|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SE-001 | |  | |  | |  | | te | | We believe that the accepted proposal of **n2510**, is detrimental to the C standard. The proposal removes the requirement for parameter names when declaring a function. We believe this removes the ability for implementers to issue errors for a reasonably common mistake. This will make C a harder language to use.  The paper makes the assertion that an implementation can assume that an anonymous parameter is intentional, and we think this is dubious. Very few C developers have access to the standard, and write code C using features of the latest version, or are even aware of its latest features. It is our estimation that even once the standard has been ratified and implemented, very few C programmers will recognize when or why to use anonymous parameters for optimization purposes, and the paper does not present any implementations that have been able to report measurable performance increases.  What this change does is that it deprives users of a basic error diagnostic that is today available. A C beginner may write code like:  void function(int)  {  printf("a = %i", a);  }  int main()  {  int a = 42;  function(a);  return TRUE;  }  ... And wonder "Why does the compiler say I haven't defined a, when I am passing it in?". Today the user would hopefully get a clear error from the implementation that the parameter requires a name. This won't be possible in C2X. For an experienced programmer a simple typo can become a bug orders of magnitude harder to detect. Consider:  MyStruct \*s /\* commonly used param/variable name for pointer to MyStruct \*/  void function(/\* lots of params\*/ , MyStruct \*) /\* ops forgot to add param name "s" \*/  {  s->param = 42;  }  ...  function(/\* lots of params\*/ , q);  This simple one character typo is now infinitely harder to find and diagnose because the compiler has to assume that the lack of a parameter name is intentional. Implementers will be put in a very difficult position, where they no longer can issue an error for something that in 99% of cases is a basic mistake. Even issuing a warning will be controversial since it prevents anyone from using this entirely ISO C compliant feature in warning free code.  We think an omission is a poor indicator of intent.  Further the design of this feature does not fully achieve its stated goal. If we look at the declaration:  extern void function(int);  And compare it to the definition:  void function(int)  {  ...  }  The function signatures are no longer the same. In the declaration the omission of a parameter name has no meaning, whereas in the definition the omission does have a meaning. Any caller of this function that wants to optimize its call can therefore only do so if it exists in the same translation unit, in order to be able to assume that the parameter is not used.  Aside: (Allowing anonymous parameters in function declarations was in our opinion a mistake to begin with. We should strive for a clear, explicit language that has strict requirements that catch any typing mistake. Any claim of saving typing by omitting parameter names clearly doesn't hold water, since the programmer has to type the signature with parameter names in the definition of the function, and can simply copy that signature. Removing parameters from the signature takes more work, and reduces readability and clarity, something we should consider a loose-loose.)  The proposal assumes that there are optimizations that can be made when calling a function if you know that not all parameters will be used. If we assume the author is right despite not providing any measurements, this optimization would only be applicable if:  -The function is used as a function pointer, and therefore has to conform to a signature that includes useless params.  -The function is also called directly, and only then can the caller skip some unused params. (The compiler can not optimize this when calling a function pointer, since it may point to a function that does use the parameter.)  -The caller that calls directly is in the same translation unit as the function it's calling.  This seems like a very slim use case, where a user interested in increasing performance should instead consider inlining.  C2X does add a new standardized way to attach attributes. This functionality is much better suited as an attribute. An attribute could also be included in a declaration, and would explicitly unambiguously show user intent. We do note that this feature was added before the attribute format was introduced. While an attribute version of the feature would alleviate many of our concerns with the change, we are still not convinced an attribute version is justified. It still only applies to functions that are used as function pointers when they are not used as function pointers. To change the spec for such a corner case, we should require implementations of the optimization along with measurable improvements in performance in real world applications. We believe that even an attribute version of this feature is bloating the standard without a clear, common and measurable benefit.  If in some future version of C, the wg14 does decide to add this feature in the form of an attribute, then we will not be able to undo the damage done by this change. Implementations will still be forced to not issue errors on parameters without names because doing so would break backward compatibility with c2X.  **n2953** Type inference for object definitions / Under specified Types.  We believe that the proposal, that allows the use of auto to infer the type from initialization, will have a detrimental effect on safety and reliability if that chooses to use it. In C different types can have very different behaviors. The most obvious case is if the different ways signed and unsigned integers handle overflow. Not explicitly stating the type of but to rather rely on an implicit assumption, is in our view a serious risk.  Consider the following:  auto limit = MY\_LIMIT;  if(limit + add < limit) /\* overflow protection \*/  return;  The user here assumes that MY\_LIMIT is an unsigned int. Since the unsigned int has a well defined overflow behavior, the overflow test is entirely valid. However, should the MY\_LIMIT define be a signed int, then overflow is undefined, and a compiler may choose to optimize away the overflow protection. The implementation will issue no warning if the assumption is wrong. If instead the user would have explicitly declared that they expect a limit to be, say an unsigned int, then the code would not break, and if MY\_LIMIT would be out of range of the unsigned int, the implementation can issue a clear diagnostic.  For this reason we believe that this functionality will be discurraged by organizations like MISRA, and various style guides, since it actively prevents compilers and other diagnostic tools from verifying that the actions of the program matches the programmers intentions. We believe that the wg14 should not introduce new functionality whose use is likely to be discouraged from a safety and reliability perspective.  While we believe this to be functionality to be adopted by a very small minority of C programmers. This however leaves the possibility of accidental use, when a user forgets to specify a type properly and the implementation is no longer able to issue a diagnostic, because it is forced to assume that the omission is intentional. | | Remove n2510, and n2953 | |  | |
| GB-002 | |  | | 2 | | 5 | | ed | | The URL given here now redirects and the new version is shorter and friendlier. | | Change the URL in this paragraph to: https://standards.iso.org/ittf/PubliclyAvailableStandards/ | |  | |
| GB-003 | |  | | 2 | | 9, 10 | | ed | | www.unicode.org now uses https so the links should be updated accordingly. | | Change http to https in both paragraphs. | |  | |
| GB-004 | |  | | 2 | | 9, 10, 11 | | ed | | Have the requirements of ISO/IEC Directives Part 2 §10.2 items a through e (on requirements for having non-ISO/IEC normative references) been satisfied and recorded? | |  | |  | |
| GB-005 | |  | | 3 | | 2 | | ed | | Electropedia now uses https so the link should be updated accordingly. | | Change http to https in second bullet point. | |  | |
| CA2-006 | |  | | 5.1.1.2 | | 1 | | Te | | The removal of trigraphs makes it impossible for processing of C source code in code pages with variant code points for characters in the basic character set in a guaranteed deterministic way for all programs. | | Add to the end of 5.1.1.2#1: Trigraph sequences are replaced by corresponding single-character internal representations.  Add in a new section, 5.2.1.1 Trigraph sequences with the following body: Before any other processing takes place, each occurrence of one of the following sequences of three characters (called *trigraph sequences*)17) is replaced with the corresponding single character.  **??= # ??( [ ??/ \**  **??) ] ??’ ^ ??< {**  **??! | ??> } ??- ~**  No other trigraph sequences exist. Each **?** that does not begin one of the trigraphs listed above is not changed. | |  | |
| GB-007 | |  | | 5.1.2.3, 5.2.4.2.2, 6.7.1 | | 12 | | ed | | The terms “single-precision” and “double-precision” are not defined in this document and the definitions in ISO/IEC 2382 (in terms of use of one or two computer words to represent a value) are not appropriate for their uses in this document because there is no guarantee that the types float or double correspond to those definitions. Furthermore, IEC 60559 refers to the formats as binary32 and binary64, not single and double. | | In 5.1.2.3 paragraph 12, change “single-precision” to “float” and “double-precision” to “double” (in both cases formatted as C keywords).  In 5.2.4.2.2 paragraph 29, change “single-precision” to “binary32” and “double-precision” to “binary64”.  In 6.7.1 paragraph 16 (Example 2), change “single precision” and “32-bit single-precision” to “binary32”.  Also ensure associated index entries for the “single-precision” and “double-precision” terms are removed. | |  | |
| GB-008 | |  | | 5.1.2.4 | | 1 | | ed | | The description of when threads are supported (for hosted implementations, implementation-defined for freestanding implementations) fails to take account of threads being optional in hosted implementations. | | Change “hosted implementation” to “hosted implementation that does not define \_\_STDC\_NO\_THREADS\_\_”. | |  | |
| GB-009 | |  | | 5.2.1 | | 1 | | ed | | “zero or more locale-specific members” doesn’t reflect that there are now three required members of the extended character set. | | Change “zero or more” to “three or more”, with a footnote “The extended characters include at least @, $, and `.”. | |  | |
| GB-010 | |  | | 5.2.1 | | 3 | | ed | | “N” in the table of uppercase letters is in bold, unlike the rest of the letters. | | For this table, disable whatever formatting causes “N” to be in bold. | |  | |
| US 3  -011 | |  | | 5.2.1 | | 3 | | te | | Adopt Wording Alternative #1 from N3046 <https://www.open-std.org/jtc1/sc22/WG14/www/docs/n3046.pdf> | | If any other characters are encountered in a source file (except in an identifier, a character constant, a string literal, a header name, a comment, or a preprocessing token that is never converted to a token), the behavior is undefined. The $ character is reserved for use in identifiers as an implementation-defined extension. | |  | |
| GB-012 | |  | | 5.2.4.1, 6.4.3 | |  | | ed | | The 2011 edition of ISO/IEC 10646 removed eight-digit short identifiers that were present in the 2003 edition (and this removal still applies as of the 2020 edition). Thus, the C standard should only use short identifiers with no more than six digits. | | In 5.2.4.1 paragraph 1, change “0000FFFF” to “00FFFF” and change “00010000” to “010000”.  In 6.4.3 paragraph 4, change “\Unnnnnnnn” to “\U00nnnnnn”, “eight digit” to “six digit” (twice), “nnnnnnnn” to “nnnnnn” and “0000nnnn” to “00nnnn”. At the end of footnote 80, add “If either of the first two digits after \U is not 0, this violates a constraint.”. | |  | |
| US 4  -013 | |  | | 5.2.4.2.1 | | Para 5 | | te | | The C Standard considers the language and the library components to both be combined into the “implementation”, but the reality of modern C implementations is that the compiler is potentially produced by a different vendor than the standard library. The requirements added by N2359 poses burdens for the latter kind of implementation scenario because the compiler has no control over the standard library features and the standard library has no control over the host compiler, so there is no way to know how to define these macros. As a concrete example, the Windows SDK is what vends the C Standard Library on Windows (used by MSVC and Clang, at a minimum), so it's quite easy to have an old version of a compiler that's linking against a new version of a standard library (or vice versa). | | We recommend reverting N2359 from C2x to give more time to explore the design space. | |  | |
| GB-014 | |  | | 5.2.4.2.2 | | 15 | | te | | Stating that floating values shall be constant expressions ensures they can be used in static initializers, but the stronger condition of being **arithmetic** constant expressions (rather than some other kind of implementation-defined constant expression permitted in initializers) is needed for them to be usable in constexpr initializers. | | Assuming those macros should be usable in constexpr initializers, change “all floating values shall be constant expressions” to “all floating values shall be arithmetic constant expressions”. | |  | |
| GB-015 | |  | | 5.2.4.2.2 | | 24 | | ed | | In the description of the \*\_MIN\_10\_EXP and \*\_MAX\_10\_EXP macros, “log” should not be in italics. | | Change to roman font (LaTeX \log), in both places. | |  | |
| GB-016 | |  | | 5.2.4.2.2 | | 29 | | ed | | When the value 2 for \*\_IS\_IEC\_60559 was removed editorially at a late stage, some uses of it were left in examples. | | Change the values of FLT\_IS\_IEC\_60559 and DBL\_IS\_IEC\_60559 to 1 in Example 2. | |  | |
| GB-017 | |  | | 5.2.4.2.2 | | 5, 6 | | ed | | The subscript “min” should not be in italics. | | Change to roman font (LaTeX \min), in both places. | |  | |
| US 5  -018 | |  | | 5.2.4.2.2 | | paragraph 18 | | te | | “match(es)” is used in a meaningful way in F.2 but is ambiguous here in the more general context of 5.2.4.2.2. | | Replace the paragraph with:  Whether a type has the same precision (*p*) and exponent range (*e*min ≤ *e* ≤ *e*max) as an IEC 60559 format is characterized by the implementation-defined values of FLT\_IS\_IEC\_60559, DBL\_IS\_IEC\_60559 and LDBL\_IS\_IEC\_60559 (this does not imply conformance to Annex F):  0 type does not have the precision and exponent range of an IEC 60559 format  1 type has the precision and exponent range of an IEC 60559 format | |  | |
| GB-019 | |  | | 5.2.4.2.3 | | 10 | | ed | | In the table of preferred quantum exponents, the entry for increment and decrement operators shows “min” in italics when other entries have it in roman font. | | Change to roman font (LaTeX \min). | |  | |
| GB-020 | |  | | 5.2.4.2.3 | | 8 | | ed | | The upper bound on the coefficient given is incorrect. | | Change (LaTeX notation) $10^{(p-1)}$ to $10^p-1$. | |  | |
| US 8  -021 | |  | | 5.2.4.2.3 | | Preferred quantum exponents table | | ed | | Beginning with the row for compoundn, the typeface style for the arguments n, x, and y are incorrect and inconsistent with the table entries above. | | Beginning with the row for compoundn, fix the typeface style for the arguments n, x, and y to be consistent with the table entries above. | |  | |
| US 7  -022 | |  | | 5.2.4.2.3 | | Preferred quantum exponents table | | ed | | In the row for postfix and prefix operators, in the second column, “min” is not italicized (as it is in other cells). | | In the row for postfix and prefix operators, in the second column, italicize “min” (as in other cells). Make other typeface changes if needed for consistency. | |  | |
| US 6  -023 | |  | | 5.2.4.2.3 | | Preferred quantum exponents table | | ed | | The Operation column contains C operations, not mathematical ones. Thus, hyphen in program font should be used instead of minus math symbol, and likewise “+” in program font should be used instead of plus math symbol. | | In the Operation column, use hyphen in program font instead of minus math symbol, and use “+” in program font instead of plus math symbol. | |  | |
| GB-024 | |  | | 6.2.2 | | 5 | | te | | The specification of linkage for file-scope objects fails to cover the case of objects declared as thread\_local without static or extern. | | Change “no storage-class specifier or only the specifier auto” to “does not contain the storage-class specifier static or constexpr”. | |  | |
