ] ( [ parameter-type-list ] ) ;
```

Orthosyntax:

```
typedef-name           =     identifier ;
```

Orthosyntax:

```
initializer            =     assignment-expr
                       |     { initializer-list }
                       |     { initializer-list ,  } ;
```

initializer-list	=	*initializer*
	|	*initializer-list* **`,`** *initializer* ;

8.2.3 Statements

Orthosyntax:

statement	=	*labeled-statement*
	|	*compound-statement*
	|	*expression-statement*
	|	*selection-statement*
	|	*iteration-statement*
	|	*jump-statement* ;

Orthosyntax:

labeled-statement	=	*identifier* **`:`** *statement*
	|	**`case`** *constant-expr* **`:`** *statement*
	|	**`default`** **`:`** *statement* ;

Parasyntax:

labeled-statement	=	*IDENTIFIER-LABELED-STATEMENT*
	|	*CASE-LABELED-STATEMENT*
	|	*DEFAULT-LABELED-STATEMENT* ;
IDENTIFIER-LABELED-STATEMENT	=	*identifier* **`:`** *statement* ;
CASE-LABELED-STATEMENT	=	**`case`** *constant-expr* **`:`** *statement* ;
DEFAULT-LABELED-STATEMENT	=	**`default`** **`:`** *statement* ;

Orthosyntax:

compound-statement	=	**`{`** [*declaration-list*] [*statement-list*] **`}`** ;
declaration-list	=	*declaration*
	|	*declaration-list declaration* ;
statement-list	=	*statement*
	|	*statement-list statement* ;

Orthosyntax:

expression-statement	=	[*expression*] ;

Orthosyntax

selection-statement	=	**`if`** **`(`** *expression* **`)`** *statement*
	|	**`if`** **`(`** *expression* **`)`** *statement* **`else`** *statement*
	|	**`switch`** **`(`** *expression* **`)`** *statement ;*

Parasyntax

selection-statement	=	*BINARY-SELECTION*
	|	*SWITCH-STMT* ;

```
BINARY-SELECTION          =      PLAIN-IF-STMT
                          |      IF-ELSE-STMT ;

PLAIN-IF-STMT             =      if ( IF-EXPR ) TRUE-STMT ;

IF-ELSE-STMT              =      if ( IF-EXPR ) TRUE-STMT else FALSE-STMT ;

IF-EXPR                   =      expression ;

EXPLICIT-LOGICAL-EXPR     =      EXPLICIT-REL-EXPR
                          |      EXPLICIT-EQUALITY-EXPR
                          |      EXPLICIT-LAND-EXPR
                          |      EXPLICIT-LOR-EXPR
                          |      ! ( EXPLICIT-LOGICAL-EXPR ) ;

TRUE-STMT                 =      statement ;

FALSE-STMT                =      statement ;

SWITCH-STMT               =      switch ( SWITCH-EXPR ) SWITCH-BODY ;

SWITCH-EXPR               =      expression ;

SWITCH-BODY               =      statement ;

STRUC-SWITCH-STMNT        =      switch ( SWITCH-EXPR ) STRUC-SWITCH-BODY ;

STRUC-SWITCH-BODY         =      { CASE-CLAUSES ; DEFAULT-CLAUSE } ;

CASE-CLAUSES              =      CASE-CLAUSE
                          |      CASE-CLAUSES ; CASE-CLAUSE ;

CASE-CLAUSE               =      case constant-expr : CASE-GROUP ;

DEFAULT-CLAUSE                   =      default : CASE-GROUP ;

CASE-GROUP                =      { statement-list ; break } ;
```

Orthosyntax:

```
iteration-statement   =    while ( expression ) statement
                      |    do statement while ( expression ) ;
                      |    for ( clause-1 ; expression-2 ; expression-3 ) statement ;
```

Parasyntax:

```
iteration-statement   =    WHILE-STATEMENT
                      |    DO-WHILE-STATEMENT
                      |    FOR-STATEMENT ;

WHILE-STATEMENT       =    while ( WHILE-EXPRESSION ) BODY ;

DO-WHILE-STATEMENT    =    do BODY while ( WHILE-EXPRESSION ) ;
```

```
FOR-STATEMENT        =    for  ( clause-1 ; expression-2 ; expression-3 ) BODY ;

WHILE-EXPRESSION     =    expression ;

BODY                 =    statement ;
```

Orthosyntax:

```
jump-statement       =    goto identifier ;
                     |    continue  ;
                     |    break  ;
                     |    return [ expression ] ; ;
```

Parasyntax:

```
jump-statement       =    GOTO-STATEMENT
                     |    CONTINUE-STATEMENT
                     |    BREAK-STATEMENT
                     |    RETURN-STATEMENT ;

GOTO-STATEMENT       =    goto identifier ; ;

CONTINUE-STATEMENT   =    continue  ; ;

BREAK-STATEMENT      =    break  ; ;

RETURN-STATEMENT     =    PLAIN-RETURN-STMNT
                     |    EXPR-RETURN-STMNT  ;

PLAIN-RETURN-STMNT   =    return  ; ;

EXPR-RETURN-STMNT    =    return [ expression ] ; ;
```

8.2.4 External definitions

Orthosyntax:

```
translation-unit     =    external-declaration
                     |    translation-unit external-declaration ;

external-declaration =    function-definition
                     |    declaration
```

Orthosyntax:

```
function-definition  =    [ declaration-specifiers ]  declarator  [ declaration-list ]
                                    compound-statement ;

declaration-list     =    declaration
                     |    declaration-list declaration;
```

Parasyntax:

8.3 Preprocessing directives

Orthosyntax:

```
preprocessing-file      =       [ group ] ;

group                   =       group-part
                        |       group  group-part ;

group-part              =       [ pp-tokens ] new-line
                        |       if-section
                        |       control-line ;

if-section              =       if-group  [ elif-groups ]  [ else-group ]  endif-line ;

if-group                =       # if  constant-expr new-line  [ group ]
                        |       # ifdef  identifier new-line  [ group ]
                        |       # ifndef  identifier new-line [ group ] ;

elif-groups             =       elif-group
                        |       elif-groups elif-group ;

elif-group              =       # elif  constant-expr new-line [ group ] ;

else-group              =       # else  new-line [ group ] ;

endif-line              =       # endif  new-line ;

control-line            =       # include  pp-tokens new-line
                        |       # define  identifier replacement-list new-line
                        |       # define  identifier  lparen  [ identifier-list ]
                                        replacement-list new-line
                        |       # undef  identifier new-line
                        |       # line  pp-tokens new-line
                        |       # error  [ pp-tokens ]  new-line
                        |       # pragma  [ pp-tokens ]  new-line
                        |       #   new-line ;

lparen                  =       a left-parentheses without preceding white space ;

replacement-list        =       [ pp-tokens ] ;

pp-tokens               =       preprocessing-token
                        |       pp-tokens preprocessing-token ;

new-line                =       the new-line character ;
```

Parasyntax:

```
if-group                        =       IF-DIRECTIVE  [ group ] ;
                                |       IFDEF-DIRECTIVE  [ group ] ;
                                |       IFNDEF-DIRECTIVE  [ group ] ;

IF-DIRECTIVE                    =       # if  constant-expr new-line ;
```

```
IFDEF-DIRECTIVE            =   # ifdef identifier new-line ;

IFNDEF-DIRECTIVE           =   # ifndef identifier new-line ;

elif-group                 =   ELIF-DIRECTIVE [ group ] ;

ELIF-DIRECTIVE             =   # elif constant-expr new-line ;

else-group                 =   ELSE-DIRECTIVE [ group ] ;

ELSE-DIRECTIVE             =   # else new-line ;

endif-line                 =   ENDIF-DIRECTIVE ;

ENDIF-DIRECTIVE            =   # endif new-line ;

control-line               =   INCLUDE-DIRECTIVE
                           |   PLAIN-DEFINE-DIRECTIVE
                           |   FLIKE-DEFINE-DIRECTIVE
                           |   UNDEF-DIRECTIVE
                           |   LINE-DIRECTIVE
                           |   ERROR-DIRECTIVE
                           |   PRAGMA-DIRECTIVE
                           |   NULL-DIRECTIVE ;

INCLUDE-DIRECTIVE          =   # include pp-tokens new-line ;

PLAIN-DEFINE-DIRECTIVE     =   # define identifier replacement-list new-line ;

FLIKE-DEFINE-DIRECTIVE     =   # define identifier < ( [ identifier-list ]
                                     replacement-list  new-line ;

DEFINE-DIRECTIVE           =   PLAIN-DEFINE-DIRECTIVE
                           |   FLIKE-DEFINE-DIRECTIVE ;

PAREN-REPLACEMENT-LIST     =   ( replacement-list ) ;

UNDEF-DIRECTIVE            =   # undef identifier new-line ;

LINE-DIRECTIVE             =   # line pp-tokens new-line ;

ERROR-DIRECTIVE            =   # error [ pp-tokens ]  new-line ;

PRAGMA-DIRECTIVE           =   # pragma [ pp-tokens ]  new-line ;

NULL-DIRECTIVE             =   #  new-line ;

DIRECTIVE                  =   IF-DIRECTIVE
                           |   IFDEF-DIRECTIVE
                           |   IFNDEF-DIRECTIVE
                           |   ELIF-DIRECTIVE
|        ELSE-DIRECTIVE
|        ENDIF-DIRECTIVE
|        INCLUDE-DIRECTIVE
|        PLAIN-DEFINE-DIRECTIVE
|        FLIKE-DEFINE-DIRECTIVE
|        UNDEF-DIRECTIVE
|        LINE-DIRECTIVE
|        ERROR-DIRECTIVE
|        PRAGMA-DIRECTIVE
|        NULL-DIRECTIVE ;
```

9 Annex B – Library summary (NR)

This page intentionally left blank

10 Annex C – Sequence points

This page intentionally left blank

11 Annex D - Universal character names for identifiers

This page intentionally left blank

12 Annex E – Implementation limits

This page intentionally left blank

13 Annex F – IEC 60559 floating-point arithmetic

This page intentionally left blank

14 Annex G – IEC 60559-compatible complex arithmetic

This page intentionally left blank

15 Annex H – Language-independent arithmetic

This page intentionally left blank

16 Annex I – Common warnings

This page intentionally left blank

17 Annex J – Portability issues

This page intentionally left blank