| GB-025 | |  | | 6.2.4 | | 2 | | ed | | There is a stray “.” in mid-sentence immediately before the footnote number 35, and a missing “.” at the end of that sentence. | | Move the “.” from before footnote number 35 to before footnote number 36. | |  | |
| GB-026 | |  | | 6.2.5 | | 13 | | ed | | The sentence “Decimal floating types are real floating types.” seems entirely superfluous, because the very next paragraph includes decimal floating types in the definition of real floating types. | | Remove that sentence. | |  | |
| GB-027 | |  | | 6.2.5 | | 14 | | ed | | “real floating types” is being defined in this paragraph, so should be in italics to mark it as a defined term. | | Change “real floating types” into italics. | |  | |
| GB-028 | |  | | 6.2.5, 6.3.1.1, 6.5.16.1, 6.7.10, 7.20.1, J.3.4 | |  | | ed | | The seven uses of the “plain char” phrase (two appear in 7.20.1) in the document are inconsistent about whether the word “plain” is quoted or not. | | Make them consistent, with one choice or the other. Probably use quoting, as then only 7.20.1 needs to change. | |  | |
| GB-029 | |  | | 6.2.6.1 | | 3 | | ed | | This note would seem to be superfluous, as the readership of the Standard should be expected to know what “binary” means! If the definition is felt to be important then move it to paragraph 3. | | Remove this note. | |  | |
| GB-030 | |  | | 6.2.6.2 | | 5 | | ed | | The statement in Note 1 that “this cannot occur with unsigned types” is too general; unsigned types have exceptional conditions (resulting in undefined behavior), such as division by zero or shifting by a too-large or negative amount, and those conditions could result in a non-value representation being stored as their form of undefined behavior. | | Remove “, and this cannot occur with unsigned types”. (Alternatively, a more detailed discussion could be added of which operations on unsigned types can or cannot have exceptional conditions.) | |  | |
| DE-031 | |  | | 6.2.7 | | 3 | | te | | The rules for composite types in C are not fully complete which is a known issue affecting also previous versions of C standard (cf. C90 DR#13). The most relevant issue in the past was that the composite type of an enumerated type and its compatible integer type is not specified. With the new changes to compatibility of tagged types and the changes to enumerated types in C2X the general issue becomes more problematic and the rules should be fully specified to avoid future implementation divergence. | | 6.2.7 Compatible Type and Composite Type  *3 A composite type can be constructed from two types that are compatible; it is a type that is compatible with both of the two types and satisfies the following conditions:*  ***– If both types are structure or union types the following rules apply:***   * ***If both are the same, the composite type is this type.*** * ***Otherwise, if both types are structure or union types, the composite type is determined recursively by forming the composite types of their members.*** * ***Otherwise, if both types are enumerated types, the composite type is a compatible enumerated type.*** * ***Otherwise, if one type is an enumerated type and the other is a compatible integer type, the composite type is the enumerated type.*** * *…*   ***These rules apply recursively to the types from which the two types are derived. When constructing the composite type for the same two types in different contexts, the composite type is the same. If the composite type is equivalent to one of the two original types, it is unspecified whether the composite type is the same type as an equivalent original type.\*\****  ***\*\*) The notion of "same type" affects redeclarations of typedef names and structure or union types.*** | | - | |
| GB-032 | |  | | 6.2.7, 6.7.2.3 | |  | | te | | Within a translation unit, it is problematic to allow union types with members declared in a different order to be compatible, because a braced initializer for a union initializes the first named member (and default initialization is also defined in terms of the first named member), and given such declarations, it's not clear which member should be considered first. The same issue of course applies when the union has no tag but is contained within a struct or union that does have a tag. | | In 6.2.7 paragraph 1, after “For two structures, corresponding members shall be declared in the same order.” insert “For two unions declared in the same translation unit, corresponding members shall be declared in the same order.”.  In 6.7.2.3 paragraphs 7 and 8, move the “union bar” declarations from Example 1 (valid) to Example 2 (invalid), with a comment added “only valid if the two declarations are in different translation units”. | |  | |
| GB-033 | |  | | 6.3.1.8 | | 3 | | ed | | The comment on the third multiplication example is only correct if int has less than 33 bits; if int has 33 bits or more, the \_BitInt(33) operand is converted to int. | | At the end of the comment on the third multiplication example, add “, provided int has width at most 32”. | |  | |
| US 9  -034 | |  | | 6.3.2.3 | | Para 3 | | te | | This makes nullptr a null pointer constant, as well as 0, and (void \*)0, but it seems to leave out (void \*)nullptr which seems like a minor oversight. | | Add (void \*)nullptr to the list of allowed forms of a null pointer constant. | |  | |
| US 10  -035 | |  | | 6.3.2.4 | | Para 1 | | te | | Introduces incompatible semantics with C++ regarding what can be converted to nullptr\_t type in the following example (rejected in C, accepted in C++):  void func(nullptr\_t); func(0); | | This example should be accepted, as in C++. | |  | |
| FR-036 | |  | | 6.4.1 | | p3 | | te | | Allow the bitprecise types \_BitInt(N) to be implemented as macros | | see accompanying document | |  | |
| US 11  -037 | |  | | 6.4.1 | | Para 3 | | te | | There is an unfortunate (existing) incompatibility between C and C++ regarding thread\_local that is exacerbated by renaming \_Thread\_local to thread\_local. The meaning of \_Thread\_local in C and thread\_local in C++ are subtly different in terms of behavior (this is an existing problem today with the thread\_local macro and not new to C2x). This change impacts code like:  extern thread\_local struct A a; A \*access() {  return &a; } | | We recommend rolling back just the change for thread\_local from this paper to give time for further collaboration between WG14 and WG21 to resolve the incompatibility without being under time constraints. | |  | |
| NEN/NL5-038 | | n/a | | 6.4.1; 6.4.4.5; 6.5.4; 6.5.9; 6.5.15, 7.21.2 | | n/a | | te | | The nullptr predefined keyword and nullptr\_t type was added, representing a separate way to access a null pointer constant. This may be redundant and unnecessary and the problems it addresses not suitably sufficient to justify keeping it. Users voiced concern over keeping it in the C Standard, and a few audited existing codebases and existing implementations to see if there was any need beyond just settling on an existing null pointer constant such as “(void\*)0”. Additional users found existing practice where “0” and “0L” were being used as the null pointer constant (e.g., in definitions of “NULL”) for embedded chips like those employed by U.S. vendor [REDACTED]. While some vendors responded positively to being encouraged to change from using “0” and “0L” to “(void\*)0”, others either did not respond or rejected outright the idea that they would change their NULL macro from 0 to “(void\*)0”. Some platforms use a special \_\_null even for their non-C++ platforms. Other vendors, such as [REDACTED], used “0”/”0L” explicitly unless a macro turned on to opt into a newer “(void\*)0” definition of the macro. It is noted these users and implementations were in the vast minority, even though they do definitively exist. | | Given this additional information, poll the C Standards Committee again if the nullptr changes from N3042 should remain or be removed completely from the C Standard. | |  | |
| GB-039 | |  | | 6.4.2.1 | | 1 | | te | | Consider allowing $ in identifiers again. | | See N3046 for some possible choices of wording. | |  | |
| US 12  -040 | |  | | 6.4.2.1 | | 1 | | te | | Adopt Wording Alternative #3 from N3046 <https://www.open-std.org/jtc1/sc22/WG14/www/docs/n3046.pdf> | | nondigit: one of \_ $ 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 | |  | |
| US 14  -041 | |  | | 6.4.2.1 | | 2 | | te | | Adopt Wording Alternative #3 from N3046 <https://www.open-std.org/jtc1/sc22/WG14/www/docs/n3046.pdf>    [Duplicate of US 11] | | An identifier is a sequence of nondigit characters (including the underscore \_, the dollar sign $, the lowercase and uppercase Latin letters, and other characters) and digits, which designates one or more entities as described in 6.2.1. It is implementation-defined if a dollar sign $ may be used as a nondigit character. Lowercase and uppercase letters are distinct. There is no specific limit on the maximum length of an identifier. | |  | |
| US 13  -042 | |  | | 6.4.2.1 | | 2 | | te | | Adopt Wording Alternative #2 from N3046 <https://www.open-std.org/jtc1/sc22/WG14/www/docs/n3046.pdf> | | An identifier is a sequence of nondigit characters (including the underscore \_, the lowercase and uppercase Latin letters, and other characters) and digits, which designates one or more entities as described in 6.2.1. The nondigit characters may also include a dollar sign $. Lowercase and uppercase letters are distinct. There is no specific limit on the maximum length of an identifier. | |  | |
| GB-043 | |  | | 6.4.2.1 | | 9 | | ed | | The references to “attribute token” here should be more specifically to the identifiers listed for “standard attribute”. | | Change “attribute token” to “standard attribute” (twice in this paragraph). | |  | |
| FR-044 | |  | | 6.4.2.1 | | p10 | | te | | The status of optional identifiers could be clarified | | see accompanying document | |  | |
| GB-045 | |  | | 6.4.4.1 | | 7 | | te | | A constant with the “wb” or “WB” suffix that exceeds BITINT\_MAXWIDTH should not be allowed to have an extended integer type. | | In the first sentence of this paragraph, change “If an integer constant” to “If an integer constant that does not have a suffix including wb or WB”. | |  | |
| GB-046 | |  | | 6.4.4.3, 6.7.2.3, 6.7.12.2, 7.20.1 | |  | | ed | | For consistency, the phrase “enumeration type” should be changed to “enumerated type” throughout (the latter is the defined term in 6.2.5 paragraph 21). | | Change “enumeration type” to “enumerated type” throughout the document. Do not change uses of the word “enumeration” outside that phrase. | |  | |
| GB-047 | |  | | 6.4.4.4 | | 13, 14 | | ed | | “an unsigned integer types” should be “an unsigned integer type”. | | Change “an unsigned integer types” to “an unsigned integer type” (in both paragraph 13 and paragraph 14). | |  | |
| US 15-048 | |  | | 6.4.6 | | Footnote 89 | | ed | | Change the text to refer to both digraphs and trigraphs.  This is not necessary and might be clearer, or might not depending on the association with the old semantics. | | Add the words ‘and “trigraphs”’ to the end of the footnote sentence. | |  | |
| NEN/NL1-049 | | n/a | | 6.4.6 | | Paragraph 3 | | te | | Tripgraph removal is a good and necessary change that should happen. However, in removing trigraphs we have violated some of the few norms we have about working with and adjusting the C Standard. Namely, there was no deprecation period and no consideration of alternatives for implementations and users with demonstrated need of such.  We think trigraphs should be removed! However, doing so without a deprecation period and/or an alternative is not helpful to those users and that userbase. | | One of two potential options should be polled by the Committee:   * revert N2940 “Remove Trigraphs??!” and mark Trigraphs as a deprecated feature; OR, * transition behavior of trigraphs into digraphs, removing problematic string-defying and comment-defying trigraph sequences and interchanging them with helpful digraph-based behavior, especially for the preprocessor “#” symbol. | |  | |
| US 18  -050 | |  | | 6.4.6 | | Paragraph 3 | | te | | Continue to make the remaining former trigraphs available with digraph semantics.  This minimizes the impact on existing user code.  No real-world user code is known to depend on the trigraph-specific semantics; all reported uses are as punctuators. | | Assuming comments 1 and 2 are included:  Revert paragraph 3 back to its original form, then add four new lines before “except for their spelling” (this is clearer than a single long list):  and the nine tokens  ??= ??/ ??’ ??( ??) ??! ??< ??> ??-  behave, respectively, the same as the nine tokens  # \ ^ [ ] | { } ~ | |  | |
| US 17  -051 | |  | | 6.4.6 | | Paragraph 3 | | te | | Continue to make the former trigraphs for [ and ] available, but with “digraph” semantics.  These are the most commonly-used trigraphs in existing user code. | | Assuming comment 1 is included;  Change both occurrences of “the seven tokens” to “the nine tokens”; add to the end of the first list of punctuators, “??(” and “??)” (unquoted, three characters); add to the end of the second list of punctuators “[” and “]” (unquoted, one character) | |  | |
| US 16  -052 | |  | | 6.4.6 | | Paragraph 3 | | te | | Continue to make the former trigraph for # available, but with “digraph” semantics.  This is necessary to allow basic minimal C23 portability of older projects which use a #pragma line to switch the compiler into ASCII mode. | | Change both occurrences of “the six tokens” to “the seven tokens”; add to the end of the first list of punctuators, “??=” (unquoted, three characters); add to the end of the second list of punctuators “#” (unquoted, one character) | |  | |
| GB-053 | |  | | 6.4.7 | | 3 | | ed | | The wording here about contexts for header name preprocessing tokens needs to include the same contexts as in 6.4 paragraph 4. | | Change “Header name preprocessing tokens are recognized only within #include preprocessing directives and in implementation-defined locations within #pragma directives.” to “Header name preprocessing tokens are recognized only within #include and #embed preprocessing directives, in \_\_has\_include and \_\_has\_embed expressions, as well as in implementation-defined locations within #pragma directives.” (the same wording as in 6.4 paragraph 4, and formatted the same way). | |  | |
| GB-054 | |  | | 6.5 | | footnote 95 | | ed | | The list of cases where operands may come from a later subclause doesn’t allow for \_Generic. | | After “grouping parentheses () (6.5.1),”, insert “generic selection parentheses () (6.5.1.1),” (with the first pair of parentheses in a fixed-width font). | |  | |
| GB-055 | |  | | 6.5.1 | | Constraints | | ed | | The Constraints paragraph does not have a paragraph number. | | Number this paragraph 2, adjusting numbers for paragraphs below accordingly. | |  | |
| GB-056 | |  | | 6.5.15 | | 3, 6 | | te | | Given the tag compatibility changes, conditional expressions should be allowed between structures or unions that have compatible (not necessarily the same) type. | | In paragraph 3, second bullet point, change “the same” to “compatible”. In paragraph 6, change “If both the operands have structure or union type, the result has that type.” to “If both the operands have structure or union type, the result has the type of one operand.”. | |  | |
| GB-057 | |  | | 6.5.15 | | footnote 122 | | ed | | “that is not a null pointer constant” is not necessary for this footnote to apply, and “or” should be “and”. | | Remove “that is not a null pointer constant”. Change “or does not have type” to “and does not have type”. | |  | |
| US 22-058 | |  | | 6.5.15 | | Para 3 | | te | | Introduces incompatible semantics with C++ regarding behavior of the following examples (rejected in C, accepted in C++): nullptr\_t val; (void)(1 ? val : 0); (void)(1 ? nullptr : 0); | | These examples should be accepted, as in C++. | |  | |
| GB-059 | |  | | 6.5.16.1 | | 3 | | ed | | The editorial change to say “the overlap shall exactly match” seems less clear than the previous wording. | | Change “then the overlap shall exactly match and the two objects shall have” to “then the two objects shall occupy exactly the same storage and shall have”. | |  | |
| US 25-060 | |  | | 6.5.16.1 | | Footnote 126 | | ed | | This footnote can't be implemented because it would require the implementation to inspect the value of an object as an assignment constraint at compile time. Consider:  nullptr\_t lhs\_val;  nullptr\_t rhs\_val = (nullptr\_t)(void \*)1; // UB  lhs\_val = rhs\_val; // Expects a constraint violation here per the footnote but nothing normative | | I think the footnote should read that it’s undefined behavior rather than a constraint violation. | |  | |
| US 24-061 | |  | | 6.5.16.1 | | Para 1 | | te | | Introduces a different kind of incompatible semantics with C++ regarding the following example (rejected in C, accepted in C++): nullptr\_tval; val = 0; | | This example should be accepted, as in C++. | |  | |
| US 23-062 | |  | | 6.5.16.1 | | Para 1 | | te | | Introduces incompatible semantics with C++ regarding the behavior of the following examples (accepted in C, rejected in C++): nullptr\_t val; bool b1 = val; bool b2 = nullptr; | | These examples should be rejected, as in C++. | |  | |
| GB-063 | |  | | 6.5.2.1, 6.5.3.3, 6.5.6, 7.17.7.5 | |  | | ed | | In various places, including at least 6.5.2.1 paragraph 2, 6.5.3.3 paragraphs 1, 2 and 3, 6.5.6 paragraphs 6 and 7, and 7.17.7.5 paragraph 1, the ‘+’ and ‘-’ symbols are used for C operators but the font used is not the fixed-width font generally used for C code in the standard. Some such changes were made as part of commit 2333c3344c6955f9badc8a5b433c9e067d6fd596 in the C standard git repository, to work around other formatting issues with spacing around those operators. | | Fix those spacing issues in some other way, if not already fixed, and then restore all uses of ‘-’ and ‘+’ as C operators (as opposed to as mathematical operations) to a fixed-width font. (Other changes from the referenced commit should not be reverted.) | |  | |
| CA3-064 | |  | | 6.5.2.2 | | 2 | | Te | | The removal of function declarations without a prototype without a viable replacement (var-args can have different ABIs on some systems) makes it impossible to declare functions that take a variable number of arguments but are not C var-arg functions. | | Either return the specification for called functions without a prototype or provide an alternate means of declaring such a function. | |  | |
| GB-065 | |  | | 6.5.2.2, 6.5.4 | |  | | te | | A series of changes have had the general effect of eliminating ways in which an rvalue might be considered to have a qualified type (see DR#423 / N1863, DR#481, N2726). However, those changes left two places where a literal reading would give an rvalue an atomic type: calls to functions with an atomic return type, and casts to an atomic type. This seems anomalous and contrary to the general intent, given that such types cannot occur with the result of lvalue-to-rvalue conversion, which returns an unqualified, non-atomic type, and given that it is explicitly said for both atomic and qualified types that those properties are only meaningful for lvalues (6.7.2.4, 6.7.3). | | In 6.5.2.2 paragraph 5, change “the function call expression has the same type as that object type” to “the type of the function call expression is the non-atomic version of that object type”, with a footnote saying “A function type cannot have a return type that is a qualified type (6.7.6.3).”. Alternatively, if it’s desired to avoid atomic return types on functions at all, change 6.7.6.3 paragraph 4 by changing “unqualified” to “unqualified, non-atomic”.  In 6.5.4 paragraph 5, change “unqualified” to “unqualified, non-atomic”. | |  | |
| US 19  -066 | |  | | 6.5.2.5 | | Para 4 | | te | | This specifies whether a compound literal expression is associated with block scope or file scope, but it fails to make clear what happens with a declaration like the following:  void func(int array[(register int){12}]);  Note that this is a function declaration, not a definition, so there is no block scope associated with it. This matters for conformance to 6.9p2; if the compound literal is declared at file scope, it should be diagnosed (because of the semantic rewriting to a declaration done in 6.5.2.5p4). | | Please clarify what was intended, but we believe this should be rejected as a constraint violation due to use of the register keyword at file scope. | |  | |
| US 20  -067 | |  | | 6.5.2.5 | | Paras 4 & 5 | | te | | These do not match what we think the intent is for code like:  (static int){12}  We think the intent is that this expression generates one notional static variable that is referenced each time the compound literal is evaluated. However, p4 says that it is rewritten to a form:  SC typeof(T) ID = { IL };  "where ID is an identifier that is unique for the whole program", while p5 says "A compound literal provides an unnamed object whose value, type, storage duration and other properties are as if given by the definition syntax in the constraints". The effect of this is that each time the compound literal is evaluated, you get a unique ID for the object and thus you get different static variables on each evaluation. | | Clarify whether each evaluation of a compound literal using the static storage class specifier creates a unique object or not. | |  | |
| US 21  -068 | |  | | 6.5.4 | | Para 4 | | te | | Introduces incompatible semantics with C++ regarding the following example (rejected in C, accepted in C++):  (nullptr\_t)nullptr;  because nullptr\_t is not void, bool, or a pointer type (so it cannot be cast to itself). Note, this might be editorial because the last sentence of the para says “No type other than nullptr\_t shall be converted to nullptr\_t” but it’s unclear because of the use of “only” in the preceding sentence. | | Change the penultimate sentence to: The type nullptr\_t shall not be converted to any type other than void, bool, nullptr\_t, or a pointer type. | |  | |
| DE-069 | |  | | 6.5.4, 6.7.8, 6.8. | |  | | te | | When size expressions of arrays have to be evaluated is not clear, especially when used with typeof.  **Examples:**  (char(\*)[f()])0;  typeof(f()) x;  typedef typeof(f()) t; | | 6.5.4. Cast Operators, Semantics  ***Size expressions and typeof operators contained in a type name used with a cast operator are evaluated whenever the cast expression is evaluated.***  6.7.8. Type Definitions  *Any array size expressions associated with variable length array declarators* ***and typeof operators*** *are evaluated each time the declaration of the typedef name is reached in the order of execution*  6.8. Statement and Blocks  *… ~~and the variable length array declarators~~****and any size expressions and typeof operators in declarations*** *of ordinary identifiers with block scope are evaluated.* | |  | |
| GB-070 | |  | | 6.5.8 | | 7 | | ed | | A “.” appears both before and after the reference to footnote number 119. | | Remove the second “.”. | |  | |
| GB-071 | |  | | 6.5.9 | | 2, 6 | | te | | Almost all places that allow operands of type nullptr\_t do so independent of whether the particular operand is a null pointer constant or another expression of that type. However, the rules for equality operators only allow comparison of a pointer that is not a null pointer constant with a nullptr\_t value that is a null pointer constant, not with any other nullptr\_t value. Since it seems C++ implementations allow such comparisons, disallowing them in C might not be intentional. | | If it’s not desired to disallow such comparisons after all, in the last bullet point in paragraph 2, change “is a null pointer constant” to “is a null pointer constant or has type nullptr\_t”. In paragraph 6, change “the other is a null pointer constant, the null pointer constant is converted” to “the other is a null pointer constant or has type nullptr\_t, the null pointer constant or operand of type nullptr\_t is converted”. | |  | |
| GB-072 | |  | | 6.5.9 | | 4 | | ed | | There is no “.” at the end of the sentence to which footnote 120 is attached. | | Add a “.” before the footnote number. | |  | |
| FR-073 | |  | | 6.5.9 | | p2 and p6 | | te | | There is a missing case for the comparison of pointers with type nullptr\_t | | see accompanying document | |  | |
| GB-074 | |  | | 6.6 | | 6 | | ed | | Paragraph 15 says that the “.” member access operator may form a compound literal constant, but paragraph 6 doesn’t mention that possibility (unlike paragraph 7, which mentions it for named constants). | | At the end of the first sentence in paragraph 6, insert “, as is a postfix expression that applies the . member access operator to a compound literal constant of structure or union type, even recursively”. | |  | |
| US 26  -075 | |  | | 6.6 | | Para 17 | | te | | N3018 (the constexpr specifier for object definitions) introduces the ability to have constant expressions in more situations, but it does not update the rules for constant expression evaluation of floats. This paragraph is not possible to implement in the case of floating-point types. C++ avoids this with a recommended practice http://eel.is/c++draft/expr.const#13 because they recognized the implementation impossibilities. Consider code like:  constexpr float f = 1.0f;  constexpr float g = 3.0f;  fesetround(FE\_TOWARDSZERO);  constexpr float h = f / g;  Short of performing a full analysis of the control flow graph for the translation unit, there's no way to meet this semantic requirement. | | Consider using a recommended practice for the floating-point semantic requirements instead of mandating them. | |  | |
| GB-076 | |  | | 6.7 | | 5 | | te | | There are several places in the standard that impose restrictions on what can occur within certain kinds of declarations, or require certain kinds of declarations to contain certain kinds of constructs or certain kinds of constructs to appear in certain declaration contexts, without being fully clear about what syntactic positions (within the tokens matching the “declaration” syntax specifier) count for the purposes of those rules. To the extent that the intent of these have been clarified in DRs, or are clarified in examples, it is not clear that the clarification actually follows from the normative text, especially since the intent seems to be different in different cases. See attached document c2x-declaration-context.pdf for more details of this issue. | |  | |  | |
| GB-077 | |  | | 6.7 | | 6 | | ed | | The bullet point on definitions, “an enumeration constant, is the only declaration of the identifier” (parentheses around “only” removed since the previous revision of the C standard) doesn’t cover the case of redeclaration of enumeration constants, which is now permitted; one declaration should still be considered a definition in that case. | | Change “only” to “first (or only)” for enumeration constants (matching the wording for typedef names). | |  | |
| GB-078 | |  | | 6.7.1 | | 14 | | ed | | The example doesn’t make clear why the initializer for unstring might be invalid. | | At the end of the sentence about unstring, add “, because if char has eight bits and the same range of values as signed char, the initializer "\xFF" has type “array of char”, and its first element has value -1, which is not representable in unsigned char’. | |  | |
| GB-079 | |  | | 6.7.1 | | 5 | | te | | The reference to implicit initializer values in paragraph 5 runs into the issue that the definition of default initialization in 6.7.10 only says what the initializer value is, not what kind of constant expression it is. For example, default initialization for a pointer type is said to be with a null pointer; it is not specified whether it is a null pointer constant. | | If the removal of the reference to implicit initializers from the previous comment is accepted, this issue goes away. Otherwise, 6.7.10 would need amending to say what kind of constant default initializers are. In any case, add a new Example:  constexpr int \*p = {}; // Default initialization with a null pointer | |  | |
| GB-080 | |  | | 6.7.1 | | 5 | | te | | Suppose a structure or union object is, as the whole or part of a constexpr initializer, initialized with a single expression (a named or compound literal constant) of that type (see comment below on 6.7.10 regarding the details of when exactly this is permitted). How exactly should the constraints that constexpr pointer initializers be null pointer constant be applied in this case, if the structure or union contains such a member (and, in the case of the union, the pointer is the active member or part of the active member)? Should it be a constraint violation because a hypothetical expression accessing the pointer member of the constant structure or union would not itself be a null pointer constant? Or should it be permitted because no such hypothetical expression is actually formed in the initialization process and the initializer actually written for the structure or union is a structure or union constant? | | Supposing such an initialization should be permitted, replace “the implicit or explicit initializer value” by “any explicit initializer value”. Add a new Example:  struct s { void \*p; };  constexpr struct s A = { nullptr };  constexpr struct s B = A;  /\* Although the expression A.p is not a null pointer constant, only a null pointer, the only explicit initializer in the initialization of B is A, not A.p, so no constraint is violated by that initialization. \*/  If (see comment below on 6.7.10) such initializations are only accepted with automatic storage duration (if the proposed change for that comment is rejected), this example will need adjusting accordingly. | |  | |
| GB-081 | |  | | 6.7.1 | | 5 | | te | | The requirement on constexpr initializers that “The value of any constant expressions or of any character in a string literal of the initializer shall be exactly representable in the corresponding target type; no change of value shall be applied” is unclear in many cases, especially concerning floating point, regarding what exactly counts as a change of value when the type changes; the footnote and examples are only of limited value for clarifying this, and to the extent that the do clarify it, it is not clear that the results follow from the normative text. See attached document c2x-constexpr-init.pdf for more details of this issue. | |  | |  | |
| GB-082 | |  | | 6.7.10 | | 11, 12, 20, 22 | | ed | | The changes to define “default initialization” left awkward wording and duplication and were not consistently implemented. | | Change paragraph 11, before the bullet points, to “if an object that has automatic storage duration is not initialized explicitly, its representation is indeterminate. If an object that has static or thread storage duration is not initialized explicitly, or any object is initialized with an empty initializer, then it is subject to *default initialization*, which initializes an object as follows:”.  Change “;” to “.” at the end of the last bullet point in paragraph 11.  In paragraph 12, change “If the initializer is the empty initializer, the initial value is the same as the initialization of a static storage duration object. Otherwise” to “If the initializer is not the empty initializer”.  In paragraph 20, change “shall be initialized implicitly the same as objects that have static storage duration” to “are subject to default initialization”.  In paragraph 22, change “shall be initialized implicitly the same as objects that have static storage duration” to “is subject to default initialization”. | |  | |
| GB-083 | |  | | 6.7.10 | | 14, 17 | | te | | When should it be possible to initialize an object of structure or union type with a named constant or compound literal constant of such type? The current wording seems to indicate that it is only possible at automatic storage duration – but when such initialization occurs, it is permitted even when the object being initialized is constexpr. Since the normal rule is that the requirements on constexpr initializers are stricter than those on static storage duration initializers, it is an anomaly to allow named and compound literal constant of structure or union type in constexpr initializers of automatic storage duration but not in any initializers of static storage duration. | | Assuming such initializers are intended to be allowed at static storage duration, in paragraph 14, remove “that has automatic storage duration”. Add a footnote to the end of the paragraph: “If the object being initialized does not have automatic storage duration, this case violates a constraint unless the expression is a named constant or compound literal constant (6.6).”. Also remove “that has automatic storage duration” in J.2 item 81. | |  | |
| CA1-084 | |  | | 6.7.12 | | 2 | | Te | | Standard attribute syntax errors should be diagnosed as was the intent in C++. | | Add a paragraph after 2 in the constraints section to say that an attribute must be added to something that it can apply to, and that the syntax rules for any argument to the attribute must be as specified for the attribute. | |  | |
| US 35  -085 | |  | | 6.7.12 and 6.7.12.1 | | Para 2, Para 3 | | te | | At the C++23 NB comment resolution meeting in Nov 2022, WG21 clarified their model of what it means to ignore an attribute in a way that is incompatible with how WG14 has it specified for C23. Specifically, ignored attributes are not actually ignored in the usual sense of the English phrase, but are instead required to be syntax and semantically checked if the attribute is a standard attribute. e.g., if an application ignores an attribute, it is still required to diagnose incorrect appertainment, presence or absence of an argument list, syntax or semantic errors in the argument list, etc. This was done in the name of improved portability.  Note, there is at least one C++ implementation that has said they will not be implementing that change. | | If being able to syntactically ignore attributes is not important to WG14, we can change our model to match the new C++ model. If this property is critical to WG14, we can diverge from C++ with our model – however, that is user-hostile given how frequently attributes show up in header files shared between C and C++. A third option is for impacted companies or national bodies to vote NO on the ISO/IEC 14882:2023 DIS ballot to reopen the discussion with WG21. | |  | |
| GB-086 | |  | | 6.7.12, 6.7.12.3, 6.7.12.4, 7.17.2.1, 7.17.3, 7.28.3.3, 7.28.3.5, K.3.9.3.1.1 | |  | | ed | | The C standard normally uses the word “object” instead of “variable” as a noun in normative text, with noun uses of “variable” being limited to the phrases “system variable” and “condition variable”. However, some normative wording, mainly for attributes, atomics and threads, uses “variable” as a noun when “object” would be more consistent with the usual wording practice. | | Change “variable” to “object” when used as a noun in normative text, outside of the phrases referred to. Those phrases using the word should not be changed, use as an adjective should not be changed, and non-normative uses should be considered individually as the informal term “variable” may sometimes be appropriate there. | |  | |
| US 36  -087 | |  | | 6.7.12.\* | |  | | te | | There is an ambiguity between 6.10.1p9 (\_\_has\_c\_attribute returns 0 for an ignored attribute) and 6.7.12.\* (each attribute specifies that \_\_has\_c\_attribute \*shall\* return a nonzero value). | | The attributes in 6.7.12.\* should be amended to say "...when given <attribute name> as the pp-tokens operand<ins> if the implementation supports the attribute</ins>." | |  | |
| GB-088 | |  | | 6.7.12.2, 6.7.12.6 | |  | | ed | | The wording “shall be applied to the identifier in a function declaration”, taken literally, would seem to require the syntax “identifier attribute-specifier-sequence[opt]” to be used, rather than other syntax to apply an attribute to a function. Since the examples given show the attribute at the start of the declaration instead, that clearly isn’t intended. Other attributes refer to applying an attribute to a function, so avoiding this issue. | | In 6.7.12.2 (nodiscard) paragraph 1, and 6.7.12.6 (noreturn) paragraph 2, change “the identifier in a function declaration” to “a function”. | |  | |
| GB-089 | |  | | 6.7.12.5 | | 1 | | te | | It is unclear whether “next block item that would be encountered” refers to next lexically, or in execution order (the latter being problematic for a Constraint, since those are translation-time properties). A similar issue was previously raised for C++ - see <https://www.open-std.org/jtc1/sc22/wg21/docs/cwg_defects.html#2406> – but it appears the resolution for C++ was not applied for C. | | Add the same wording and changes to the example as for C++. | |  | |
| GB-090 | |  | | 6.7.12.7 | | 11 | | ed | | “execution wide” should have a hyphen. | | Change “execution wide” to “execution-wide”. | |  | |
| GB-091 | |  | | 6.7.12.7 | | 2 | | ed | | “referred” seems unidiomatic here; it could be made “referenced”, but the word doesn’t actually seem necessary at all. | | Remove “referred”. | |  | |
| GB-092 | |  | | 6.7.12.7 | | 3 | | ed | | “a such annotated function” seems unidiomatic. | | Change “a such annotated function” to “such a function”. | |  | |
| GB-093 | |  | | 6.7.12.7 | | 4 | | ed | | “account as” used here seems unidiomatic, as does “allow to query” | | Change “account as objects” to “are considered as objects, for the purposes of these attributes”. Change “allow to query” to “access”. Change “account as lvalue conversions” to “are considered as lvalue conversions, for the purposes of these attributes”. Change “account as store operations” to “are considered as store operations, for the purposes of these attributes”. | |  | |
| GB-094 | |  | | 6.7.12.7 | | footnote 187 | | ed | | “if” here seems unidiomatic. | | Change “independently if” to “independently of whether”. Change “or if” to “or whether”. | |  | |
| GB-095 | |  | | 6.7.12.7.2 | | 3 | | ed | | “the such typed function” seems unidiomatic. | | Change “the such typed function” to “such a function”. | |  | |
| GB-096 | |  | | 6.7.12.7.2 | | 4 | | ed | | “property if” seems unidiomatic. | | Change “property if” to “property of whether”. | |  | |
| GB-097 | |  | | 6.7.2.1 | | 24 | | ed | | “sizeof(ouble)” should be “sizeof(double)”. | | Change “sizeof(ouble)” to “sizeof(double)”. | |  | |
| GB-098 | |  | | 6.7.2.2 | | 12 | | te | | Because of the special semantics of conversions to bool, it seems a bad idea to allow bool as the type with which an enumeration without fixed underlying type is compatible. | | In the wording as modified by the previous comment, change “that is not a bit-precise integer type” to “that is not bool or a bit-precise integer type”. | |  | |
| GB-099 | |  | | 6.7.2.2 | | 12 | | ed | | The first sentence of this paragraph should end “type” not “types”, and could be read such that “a signed integer type” and “an unsigned integer type that is not a bit-precise integer type[s]” are two separate entries in the list (so allowing a signed bit-precise integer type as the type with which the enum is compatible, which clearly isn’t intended). | | Change “char, a signed integer type, or an unsigned integer type that is not a bit-precise integer types” to “char or a signed or unsigned integer type that is not a bit-precise integer type”. | |  | |
| GB-100 | |  | | 6.7.2.2 | | 15 | | te | | This paragraph talks about behavior of values of an enumerated type after lvalue conversion, with a footnote about the case of bool. However, most of the special semantics of bool relate to lvalues of that type rather than use of rvalues in arithmetic, and neither the normative text nor the footnote explicitly say that the semantics of conversion **to** the enumerated type are the same as those of conversion to the compatible type. | | Add a sentence at the end of the paragraph, “Conversion to the enumerated type has the same semantics as conversion to the underlying type.”. Move footnote 153 to be after that sentence. In footnote 153, change “the same as bool” to “the same as bool, conversion to the enumerated type behaves the same as conversion to bool (6.3.1.2)”. | |  | |
| GB-101 | |  | | 6.7.2.2 | | 19 | | ed | | “initialized to one” should be “initialized to 1” for consistency with the previous paragraph. | | Change “initialized to one” to “initialized to 1” (with “1” in fixed-width type). | |  | |
| GB-102 | |  | | 6.7.2.2 | | 3 | | ed | | It appears to be intended, for consistency with C++, that if an enum with fixed underlying type bool has an enumeration constant with value 1, and the following enumeration constant does not have a defining constant expression, a constraint is violated because 1 + 1 is not representable in bool. However, it is clear that the conversion of 1 + 1 to bool does not meet the definition of wraparound, and it might not meet the definition of overflow either, so different wording may be needed to ensure this case is a constraint violation. | | Change “The definition of an enumeration constant without a defining constant expression shall neither overflow nor wraparound the fixed underlying type by adding 1 to the previous enumeration constant.” to “If the value of an enumeration constant without a defining constant expression for an enumeration with fixed underlying type is obtained by adding 1 to the previous enumeration constant, the value of that previous enumeration constant shall not be the maximum value of the underlying type.”. | |  | |
| GB-103 | |  | | 6.7.2.2 | | 5 | | te | | The N2904 version of the proposal for enums with fixed underlying types included a sentence “The underlying type of the enumeration is the unqualified, non-atomic version of the type specified by the type specifiers in the specifier qualifier list.”. That sentence disappeared in the N2963 version and so is not in the current draft, but it appears it is still the intent that the unqualified, non-atomic version of the type be used, and there is no other normative wording to replace that sentence to achieve that effect. | | At the end of paragraph 5, add “The underlying type of the enumeration is the unqualified, non-atomic version of the type specified by the type specifiers in the specifier qualifier list.”. | |  | |
| GB-104 | |  | | 6.7.2.2 | | 5 | | ed | | The N2904 version of the proposal for enums with fixed underlying types included a sentence “No alignment specifiers shall appear in the specifier qualifier list.”, which disappeared in the N2963 version. There is no actual need for such a normative sentence, since it follows from requirements elsewhere in the standard, but a footnote pointing this out would be helpful. | | At the end of paragraph 5, add a footnote “The specifier qualifier list is not a context listed in 6.7.5 as permitted for alignment specifiers, so the presence of an alignment specifier in the list violates a constraint.”. | |  | |
| GB-105 | |  | | 6.7.2.2 | | 7 | | te | | There appears to be no Constraint that would explicitly disallow declaring an enum with a fixed underlying type and then defining it without one.  enum e : int;  enum e { A }; | | At the end of the Constraints in 6.7.2.2, add a new paragraph (after paragraph 7): “An enumeration with a fixed underlying type shall be defined with an enum type specifier. No enum specifier for an enumeration without a fixed underlying type shall include an enum type specifier.". | |  | |
| GB-106 | |  | | 6.7.2.3 | | 10, 11, footnote 156 | | te | | Some of the changes related to tags in accepted paper N3030 have not been applied to the working draft. | | Apply the changes to paragraph 10: insert “enum-type-specifier[opt]” in both enum syntax alternatives, after identifier[opt].  Apply the changes to paragraph 11: add “**enum** identifier enum-type-specifier **;”** as a syntax alternative and change “structure or union type” to “structure, union, or enumerated type”.  Apply the changes to footnote 156 as shown in the paper. | |  | |
| FR-107 | |  | | 6.7.2.3 | | p1 | | te | | There is a new ambiguity for the initialization of unions if they are redeclared | | see accompanying document | |  | |
| GB-108 | |  | | 6.7.2.5 | | 4, 5 | | ed | | The typeof specifiers should be referred to as specifying a type, not a “type name” (the latter is a different syntax production for certain sequences of tokens). | | In paragraph 4, remove “the type name representing”. Change “produce the type name” to “designate the same type as the type name”. In paragraph 5, change “non-atomic unqualified type name” to “non-atomic unqualified version of the type”. | |  | |
| US 28  -109 | |  | | 6.7.2.5 | | Example 3 | | ed | | This example uses smart quotes rather than straight quotes around the character constants. | | Use straight quotes instead. | |  | |
| US 27  -110 | |  | | 6.7.2.5 | | Examples 1 & 2 | | ed | | 5.1.2.2.1p1 says that main "shall be defined with a return type of int", so use of typeof and typeof\_unqual in these examples are technically violating a constraint. | | Use `int` as the return type for `main()` and pick different examples for those two cases. | |  | |
| GB-111 | |  | | 6.7.2.5 | | footnote 159 | | ed | | “type-name” should not be in a fixed-width font. | | Change “type-name” in the footnote out of a fixed-width font. | |  | |
| GB-112 | |  | | 6.7.3 | | 9 | | ed | | The statement about removing restrict not changing behavior of a conforming program is not strictly accurate for normative text, since \_Generic can be used in a way that distinguishes between e.g. pointers to restrict-qualified and non-restrict-qualified types. | | At the end of paragraph 9, insert “, unless \_Generic is used to distinguish whether or not a type has that qualifier”. | |  | |
| US 29  -113 | |  | | 6.7.3.1 | | example box | | ed | | Inconsistent use of space in front of operator parentheses. | | Remove space between “sizeof” and “(float)”. | |  | |
| CA4-114 | |  | | 6.7.6 | | 1 | | Te | | The removal of “K&R” declarations made classes of interfaces impossible and breaks a lot of existing code. | | Either return K&R declarations or allow forward declaration of parameters in some manner. | |  | |
| US 30  -115 | |  | | 6.7.6 | | Para 2 | | ed | | “… and asserts that when an operand of the same form as the declarator appears in an expression, it designates a function or object with the scope, storage duration, and type indicated by the declaration specifiers.” Consider:  static int \*g;  void f() {  int x;  g=&x;   // ...  }  How does the object `g` designate a scope indicated by the declaration specifiers? Declaration specifiers don’t indicate a scope – the lexical location of the declarator indicates the scope.  Also, the declarator is \*g and not g; \*g does not have static storage duration, g does. | | I think we can find a more clear way to specify that a declarator declares one identifier and that identifier has properties from the declaration specifiers associated with it. | |  | |
| DE-116 | |  | | 6.7.6.2 | | Example 2 | | ed | | The term “unspecified” is used incorrectly. | | *… array of* ~~unspecified~~ ***unknown*** *size ...* | |  | |
| DE-117 | |  | | 6.7.6.2, 6.5.3.4, 6.5.6 | |  | | te | | The size expression can appear inside a subexpression which is not evaluated according to usual rules. I propose to extend the use of the existing notion of "unspecified size" to arrays with unevaluated size expressions. Existing language rules for evaluation of expressions and for composite types would then give useful behavior for common cases. In other cases, the resulting type would be an array of unspecified size, which then should explicitely be UB in sizeof or pointer arithmetic.  **Examples:**  if (0) { (char(\*)[f()])0; }  0 ? (char(\*)[f()])0 : 0;  1 ? (char(\*)[f()])0 : (char(\*)[g()])0;  int a[n]; sizeof(\*(0 ? &a : 0));  sizeof(\*(0 ? (char(\*)[f()])0 : 0)); | | 6.7.6.2 Array Declarators  5 … *~~f it occurs in a declaration at function prototype scope,~~* ***if it is not evaluated,*** *it is treated as if it were replaced by \*;*  … ***A size expression in a declaration at function prototype scope is not evaluated.***  6.5.3.4. sizeof  *If the type of the operand is a variable length array type, the operand is evaluated.* ***If the variable length array type has unspecified size, the behavior is undefined.*** *Otherwise, the operand is not evaluated and the result is an integer constant.*  6.5.6 Additive operators, Semantics  ***If one of the operands is a pointer to an array of unspecified size, the behavior is undefined.***  6.5.15 Conditional operator, Constraints  **5 If either of the second or third operands is a null pointer constant or has type nullptr\_t, the other operand shall not be a variably modified type that has an unspecified size if the operand is not evaluated** | |  | |
| US 31  -118 | |  | | 6.7.6.3 | | 13 | | te | | Subclause 6.7.6.3, paragraph 4 says "the type specified for ident is "derived-declarator-type-list function returning the unqualified version of T". Qualified return type is a misleading language feature. | | Add the following paragraph after paragraph 12:  A return type declared to have a qualified type is an obsolescent feature. | |  | |
| GB-119 | |  | | 6.7.6.3 | | footnote 174 | | ed | | This footnote, talking specifically about a function definition, was relevant when empty parentheses in a function definition provided a prototype but those in other declarations didn’t. Since the working draft was changed to remove unprototyped functions, this footnote seems no longer relevant. | | Remove footnote 174. | |  | |
| GB-120 | |  | | 6.7.7 | | footnote 175 | | ed | | The reference to “no parameter specification” predates the removal of unprototyped functions. | | Change “no parameter specification” to “no parameters”. | |  | |
| US 32  -121 | |  | | 6.7.9 | |  | | ge | | As we've been working on implementing this functionality in Clang, we're finding that the specification differences between C and C++ are a significant source of consternation for us. In C++, auto is a type specifier. In C, auto is not a type, it's the \*absence\* of a type and the use of a storage class specifier. | | Please do not resurrect implicit int with different semantics, but define this as a type specifier. Logically, a deduced type is a type and not a class of storage. | |  | |
| US 34  -122 | |  | | 6.7.9 | | Para 2 | | te | | Introduces incompatible semantics with C++ regarding the following example (undefined behavior in C, accepted in C++):  int i;  auto good = &i;  auto \*bad = &i; // Cannot specify the pointer  This style is often a coding standard requirement for code bases in C++ due to the improved code readability: e.g., https://llvm.org/docs/CodingStandards.html#beware-unnecessary-copies-with-auto. | | We prefer that it be required to support (optional) pointer and array declarators as part of a deduced type. | |  | |
| US 33  -123 | |  | | 6.7.9 | | Para 2 | | te | | Adds a constraint on programs under a "Description" heading which makes it a bit less clear as to how to interpret the "shall" clauses used. For example, is this code UB or is it simply not possible to write:  auto a = { 1, 2 };  If it's UB, an implementation could elect to deduce `a` as `int[3]` or `int \*` and I don't think we want to allow an extension into that space (FWIW, in C++ that would deduce to a std::initializer\_list<int>). | | Clarify the intent by moving the specification either to a Constraints or Semantics section, or remove use of the word “shall | |  | |
| GB-124 | |  | | 6.8.4.2 | | 5 | | ed | | One reference to “default label” does not have “default” marked up in a fixed-width font as a keyword. | | In “following the default label”, put “default” in a fixed-width font. | |  | |
| GB-125 | |  | | 6.8.5 | | 3 | | te | | May a declaration in a for loop use the constexpr storage class specifier (such an object being considered implicitly auto if no other storage class specifier used), or does “having storage class auto or register” exclude that case? (Cf. DR#277 regarding interpretation of that sentence.) | | Change “having storage class auto or register” to “with automatic storage duration”, to avoid ambiguity (assuming the constexpr case should be allowed). | |  | |
| GB-126 | |  | | 6.9.2 | | 1-2 | | te | | The rules on external and tentative definitions do not appear to make a file-scope thread\_local declaration of an object without “static” and without an initializer into either an external definition or a tentative definition, contrary to implementation practice. | | If it’s intended for such declarations to be tentative definitions, change “without a storage-class specifier or with the storage-class specifier static” in paragraph 2 to “without the storage-class specifier extern”. Otherwise, say “without the storage-class specifier extern or thread\_local” there, and, in paragraph 1, change “an initializer” to “an initializer, or has file scope and the storage-class specifier thread\_local”. | |  | |
| GB-127 | |  | | 6.9.2 | | 2 | | te | | The “initializer equal to { 0 }” wording is contrary to the intent of making the quantum exponent of a default initialized object of decimal floating type implementation-defined, as taken literally it would prevent the choice of the least possible exponent (all bits zero) if the object with a tentative definition is, or starts with, a scalar with decimal floating type. | | Change “with the composite type as of the end of the translation unit, with an initializer equal to { 0 }.” to “with an empty initializer and type determined as follows: if the composite type as of the end of the translation unit is an array of unknown size, then an array of size one with the composite element type, and otherwise the composite type as of the end of the translation unit.”. | |  | |
| GB-128 | |  | | 6.10 | | 1 | | ed | | The second alternative in the syntax for pp-balanced-token-sequence is not properly indented. | | Indent that alternative to match the previous alternative. | |  | |
| GB-129 | |  | | 6.10.1 | | 8 | | ed | | “Unrecognized preprocessor prefixed parameters” is a plural phrase but subsequent verbs in this sentence expect the singular; “to be evaluate” should be “to be evaluated”. | | Change “is not” to “are not”, “causes” to “cause” and “to be evaluate” to “to be evaluated”. | |  | |
| FR-130 | |  | | 6.10.1 and 6.10.9.1 | |  | | te | | The magic constants for \_\_has\_embed should have symbolic names | | see accompanying document | |  | |
| FR-131 | |  | | 6.10.3 | |  | | te | | It seems that #embed offers multiple ways to express the same feature | | see accompanying document | |  | |
| GB-132 | |  | | 6.10.3.1 | | 10 | | ed | | “Either form” (singular) does not agree with “behave” (plural). | | Change “behave” to “behaves”. | |  | |
| GB-133 | |  | | 6.10.3.1 | | 15 | | ed | | “execution time” should be “execution-time”. | | Change “execution time” to “execution-time”. | |  | |
| GB-134 | |  | | 6.10.3.1 | | 18 | | ed | | In this example, the “first entry” reference is misleading, since scalar initializers inside braces can only contain one expression (the syntax shows that multiple assignment expressions separated by commas are not handled as a use of the comma operator, but as multiple initializers and thus a constraint violation). | | Remove “first entry”, and add the “valid if i.dat produces 1 value,” from the second #embed use in this example to the start of that comment. | |  | |
| GB-135 | |  | | 6.10.3.2 | | 8 | | ed | | Example 4 is a duplicate of Example 3. | | Remove Example 4 (renumbering Example 5 below accordingly). | |  | |
| GB-136 | |  | | 6.10.3.3 | | Constraints | | ed | | “suffix” should be in a fixed-width font. | | Put “suffix” in a fixed-width font. | |  | |
| GB-137 | |  | | 6.10.3.3 | | Constraints | | ed | | The Constraints paragraph is missing a paragraph number. | | Number this paragraph 1, renumbering paragraphs below accordingly. | |  | |
| GB-138 | |  | | 6.10.3.5 | | Constraints | | ed | | The Constraints paragraph is missing a paragraph number. | | Number this paragraph 1, renumbering paragraphs below accordingly. | |  | |
| GB-139 | |  | | 6.10.4.5 | | 8 | | ed | | In the example of valid redefinitions, the redefinition of FUNC\_LIKE isn’t actually valid, because the first definition lacks white space separation present in the second one. This appears to be a previously unnoticed mistake introduced by the conversion to LaTeX for C17. | | Restore presence and absence of white space to the state it had in C11, for both the valid and the invalid definitions, as follows. Inside the replacement lists for the first, third and fourth FUNC\_LIKE definitions, insert a space after ‘(‘ and one before ‘)’. Additionally, in the second FUNC\_LIKE definition, insert a space after ‘(‘ and before ‘)’ in the listing of parameter names (not the replacement list), to illustrate, as in C11, that variation in the presence of space there is OK. | |  | |
| GB-140 | |  | | 6.10.4.5 | | 9 | | ed | | “and”, in the argument to showlist, should not be in bold, since it’s not being used in any way as a standard macro. | | Arrange for “and” to be shown in the same font as the rest of that argument. | |  | |
| GB-141 | |  | | 6.10.5 | | 3 | | ed | | “If a macro invocation spans multiple physical or logical lines” shouldn’t refer to logical lines, since line numbers are always associated with physical lines. | | Remove “or logical”. | |  | |
| US 37  -142 | |  | | 6.10.9.1 | |  | | ge | | Pages 184+: Should the token "202311L" have italics? | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-143 | |  | | 6.10.9.1 | | 1 | | ed | | The references to “asctime functions” (plural) are left over from when asctime\_r was added. Since it was subsequently removed, those references should revert back to singular. | | Change “asctime functions” to “asctime function”, both in the description of \_\_TIME\_\_ and in the forward references. | |  | |
| GB-144 | |  | | 6.10.9.2 | | 2 | | ed | | The example value of \_\_STDC\_ISO\_10646\_\_ corresponds to a very old revision of ISO/IEC 10646 (amendment 9 to the 1993 edition). A more recent version might be appropriate for the example. | | Change “199712L” to “202012L”. | |  | |
| GB-145 | |  | | 6.10.9.3 | | 1 | | ed | | Footnote 225 (intent for version macros) should be attached to all the macros using a date as a version, not just one. | | Attach footnote 225 also to \_\_STDC\_IEC\_60559\_BFP\_\_, \_\_STDC\_IEC\_60559\_DFP\_\_, \_\_STDC\_IEC\_60559\_COMPLEX\_\_ and \_\_STDC\_IEC\_60559\_TYPES\_\_. | |  | |
| GB-146 | |  | | 6.11.2 | | 1 | | ed | | The introduction of the keyword constexpr provides a new way to declare an identifier with internal linkage at file scope. Since such a new feature should not be immediately considered obsolescent, presumably it should only be the pre-existing case (of redeclaring without static after an initial declaration with static) that is listed as obsolescent. | | Change “static” to “static or constexpr”. | |  | |
| GB-147 | |  | | 7.1.2 | | 1, 7 | | ed | | It’s not just Annex K that defines interfaces to be considered together with the ones in clause 7; Annexes F, G and H do so as well and so should be mentioned together with Annex K. | | In paragraph 1, change “that defines \_\_STDC\_LIB\_EXT1\_\_ shall conform to the specifications in Annex K and Subclause K.3 should be read as if it” to “that defines \_\_STDC\_IEC\_60559\_BFP\_\_, \_\_STDC\_IEC\_559\_\_, or \_\_STDC\_IEC\_60559\_DFP\_\_ shall conform to the specifications in Annex F, one that defines \_\_STDC\_IEC\_60559\_COMPLEX\_\_ or \_\_STDC\_IEC\_559\_COMPLEX\_\_ shall conform to the specifications in Annex G, one that defines \_\_STDC\_IEC\_60559\_TYPES\_\_ shall conform to the specifications in Annex H and one that defines \_\_STDC\_LIB\_EXT1\_\_ shall conform to the specifications in Annex K, and those Annexes should be read as if they”. Change “here or in Annex K” to “here or in Annexes F, G, H, or K”.  In paragraph 7, change “Annex K” to “Annexes F, G, H, or K”. | |  | |
| CA5-148 | |  | | 7.2 | | 2 | | Te | | It is not clear what the effect of no NDEBUG defined should be for assert with no or multiple parameters passed in given the quasi-prototype listed in 7.2.1.1#1. | | Add to paragraph 2:  The assert macro shall be implemented as a macro with an ellipsis parameter, not as an actual function. If the macro definition is suppressed to access an actual function, the behavior is undefined.NOTE Nevertheless, when NDEBUG is not defined, the macro acts as a function taking one parameter as indicated by the given prototype. For both assert() and assert(1, 1), the number of arguments do not agree with the number of parameters. A diagnostic is required by 6.5.2.2. | |  | |
| GB-149 | |  | | 7.3.1 | | 5, 6 | | te | | To ensure usability in constexpr initializers, the “\_Complex\_I” and “\_Imaginary\_I” macros should be required to expand to arithmetic constant expressions. | | Change “constant expression” to “arithmetic constant expression” in paragraphs 5 and 6. | |  | |
| US 38  -150 | |  | | 7.3.5.1 | |  | | ed | | Page 194: Remove page break before 7.3.5.2 casin (on page 195). | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-151 | |  | | 7.3.9.2, 7.3.9.6 | |  | | ed | | The equality stated in footnotes for creal and cimag is only correct if imaginary types are supported and if neither part of the number is a NaN. | | In both footnotes 245 and 246, change “creal(z)+cimag(z)\*I” to “CMPLX(creal(z), cimag(z))”. After “of complex type”, insert “, if neither part is a NaN,”. | |  | |
| GB-152 | |  | | 7.3.9.3 | | 2 | | te | | The CMPLX macros should expand to arithmetic constant expressions to be usable in constexpr initializers, not just those of static or thread storage duration. (But if the implementation has floating expressions that are valid in static initializers but not in constexpr initializers, CMPLX calls with such arguments should have the same property.) | | At the end of the paragraph, add another sentence: “The resulting expression shall be an arithmetic constant expression, provided both arguments are arithmetic constant expressions.”. | |  | |
| GB-153 | |  | | 7.3.9.5 | | 2 | | te | | The given equivalence is only valid if imaginary types are supported. | | Change “INFINITY + I \* copysign(0.0, cimag(z))” to “CMPLX(INFINITY, copysign(0.0, cimag(z)))”. | |  | |
| GB-154 | |  | | 7.5 | | footnote 250 | | ed | | The suggested macro definition of errno is not valid because it is not fully protected by parentheses. | | Change “\*errno()” to “(\*errno())”. | |  | |
| US 39  -155 | |  | | 7.6 | |  | | te | | Page 208: Add to 7.6 paragraph 13 (one about FE\_DOWNWARD): The defined macros expand to integer constant expressions whose values are distinct nonnegative values. [This was in C17] | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-156 | |  | | 7.6 | | 14 | | ed | | It seems misleading to say that macros are defined for use with the FENV\_DEC\_ROUND pragma, when standard pragmas do not macro-expand their arguments, so while those identifiers are used as arguments to the pragma, that’s independent of their definitions as macros. | | Remove “, and with the FENV\_DEC\_ROUND rounding control pragma (7.6.3) for specifying a constant rounding direction,”. | |  | |
| GB-157 | |  | | 7.6.1 | | 2 | | te | | The wording seems, in one place, to leave open the possibility that code under an FENV\_ACCESS pragma with state “on” changes the dynamic rounding mode, and then control flow passes to code with state “off”, since that doesn’t fall under “establishes non-default floating-point mode settings”, but it probably wasn’t intended to allow code with FENV\_ACCESS off to execute with a non-default rounding mode like that. | | Change “establishes” to “establishes or is executed with”. | |  | |
| FR-158 | |  | | 7.8.1 | | after p3 | | te | | Add optional macros PRIBN and similar | | see accompanying document | |  | |
| FR-159 | |  | | 7.8.1 | | p3 and p5 | | te | | PRI and SCN macros are missing for new format specifiers | | see accompanying document | |  | |
| GB-160 | |  | | 7.8.2.1 | | footnote 271 | | ed | | Now that two’s complement is required, it is known that the absolute value of the most negative integer is definitely not representable. | | Change “may not be” to “is not”. | |  | |
| GB-161 | |  | | 7.11 | | footnote 272 | | ed | | The current revision of ISO/IEC 9945 is in a single part. | | Change “9945-2” to “9945”. | |  | |
| GB-162 | |  | | 7.11.2.1 | | 9 | | ed | | Since three of the four currencies in the example are no longer in use, the past tense seems more appropriate. | | Change “be used” to “have been used”. | |  | |
| GB-163 | |  | | 7.12 | | 5, 6, 7, 8, 9, 10 | | te | | To be usable in constexpr initializers, constant expressions should be specified as arithmetic constant expressions. | | Change “constant expression” to “arithmetic constant expression”, in paragraphs 5, 6, 7, 8, 9 and 10. | |  | |
| GB-164 | |  | | 7.12.1 | | 5 | | te | | In previous versions of the C standard, the requirements for setting errno and raising exceptions on overflow only applied when default rounding is in effect. The current wording appears to have those requirements regardless of the rounding direction, but it looks like this was an accidental side effect of splitting a single sentence into multiple sentences, rather than an intended change. Requiring errno setting when the rounding direction results in overflow returning a finite value rather than an infinity is liable to be problematic. | | In the penultimate sentence (errno setting), change “If a floating result overflows” to “If a floating result overflows, default rounding is in effect,”. Likewise in the final sentence (exception raising), unless it’s not desired to restore the previous semantics in that case. | |  | |
| GB-165 | |  | | 7.12.3 | | 1 | | ed | | The “min” and “max” subscripts in expressions should be in roman font not italics. | | Change them to roman font (LaTeX \min, \max). | |  | |
| US 40  -166 | |  | | 7.12.4.14 | |  | | te | | Page 243: Add to 7.12.4.14 (tanpi): A pole error may occur if |x| is (n+0.5) for integer n. | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| US 41  -167 | |  | | 7.12.7.4 | |  | | ed | | Page 254: 7.12.7.4 (hypot): Paragraph 3 is empty. | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-168 | |  | | 7.12.9.10 | | 2 | | ed | | “NaN” should not be in a fixed-width font. | | Change “NaN” out of a fixed-width font. | |  | |
| US 42  -169 | |  | | 7.12.11.04 | |  | | ed | | Page 265: 7.12.11.4 (nexttoward): Should a "Returns" paragraph be added? | | If added, it should be similar to the “nextafter” one | |  | |
| US 43  -170 | |  | | 7.12.12.02 | |  | | te | | Page 267: 7.12.12.2 (fmax): Seems like footnotes 297 and 298 are normative; so should not be footnotes, but part of the description. | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-171 | |  | | 7.12.14 | | footnote 299 | | ed | | “double” and “long double” should be in a fixed-width font. | | Change “double” and “long double” to a fixed-width font. | |  | |
| US 44  -172 | |  | | 7.12.14.05 | |  | | ed | | Page 273: 7.12.14.5 (narrow fma): "(" at end of sentence should be moved to next line. | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-173 | |  | | 7.12.16.01, 7.12.16.2, 7.12.16.3, 7.12.16.4, H.11.3.1.1, H.11.3.1.2, H.11.3.2.1, H.11.3.2.2, H.12.3.1, H.12.3.2, H.12.4.1, H.12.4.2 | |  | | te | | Implementation-defined byte ordering made sense originally in TS 18661-2 and TS 18661-3, when there was nothing in the standard defining ways to talk about the implementation’s byte ordering. But now we have <stdbit.h> in the working draft, it seems appropriate, and more useful for users, to specify that the byte ordering matches \_\_STDC\_ENDIAN\_NATIVE\_\_. | | In each of these twelve subclauses, change “The order of bytes in the array is implementation-defined.” to “The order of bytes in the array follows the endianness specified with \_\_STDC\_ENDIAN\_NATIVE\_\_ (7.18.2).”. (In H.11.3.2.1, and H.11.3.2.2, the sentence says “arrays” now and should still say “arrays” after the change. In H.12.3.1, H.12.3.2, H.12.4.1, and H.12.4.2, the current text says “arrays”, but it should say “array” after the change since only a single array is involved for each of those functions.) Also remove item 14 in J.3.12. | |  | |
| US 45  -174 | |  | | 7.12.16.02 | |  | | ed | | Page 276: 7.12.16.2 (decodedecN): "15" after Description seems wrong. | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-175 | |  | | 7.12.16.02 | | 1 | | ed | | There is a stray “15” at the start of the Description. | | Remove “15”. | |  | |
| US 46  -176 | |  | | 7.12.17 | | subclause 1-5, paragraph 2 and 3 | | ed | | The typeface of parentheses is inconsistent. | | Style all parentheses in the indicated paragraphs like the ones in 7.12.17.1. | |  | |
| US 47  -177 | |  | | 7.12.17.01 | | paragraph 2 | | ed | | Bad line break between “isgreater” and “(x,y)”. | | Fix bad line break between “isgreater” and “(x,y)”. | |  | |
| US 48  -178 | |  | | 7.12.17.02 | | paragraph 2 and 3 | | ed | | The intention is to define the macro isgreaterequal(x, y) with reference to the C expression (x) >= (y), not to a mathematical expression. Thus, the three instances of the mathematical symbol for greater than or equal are not correct. | | Replace the three instances of the mathematical symbol for greater than or equal with the C relational operator >= with the appropriate typeface. | |  | |
| GB-179 | |  | | 7.12.17.02, 7.12.17.3, 7.12.17.4, 7.12.17.5 | |  | | ed | | The descriptions of the type-generic comparison macros are inconsistent about whether the expression used to describe the value returned by the macro is shown as a C expression (all C tokens in a fixed-width font, C operators used with corresponding C semantics) or as some kind of mathematical expression (operators not in a fixed-width font and not always C operators at all). The cases using a mathematical expression do not define semantics for how to go from a mathematical result to a C value in type int. Thus, I think the approach in 7.12.17.1 (isgreater), showing a C expression, is to be preferred over the approach in the following four subclauses. | | In 7.12.17.2 (isgreaterequal), change the expressions to be in a fixed-width font (both parentheses and comparison operator), using the C “>=” operator instead of “≥”; this change applies three times in this subclause.  Likewise in 7.12.17.3 (isless).  Likewise in 7.12.17.4 (islessequal), using “<=” instead of “≤”.  Likewise in 7.12.17.5 (islessgreater) (this case also uses the “||” operator, which looks particularly awkward when not in a fixed-width font). | |  | |
| US 49  -180 | |  | | 7.12.17.03 | | paragraph 2 and 3 | | ed | | The intention is to define the macro isless(x, y) with reference to the C expression (x) < (y), not to a mathematical expression. Thus, the three instances of the mathematical symbol for less than are not correct. | | Replace the three instances of the mathematical symbol for less than with the C relational operator < with the appropriate typeface. | |  | |
| US 50  -181 | |  | | 7.12.17.04 | |  | | ed | | Page 278: 7.12.17.4 (islessequal): Better if "(" at end of line were on next line to end up with "(y)". | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| US 51  -182 | |  | | 7.12.17.04 | | paragraph 2 | | ed | | Bad line break between “(“ and “y)”. | | Fix bad line break between “(“ and “y)”. | |  | |
| US 52  -183 | |  | | 7.12.17.04 | | paragraph 2 and 3 | | ed | | The intention is to define the macro islessequal(x, y) with reference to the C expression (x) <= (y), not to a mathematical expression. Thus, the three instances of the mathematical symbol for less than or equal are not correct. | | Replace the three instances of the mathematical symbol for less than or equal with the C relational operator <= with the appropriate typeface. | |  | |
| US 53  -184 | |  | | 7.12.17.05 | | paragraph 2 | | ed | | Though not incorrect, lack of any space around “||” suggests an erroneous order of operations. | | Add space around “||”. | |  | |
| US 54  -185 | |  | | 7.12.17.05 | | paragraph 2 and 3 | | ed | | The intention is to define the macro islessgreater(x, y) with reference to the C expression (x) < (y) || (x) > (y), not to a mathematical expression. Thus, use of mathematical symbols in the two instances of the expression is not appropriate. | | Rewrite the two instances of the expression (x) < (y) || (x) > (y) in the style of a C expression, where <, ||, and > have the typeface for C operators. | |  | |
| US 55  -186 | |  | | 7.12.17.05 | | paragraph 3 | | ed | | Though not incorrect, lack of any space around “||” suggests an erroneous order of operations. | | Add space around “||”. | |  | |
| US 56  -187 | |  | | 7.13 | |  | | te | | Page 280: 7.13 #3: "the floating-point status flags" -> "the floating-point environment" | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-188 | |  | | 7.13 | | 1-2 | | te | | There are no interface changes in setjmp.h, so the addition of a version macro for this header seems inappropriate. | | Remove paragraph 2. In paragraph 1, change “the macros setjmp and \_\_STDC\_VERSION\_SETJMP\_H\_\_” to “the macro setjmp”. | |  | |
| US 57  -189 | |  | | 7.13.2.1 | |  | | te | | Page 281: 7.13.2.1#3 (longjmp): footnote 305: "the floating-point status flags" -> "the floating-point environment" | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-190 | |  | | 7.14.1.1 | | 5 | | te | | AG Reference Bug 728 (https://austingroupbugs.net/view.php?id=728)  =======  Title: Restrictions on signal handlers are excessive  The description of the signal function includes the following:  If the signal occurs other than as the result of calling the abort or raise function, the behavior is undefined if the signal handler refers to any object with static or thread storage duration that is not a lock-free atomic object other than by assigning a value to an object declared as volatile sig\_atomic\_t  This is overly restrictive in several cases:   * It does not allow read access to const-qualified objects * It does not allow read access to string literals * It does not allow referencing a modifiable object with static or thread storage duration (that is not a lock-free atomic object) whose last modification was sequenced before the call to the signal handler and whose next modification will be sequenced after the call to the signal handler. | | Change the text to:  If the signal occurs other than as the result of calling the abort or raise function, the behavior is undefined if the signal handler refers to any object with static or thread storage duration that is not a lock-free atomic object, not a const-qualified object, and not a string literal, other than by assigning a value to an object declared as volatile sig\_atomic\_t, unless the previous modification (if any) to the object happens before the signal handler is called and the return from the signal handler happens before the next modification (if any) to the object | |  | |
| GB-191 | |  | | 7.14.1.1 | | 5 | | te | | It might be appropriate to allow constexpr objects with static storage duration to be accessed in signal handlers. | | Consider the following change: after “not a lock-free atomic object” insert “and that is not declared with the constexpr storage-class specifier”. | |  | |
| FR-192 | |  | | 7.16.1.4 | | p4 | | te | | va\_start becomes too permissive | | see accompanying document | |  | |
| NEN/NL6-193 | | n/a | | 7.16.1.4 | | Paragraph 4 | | te | | Accepted proposal N2975 is a positive change for the C Standard. However, because it definitively declares that all parameters passed into va\_start must be ignored, it leaves no room to add warnings or errors when someone performs an objectively wrong call to va\_start, such as  va\_list vl; va\_start(vl, 1, 2, djwkdeshfes, xD);  These calls should be diagnosable by most implementations. | | 1.) Change the wording of 7.16.1.4, paragraph 4, from:  ~~Any additional arguments are not used by the macro and will not be expanded or evaluated for any reason.~~  To  Any additional arguments are not used by the macro and shall not be evaluated.  2.) Add a new RECOMMENDED PRACTICE subclause to 7.16.1.4:  **Recommended Practice**  Additional arguments beyond the first given to the va\_start macro may be expanded and used in unspecified contexts where they are unevaluated. For example, an implementation diagnoses potentially erroneous input for a call to va\_start such as:  #include <stdarg.h> void miaou (…) {  va\_list vl;  va\_start(vl, 1, 3.0, “12”, xd); // diagnostic encouraged  /\* .. \*/  va\_end(vl);  }  Simultaneously, va\_start usage consistent with older revisions of this document should not produce a diagnostic:  #include <stdarg.h> void neigh (int last\_arg, …) {  va\_list vl;  va\_start(vl, last\_arg); // no diagnostic  /\* .. \*/  va\_end(vl);  } | |  | |
| GB-194 | |  | | 7.17.4.1 | | 2 | | ed | | It’s unfortunate that the final semicolon of the second list item is now pushed onto the next line when previously it was not. (Similar issues with formatting around code fragments appear elsewhere as well.) | | Avoid line breaks between code fragments in text and adjacent punctuation. | |  | |
| GB-195 | |  | | 7.17.6 | | 1 | | te | | “same representation and alignment requirements” leaves open whether other semantics match the corresponding direct type; in particular, whether conversions to atomic\_bool have the semantics of conversions to bool. | | Preferably, change “a type that has the same representation and alignment requirements as the corresponding direct type” to “a typedef name for the corresponding direct type”, and remove footnote 316. If that is considered problematic, however, add “and other semantics” after “alignment requirements”, and, at the end of footnote 316, add a sentence “Other semantics include that conversions to atomic\_bool behave the same as conversions to bool.”. | |  | |
| GB-196 | |  | | 7.17.7.5 | | 1 | | te | | The last “For address types” sentence doesn’t seem to apply to anything, if “address types” means “pointer types”, since these operations only apply to integer types, not pointers. | | Remove the last sentence. | |  | |
| GB-197 | |  | | 7.17.7.5 | | 1 | | te | | “any atomic integer type” is not entirely clear as to whether it means any of the types listed in subclause 7.17.6 (Atomic integer types) or whether it includes the atomic version of any integer type; the latter seems the more natural reading, unless some rule of interpretation means the subclause heading serves as a definition. This means that disallowing atomic\_bool fails to disallow the atomic versions of enumerations with bool as underlying type, even though those have the same semantics as bool and so probably should be disallowed, because those are different types compatible with bool. Depending on the resolution of the previous comment, it might also not disallow \_Atomic bool if that can be different from atomic\_bool. | | At the end of the first sentence, change “atomic integer type” to “atomic integer type other than \_Atomic bool or the atomic version of an enumeration with underlying type bool”. Remove the second sentence. If the first proposed change for the previous comment is not accepted, “, atomic\_bool,” should be inserted before “or”. | |  | |
| GB-198 | |  | | 7.17.7.5 | | 5 | | ed | | “the compound assignment operators are not guaranteed to operate atomically” is incorrect; they are defined as memory\_order\_seq\_cst operations. However, other differences with the compound assignment operators are not mentioned here. | | Change “The only differences are that the compound assignment operators are not guaranteed to operate atomically, and” to “The only differences are that the compound assignment operators have undefined behavior on signed integer overflow, support more types, and”. | |  | |
| GB-199 | |  | | 7.18 | |  | | ed | | The practice used for paragraph numbering in this subclause does not match that used elsewhere. In this subclause only the Synopsis for each function family is numbered; the practice used elsewhere would indicate having three numbered paragraphs in each Description / Returns, where at present the paragraphs are unnumbered. | | Add the paragraph numbers in Description / Returns. | |  | |
| NEN/NL7-200 | | n/a | | 7.18 | | n/a | | te | | Users have commented that the bit utility functions would be easier to read if the suffix had a leading underscore in front of it. | | Add a “\_” before the “us”, “ui”, “uc”, “ul” and “ull” suffixes for the functions present in this clause and any added before the final revision of this C standard. | |  | |
| NEN/NL3-201 | | n/a | | 7.18 | | n/a | | te | | Due to time constraints and wording issues, fundamental bit utilities found in N3022. One of the chief issues was in the proposal trying to accommodate CHAR\_BIT ≠ 8 platforms (in particular, allow any platform where CHAR\_BIT was a multiple of 8, such as CHAR\_BIT ≡ 16,32, or 64). While noble, this effectively tanked useful functionality that could be present if these contentions were not present. | | Two potential changes based on N3022 should be polled by the Committee:   * Accept “Endian Aware Load/Store”, but with the change that CHAR\_BIT must be equivalent to 8. * Accept “Memory Reversal”, but with the change that CHAR\_BIT must be equivalent to 8. | |  | |
| NEN/NL2-202 | | n/a | | 7.18 | | n/a | | te | | Due to time constraints and wording issues, fundamental bit utilities found in N3022 were not voted on, despite some entities and representatives donating extra time to try. We believe that some effort should be made again to add the rotate bit functionality. | | Poll whether the committee should accept “Rotate Left” and “Rotate Right” functions, with the caveat that they use unsigned integers. This removes the undefined behavior issue for dealing with the most negative argument values. | |  | |
| US 58  -203 | |  | | 7.18.\* | | 1 | | te | | From N3022 <https://www.open-std.org/jtc1/sc22/WG14/www/docs/n3022.htm> section 5.1.2 Question 1 Use unsigned integers for the function parameters for counts and the count-like return value types. | | Change the return types for the following functions:   * stdc\_count\_ones (and all derivatives) from int  to unsigned int * stdc\_count\_zeros (and all derivatives) from int to unsigned int * stdc\_leading\_ones (and all derivatives) from int to unsigned int * stdc\_leading\_zeros (and all derivatives) from int  to unsigned int * stdc\_trailing\_ones (and all derivatives) from int  to unsigned int * stdc\_trailing\_zeros (and all derivatives) from int to unsigned int * stdc\_first\_leading\_one (and all derivatives) from int  to unsigned int * stdc\_first\_leading\_zero (and all derivatives) from int  to unsigned int * stdc\_first\_trailing\_one (and all derivatives) from int  to unsigned int * stdc\_first\_trailing\_zero (and all derivatives) from int  to unsigned int   stdc\_bit\_width (and all derivatives) from int  to unsigned int | |  | |
| NEN/NL4-204 | | n/a | | 7.18.3 to 7.18.12, 7.18.14 | | (all) | | te | | The bit and byte utilities which count the number of bits under some criteria have fine definitions but return signed numbers. Signed numbers have undefined behavior associated with their usage and present representation problems when maximally negative integer values are used. | | Change all of the synopses and definitions for the following functions/macros with the given prefix to return “unsigned int” or “suitably large unsigned integer(type)” numbers:  stdc\_bit\_width, stdc\_count\_(ones/zeros), stdc\_first\_(leading/trailing)\_(one/zero), stdc\_count\_(leading/trailing\_(ones/zeros). | |  | |
| GB-205 | |  | | 7.18.11, 7.18.14 | |  | | ed | | There is no apparent reason for the wording about generic\_return\_type for stdc\_count\_zeros and stdc\_bit\_width to be different from that used for other similar generic functions. | | Replace the two sentences about generic\_return\_type for stdc\_count\_zeros (7.18.11) and stdc\_bit\_width (7.18.14) by a copy of the one sentence used for stdc\_count\_ones. | |  | |
| US 59  -206 | |  | | 7.18.17 | | 1 | | te | | From N3022 <https://www.open-std.org/jtc1/sc22/WG14/www/docs/n3022.htm>  section 5.1.1 Question 0  Adopt an UnsignedType  stdc\_rotate(UnsignedType value, int count); | | **7.18.17 Rotate**  Synopsis  unsigned char stdc\_rotate\_leftuc(unsigned char value, int count);  unsigned short stdc\_rotate\_leftus(unsigned short value, int count);  unsigned int stdc\_rotate\_leftui(unsigned int value, int count);  unsigned long stdc\_rotate\_leftul(unsigned long value, int count);  unsigned long long stdc\_rotate\_leftull(unsigned long long value, int count);  generic\_value\_type stdc\_rotate\_left(  generic\_value\_type value, generic\_count\_type count);  Description  The stdc\_rotate functions perform a bitwise rotate left or right. This operation is typically known as a left or right circular shift.  Returns  Let N be the width corresponding to the type of the input value. Let r be count % N.  — If r is 0, returns value;  — otherwise, if r is positive, returns (value << r) | (value >> (N - r));  — otherwise, if r is negative, returns (value >> -r) | (value << (N - -r)).  The type-generic function (marked by its generic\_value\_type argument) returns the above-described result for a given input value so long as the generic\_value\_type is an  — standard unsigned integer type, excluding bool;  — extended unsigned integer type;  — or bit-precise unsigned integer type whose width matches a standard or extended integer type, excluding bool.  The generic\_return\_type type shall be suitably large unsigned integer type capable of representing the width of the computed result. The generic\_count\_type shall be a signed integer type. | |  | |
| GB-207 | |  | | 7.22.1.5 | | footnote 321 | | ed | | This footnote should mention bit-precise types as possibly being wider than uintmax\_t. | | Change “exception of” to “exception of unsigned bit-precise integer types and”. | |  | |
| GB-208 | |  | | 7.22.5 | | 1 | | ed | | “0 of the corresponding type” doesn’t allow for integer promotions. | | After “0 of the corresponding type” insert “, converted according to the integer promotions,”. | |  | |
| GB-209 | |  | | 7.23.2 | | 7-8 | | te | | AG Reference Bug 689 (https://austingroupbugs.net/view.php?id=689)  =======  Title: Possibly unintended allowance for stdio deadlock  7.23.2 Streams states:  7 Each stream has an associated lock that is used to prevent data races when multiple threads of execution access a stream, and to restrict the interleaving of stream operations performed by multiple threads. Only one thread may hold this lock at a time. The lock is reentrant: a single thread may hold the lock multiple times at a given time.  8 All functions that read, write, position, or query the position of a stream lock the stream before accessing it. They release the lock associated with the stream when the access is complete.  and 7.23.3 Files states in para 3:  When a stream is line buffered, characters are intended to be transmitted to or from the host environment as a block when a new-line character is encountered. Furthermore, characters are intended to be transmitted as a block to the host environment when a buffer is filled, when input is requested on an unbuffered stream, or when input is requested on a line buffered stream that requires the transmission of characters from the host environment.  Although support for the latter is implementation-defined, if the "when input is requested" parts are implemented, it creates the potential for deadlock.  For example, if thread A is holding the lock associated with a line-buffered output stream and its progress is blocked waiting for thread B to do something, and thread B happens to use stdio for reading any unbuffered (or line buffered with an empty buffer) stream as part of its operation, the requirement in 7.23.2 para 8 means the program will deadlock. This behavior seems highly undesirable and unintended. | | Requiring deadlock detection seems too onerous, given that POSIX makes it optional for pthread\_mutex\_lock, but perhaps there ought at least to be an attempt at detection. The question is then what to do if deadlock is not detected but nor has it been established that a deadlock situation does not exist. Since implementing support for the flush is optional anyway, just not doing the flush seems like an acceptable solution.  After:  All functions that read, write, position, or query the position of a stream lock the stream before accessing it. They release the lock associated with the stream when the access is complete.  add:  If the lock is not immediately available, the function waits for it to become available, except in the following circumstances. If the stream is line buffered and is open for writing or for update, and the reason the function is attempting to lock the stream is because it is going to request input on another stream that is unbuffered, or is line buffered and requires the transmission of characters from the host environment (see 7.23.3), then the function attempts to determine whether a deadlock situation exists. If a deadlock situation is found to exist, the function shall fail. If the function is able to establish that a deadlock situation does not exist, it shall wait for the lock to become available. If the function does not establish whether or not a deadlock situation exists, it shall continue as if it had already locked the stream, found its buffer to be empty, and released the lock. | |  | |
| GB-210 | |  | | 7.23.5.3 | | 3 | | te | | The undefined behaviour with the mode string would seem to be better classified as implementation defined (which forces a conforming implementation to document it). | | Change “is undefined” to “is implementation-defined”. | |  | |
| IE-211 | |  | | 7.23.5.3 | | 5 | | te | | Adopt changes from N3059 <https://www.open-std.org/jtc1/sc22/WG14/www/docs/n3059.pdf> | | From N3059 Section 1.1: Opening a file with exclusive mode (’x’ as the last character in the mode argument) fails if the file already exists or cannot be created. ~~Otherwise, the file is created with exclusive (also known as non-shared) access to the extent that the underlying system supports exclusive access.~~ The check for the existence of the file and the creation of the file if it does not exist is atomic with respect to other threads and processes. If the implementation is not capable of performing the check for the existence of the file and the creation of the file atomically, it shall fail instead of performing a non-atomic check and creation. | |  | |
| US 60  -212 | |  | | 7.23.5.3 | | 5 | | te | | Adopt changes from N3059 <https://www.open-std.org/jtc1/sc22/WG14/www/docs/n3059.pdf> | | From N3059 Section 1.1: Opening a file with exclusive mode (’x’ as the last character in the mode argument) fails if the file already exists or cannot be created. ~~Otherwise, the file is created with exclusive (also known as non-shared) access to the extent that the underlying system supports exclusive access.~~ The check for the existence of the file and the creation of the file if it does not exist is atomic with respect to other threads and processes. If the implementation is not capable of performing the check for the existence of the file and the creation of the file atomically, it shall fail instead of performing a non-atomic check and creation. | |  | |
| IE-213 | |  | | 7.23.5.3 | | 6 | | te | | Adopt changes from N3059 <https://www.open-std.org/jtc1/sc22/WG14/www/docs/n3059.pdf> | | From N3059 Section 1.2: Opening a file with append mode (’a’ as the first character in the mode argument) causes all subsequent writes to the file to be forced to the current end-of-file at the point of buffer flush or actual write, regardless of intervening calls to the fseek ~~function~~, fsetpos, or rewind functions. Incrementing the current end-of-file by the amount of data written is atomic with respect to other threads writing to the same file provided the file was also opened in append mode. If the implementation is not capable of incrementing the current end-of-file atomically, it shall fail instead of performing non-atomic end-of-file writes. In some implementations, opening a binary file with append mode (’b’ as the second or third character in the above list of mode argument values) may initially position the file position indicator for the stream beyond the last data written, because of null character padding. | |  | |
| US 61  -214 | |  | | 7.23.5.3 | | 6 | | te | | Adopt changes from N3059 <https://www.open-std.org/jtc1/sc22/WG14/www/docs/n3059.pdf> | | From N3059 Section 1.2: Opening a file with append mode (’a’ as the first character in the mode argument) causes all subsequent writes to the file to be forced to the current end-of-file at the point of buffer flush or actual write, regardless of intervening calls to the fseek ~~function~~, fsetpos, or rewind functions. Incrementing the current end-of-file by the amount of data written is atomic with respect to other threads writing to the same file provided the file was also opened in append mode. If the implementation is not capable of incrementing the current end-of-file atomically, it shall fail instead of performing non-atomic end-of-file writes. In some implementations, opening a binary file with append mode (’b’ as the second or third character in the above list of mode argument values) may initially position the file position indicator for the stream beyond the last data written, because of null character padding. | |  | |
| US 62  -215 | |  | | 7.23.6.1 | | paragraph 6 | | ed | | In the “#” bullet, the “b” in “For b conversion, …” has the wrong typeface. | | In the “#” bullet, fix the typeface for the “b” in “For b conversion, …”. | |  | |
| US 63  -216 | |  | | 7.23.6.1 | | paragraph 8 | | ed | | In the a,A bullet, in the text beginning with, “If the precision p is present”, the capitalization and italics for the p’s is wrong. The (formatting) precision is denoted *P* (upper case, italic); the format of the (decimal floating) type is denoted by *p* (lower case, italic). | | In the a,A bullet, change the capitalization and italics of the p’s in the text beginning with, “If the precision p is present” to get “If the precision *P* is present (in the conversion specification) and is zero or at least as large as the precision *p* (5.2.4.2.2) of the decimal floating type, the conversion is as if the precision were missing. If the precision *P* is present (and nonzero) and less than the precision *p* of the decimal floating type …”. | |  | |
| FR-217 | |  | | 7.23.6.1 and 7.31.2.1 | |  | | te | | Make %B optional | | see accompanying document | |  | |
| GB-218 | |  | | 7.23.6.1, 7.23.6.2, 7.24.1.3, 7.29.3.5, 7.31.2.1, 7.31.2.2 | |  | | te | | Similar to the above comment on 7.23.5.2 fopen the undefined behaviour with an invalid conversion specification would be better stated as being an implementation defined behaviour because the implementation can choose how to handle the invalid specification, and its not the same problem arising from a mismatch between the type indicated in the format specification and the type of the object supplied as a function argument, which is UB. (Note that unsupported %wN and %wfN are defined as errors rather than as undefined behavior.) | | In each appropriate place (depending on whether the committee considers this appropriate for all the places describing ways format strings can be invalid), change “undefined” to “implementation-defined”. This does not apply to cases where a format argument has a bad type or value. | |  | |
| GB-219 | |  | | 7.23.6.1, 7.31.2.1 | |  | | ed | | The description of %n wrongly refers to plural “length modifiers”, when a single conversion specification has only a single length modifier (which may consist of multiple characters). | | Change (for both fprintf and fwprintf) “by the length modifiers” to “by the length modifier”. Change “no length modifiers are” to “no length modifier is”. | |  | |
| GB-220 | |  | | 7.23.6.1, 7.31.2.1 | |  | | ed | | The description of %a and %A output for decimal floating types is confusing because it inconsistently uses p for two different precisions. | | The following fix (for both fprintf and fwprintf) should be checked carefully by the C floating-point group. It is written on the basis that the formatting precision is intended to be uppercase P while the precision of the type is intended to be lowercase p.  In the part of the description starting “If the precision p”, change that to “If the precision P” (P in italics). Later in that sentence, in “as the precision p (5.2.4.2.2) should have p put in italics but remain lowercase. In the following sentence, also starting “If the precision p”, also change that to “If the precision P” (P in italics); the second reference to p in that sentence should remain as a lowercase, italic p. | |  | |
| GB-221 | |  | | 7.23.6.1, 7.31.2.1 | |  | | ed | | The formatting of references to conversion specifiers and their output is inconsistent. | | In 7.23.6.1 paragraph 6, ‘#’ flag, do not use bold for ‘b and ‘0b’. In paragraph 7, each integer length modifier, do not use bold for ‘n’. In paragraph 8, content of the description of unsigned integer specifiers, use fixed-wifth type for ‘b’. In paragraph 13, do not use bold for ‘e’. In paragraph 14, do not use bold for ‘0B’. Likewise in the same paragraphs in 7.31.2.1 (fwprintf). | |  | |
| GB-222 | |  | | 7.23.6.2, 7.31.2.2 | |  | | ed | | In the description of %lc, there is no space before the sentence “No null wide character is added.”. | | Add such a space (for both fscanf and fwscanf). | |  | |
| GB-223 | |  | | 7.23.6.2, 7.31.2.2 | |  | | ed | | The description of the ‘l’ length modifier is missing the ‘b’ conversion specifier. | | In paragraph 11 (for both fscanf and fwscanf), add ‘b’ to the start of the list of conversion specifiers to which ‘l’ results in an argument of type pointer to long int or unsigned long int. | |  | |
| GB-224 | |  | | 7.23.7.2, 7.31.3.2 | |  | | te | | The description of fgets is unclear about the semantics when n is zero or negative, when “reads at most one less than the number of characters specified by n” would mean reading a negative number of characters. The same issue applies to fgetws. | | If undefined behavior is intended, add “If n is negative or zero, the behavior is undefined.” to the end of 7.23.7.2 paragraph 2 and 7.31.3.2 paragraph 2. If some defined behavior is intended, appropriate wording would need to be added depending on what exactly that behavior should be (whether any characters, even a null character, should be written, and what the return value should be). | |  | |
| GB-225 | |  | | 7.23.9.1 | | 3 | | ed | | The early page break at the end of the fgetpos subclause seems inconsistent with the lack of page breaks after other functions here. | | Remove that page break. | |  | |
| GB-226 | |  | | 7.23.9.4 | | 3 | | ed | | The return value -1L appears to be intended as a C token (as shown by the suffix), so should be in a fixed-width font. | | Change “-1L” to a fixed-width font. | |  | |
| GB-227 | |  | | 7.24 | | 1 | | ed | | The reference to “five types” is outdated. | | Change “five types” to “six types”. | |  | |
| FR-228 | |  | | 7.24.1, 7.31.4.1, K.3.5.3 and K.3.9.1 | |  | | te | | There are semantic changes and inconsistencies for strtol, scanf and similar functions | | see accompanying document | |  | |
| GB-229 | |  | | 7.24.1.3 | | 2 | | te | | The description of strfromf as being equivalent to a call to snprintf is unclear about whether the equivalence includes the default argument promotions that occur at the call site when a float argument is passed in variable arguments. In particular, if the default argument promotions are part of the equivalence, that would imply a convertFormat operation (so converting signaling NaNs to quiet with “invalid” raised, and possible other effect on NaN payloads, for example, that could be relevant when the snprintf output for NaN uses an implementation-defined n-char-sequence). Whereas if that’s not part of the equivalence, it’s implementation-defined whether a convertFormat operation is applied in argument passing, and any such operation would be conversion to the same format. | | After “except that”, insert “the default argument promotions are not applied and”. | |  | |
| GB-230 | |  | | 7.24.1.5, 7.31.4.1.2 | |  | | te | | Where the Returns paragraphs for strtod and wcstod say “If no conversion could be performed, zero is returned.”, this doesn’t specify the sign. | | Change “zero” to “positive or unsigned zero”, for both strtod and wcstod. | |  | |
| GB-231 | |  | | 7.24.1.5, 7.31.4.1.2 | |  | | ed | | Paragraph 4 of 7.24.1.5 appears as two paragraphs (when it was previously one) but only the first is given a paragraph number. Likewise paragraph 10 has three paragraphs. The same applies to wcstod (7.31.4.1.2) paragraphs 3 and 9. | | Add paragraph numbers to all the paragraphs in question. | |  | |
| GB-232 | |  | | 7.24.1.7,  7.31.4.1.4 | |  | | te | | AG Reference Bug 700 (https://austingroupbugs.net/view.php?id=700)  =======  Title: strtol cannot return LONG\_MIN with two's complement long  The description of strtol, strtoll, strtoul, and strtoull states:  If the subject sequence begins with a minus sign, the value resulting from the conversion is negated (in the return type).  The parenthetical phrase "(in the return type)" was added in C99 in response to DR #006 http://www.open-std.org/jtc1/sc22/wg14/docs/rr/dr\_006.html  This clarified the behavior of strtoul but it broke strtol, because with two's complement signed long, it is not possible to produce the value LONG\_MIN by negating a positive value "in the return type".  Likewise for the equivalent wcsto\* functions. | | Change the text to:  If the subject sequence begins with a minus sign, the value resulting from the conversion is negated; for functions whose return type is an unsigned integer type this negation is performed in the return type.  Make the same change for wcstol, wcstoll, wcstoul, and wcstoull. | |  | |
| GB-233 | |  | | 7.24.1.7, 7.31.4.1.4 | |  | | ed | | In the description of base 2 for strtol and wcstol, ‘0b’ and ‘0B’ should not be in bold, since ‘0x’ and ‘0X’ aren’t. | | For both strtol and wcstol, change ‘0b’ and ‘0B’ out of bold type. | |  | |
| GB-234 | |  | | 7.24.2.1 | | 4 | | ed | | Whilst the intent of the paragraph is fair, it would seem to be a Note (nay warning) rather than Recommend Practice. | | Change to a Note. | |  | |
| GB-235 | |  | | 7.24.3, 7.24.3.1 | |  | | te | | The general description of memory management functions in 7.24.3 paragraph 1 implies that the result of aligned\_alloc is suitably aligned for “any type of object with a fundamental alignment requirement and size less than or equal to the size requested”, while the specific wording in 7.24.3.1 paragraph 2 refers to alignment specified by a function argument. These are consistent with each other, but it’s not clear if the consequence (the alignment must be sufficient for whichever is greater of max\_align\_t (subject to object size) and that passed as an argument) is as intended. | | First, it’s necessary to decide whether it should be possible for aligned\_alloc to return a pointer that is less-aligned than malloc would be required to return for the same size, or whether the consequence of the existing wording is as desired.  In the first case (aligned\_alloc should be able to return a less-aligned pointer), change 7.24.3 paragraph 1: “The pointer returned” should change to “The pointer returned by calloc, malloc, and realloc”.  In the second case (aligned\_alloc should be required to satisfy the same alignment requirements as malloc, in addition to its own), a footnote in 7.24.3.1 paragraph 2 would be appropriate: after “whose alignment is specified by alignment,”, add a footnote “The alignment requirements from 7.24.3 also apply even if alignment < alignof(max\_align\_t).”. | |  | |
| GB-236 | |  | | 7.24.6.1 | | footnote 368 | | ed | | Now that two’s complement is required, it is known that the absolute value of the most negative integer is definitely not representable. | | Change “may not be” to “is not”. | |  | |
| GB-237 | |  | | 7.24.6.3 | | 4 | | ed | | In “a subsequent call to the strerror function”, “strerror” should be in a fixed-width font. | | Change “strerror” in that phrase to a fixed-width (and bold) font. | |  | |
| GB-238 | |  | | 7.24.7 | | 1 | | te | | AG Reference Bug 708 (https://austingroupbugs.net/view.php?id=708)  =======  Title: mblen, mbtowc, and wctomb data races  As per https://www.open-std.org/jtc1/sc22/wg14/www/docs/n2396.htm#dr\_498 it seems that in Oct 2018 the committee agreed in principle with the goal of N2281, and solicited a new paper from the author.  Hopefully a new paper has been submitted and will lead to a satisfactory outcome. However, if that is not the case then the Austin Group strongly recommends that the fallback position should not be the status quo, but to align with POSIX.  The current situation is that the C standard requires mblen, mbtowc, and wctomb to avoid data races, but POSIX says that they need not be thread-safe. POSIX currently still refers to C99, so this will not become a problem until the next POSIX revision, which will refer to C17, is approved. (It is in the late stages of development.)  Since DRs for C17 are not being accepted, in order not to force POSIX implementations to change, the next POSIX revision will need to state that it does not defer to C17 regarding thread-safety of these functions. Hopefully the revision after next will be able to return to deferring to the C standard, but this will depend on whether, and under what conditions, the relevant future C standard still requires them to avoid data races.  If wording cannot be agreed for C23 that specifies the precise conditions under which these functions avoid data races, then the standard should simply change to match POSIX and state that they need not avoid data races. | | Append to 7.24.7 para 1:  These functions are not required to avoid data races with other calls to the same function. | |  | |
| NEN/NL8-239 | | n/a | | 7.29.1 | | Paragraph 4  Paragraph 5 | | te | | Previously, N2878 was held up because it needed approval from the Austin Group. During the meeting the paper was postponed and corrections were asked for, F.T. stated that the Austin Group approves the changes and direction. This paper is a tiny fix that is about as close to editorial as possible, changes the paper to match existing practice, and allows for more implementation flexibility. | | Add the wording of N2878 (or its successor revision) to the C Standard. Paragraph 4 should have the following text added:  nsec\_t  which is an implementation-defined integer type capable of representing the range [0, 999999999];  and paragraph 5 should have the following change done for the timespec structure:  ~~long tv\_nsec; // nanoseconds – [0, 999999999]~~  nsec\_t tv\_nsec; // nanoseconds -- [0, 999999999] | |  | |
| GB-240 | |  | | 7.29.2.3 | | 3 | | te | | AG Reference Bug 1614 (https://austingroupbugs.net/view.php?id=1614)  ========  Title: meaning of (time\_t)-1 return from mktime  The mktime description states, under "Returns":  The mktime function returns the specified calendar time encoded as a value of type time\_t. If the calendar time cannot be represented, the function returns the value (time\_t)(-1).  An application writer reading this is likely to infer from the way it is worded that when mktime returns (time\_t)-1 it means that the calendar time to be returned was not representable.  Indeed, searching open source applications for calls to mktime turns up many uses where a return of (time\_t)-1 is assumed to indicate this. For example, the Python time module turns an error return of (time\_t)-1 from the C library mktime function into a PyExc\_OverflowError exception. A limited search also found no applications that treat a return of (time\_t)-1 as possibly indicating some other kind of failure, although an extensive search might do so.  However, this interpretation of the "Returns" text seems to be at odds with the committee's response in 1994 to DR #136, which says that mktime can return (time\_t)-1 for broken-down times that refer to times in the "spring-forward gap" (see https://www.open-std.org/jtc1/sc22/wg14/www/docs/dr\_136.html)  One way to reconcile the two is to observe that, since local time and Daylight Saving Time are implementation-defined, an implementation could define them in such a way that times in the spring-forward gap are converted to a value that cannot be represented. For example, it could say they are converted to UINT64\_MAX if time\_t is a signed 64-bit integer type. Then the C standard would require mktime to return (time\_t)-1 because UINT64\_MAX can't be represented in that time\_t type.  If this was the committee's reasoning in 1994, then it would be helpful to have this confirmed. Another possibility is that they simply overlooked the significance of the way the "Returns" clause is worded.  Deciding what the standard currently requires is one thing, but there is also the (perhaps more important) matter of what C23 should require.  The current state of applications is the result of a combination of factors:  1. Application writers interpreting the "Returns" text in the way described above.  2. Almost all implementations not returning (time\_t)-1 for times in the spring-forward gap, with the consequence that, over the decades, applications have been mostly developed and run on such systems. This is known because the original NIST-PCTS tested for this behavior (as stated in DR #136) and The Open Group test suite, which has been used to certify dozens of systems as POSIX conforming (since 2003) and (since 1990) as XPG3, XPG4, and UNIX conforming, also tests for it. The list of certified systems includes Microsoft Windows (NT 3.5, 3.51 and 4.0 were certified POSIX conforming). Running the example program from DR #136 on some non-certified systems identified that glibc and FreeBSD do not return (time\_t)-1. So far NetBSD is the only system that has been confirmed as returning (time\_t)-1, but it can actually behave both ways: there is a NO\_ERROR\_IN\_DST\_GAP compile time option to control it (with the (time\_t)-1 return as the default). DR #136 says that Arthur David Olson's popular "tz" time zone software returned (time\_t)-1 (in 1994) and this may have influenced the response to that DR, as it is known to have been adopted by many systems. However, it is now evident that almost all (if not all) systems that adopted it modified it so that it does not return (time\_t)-1.  3. Even when run on an implementation that does return (time\_t)-1 for times in the spring-forward gap, occurrences of this condition are rare, and the occasional application misbehavior (by treating it the same as the "cannot be represented" case) may have gone unnoticed.  Given that almost all implementations of mktime do not return (time\_t)-1 for times in the spring-forward gap, and that applications which can handle a return of (time\_t)-1 appropriately for any condition other than the calendar time to be returned being unrepresentable seem to be exceedingly rare, it would benefit application portability if C23 upholds the meaning of the (time\_t)-1 return as being that the calendar time to be returned cannot be represented and disallows returning (time\_t)-1 for other reasons.  If mktime can return (time\_t)-1 for other reasons, this creates a problem for applications if they want to to distinguish the different cases. If (time\_t)-1 is returned when tm\_isdst is negative, they can try a second call with tm\_isdst set to 0 (or 1) and assume that if this succeeds the original return of (time\_t)-1 was caused by a DST transition, but this additional code is unnecessary on almost all systems, and it only handles that one additional case. If (time\_t)-1 is returned when tm\_isdst is not negative, how are applications to distinguish the "cannot be represented" case from other cases that they might prefer to treat as non-fatal?  If the source of the broken-down time was from a file or database, or user input, then perhaps it is not much of a problem if the return of (time\_t)-1 is treated as a fatal error (with a misleading error message), but when mktime is used to perform manipulations of the struct tm members, it is more of a problem. DR #136 suggests that tm\_isdst is left as 0 or 1 when doing such manipulations, and that may be true when the time adjustment is small, but when adding or subtracting any whole number of days, setting tm\_isdst to -1 is a perfectly reasonable thing for an application to do.  Not returning (time\_t)-1 is simply better for applications. The glibc source contains this comment about it:  The requested time probably falls within a spring-forward gap of size DT. Follow the common practice in this case, which is to return a time that is DT away from the requested time, [...] In practice, this is more useful than returning -1.  On a system which does not return (time\_t)-1, if an application wants to detect whether the broken-down time is in a spring-forward gap, all it needs to do is look for appropriate changes to the struct tm fields after mktime returns. (In the example from DR #136, tm\_hour changes from 2 to either 1 or 3.)  Finally, there is also a subtle problem with the "cannot be represented" wording. It is not clear if it means cannot be represented in a time\_t, or cannot be represented in the time\_t encoding used for the return value. Microsoft Windows, and perhaps some other systems, uses a time\_t encoding that does not include negative values (even though its time\_t is signed), and thus returns (time\_t)-1 if the calculated calendar time is negative. To ensure this is clearly allowed, the wording should be changed to "cannot be represented in the time\_t encoding used for the return value". | | Four options are given based on two independent decisions (what C17 requires and what C23 should require). The Austin Group has a strong preference for options 1 and 2 over options 3 and 4.  Option 1  A return of (time\_t)-1 means that the calendar time to be returned could not be represented. Implementations can return (time\_t)-1 for times in the spring-forward gap by defining local time and Daylight Saving Time in such a way that times in the spring-forward gap are converted to a value that cannot be represented, but for the sake of application portability this loophole should be closed in C23 by changing:  ... not restricted to the ranges indicated above. 389) On successful completion, the values of the tm\_wday and tm\_yday components of the structure are set appropriately, and the other components are set to represent the specified calendar time, but with their values forced to the ranges indicated above; the final value of tm\_mday is not set until tm\_mon and tm\_year are determined.  to:  ... not restricted to the ranges indicated above. If the local time to be used for the conversion is one that includes Daylight Saving Time adjustments, a positive or zero value for tm\_isdst causes the mktime function to perform the conversion as if Daylight Saving Time, respectively, is or is not in effect for the specified time. A negative value causes it to attempt to determine whether Daylight Saving Time is in effect for the specified time; if it determines that Daylight Saving Time is in effect it produces the same result as an equivalent call with a positive tm\_isdst value, otherwise it produces the same result as an equivalent call with a tm\_isdst value of zero. 389) On successful completion, the components of the structure are set to the same values that would be returned by a call to the localtime function with the calculated calendar time as its argument.  and changing footnote 389 to read:  If the broken-down time specifies a time that is either skipped over or repeated when a transition to or from Daylight Saving Time occurs, it is unspecified whether the mktime function produces the same result as an equivalent call with a positive tm\_isdst value or as an equivalent call with a tm\_isdst value of zero.  Also, under "Returns" change:  If the calendar time cannot be represented  to:  If the calendar time cannot be represented in the time\_t encoding used for the return value  Option 2  A return of (time\_t)-1 can mean other things than that the calendar time to be returned could not be represented, because [insert explanation here], but for the sake of application portability C23 should disallow this by changing:  ... not restricted to the ranges indicated above. 389) On successful completion, the values of the tm\_wday and tm\_yday components of the structure are set appropriately, and the other components are set to represent the specified calendar time, but with their values forced to the ranges indicated above; the final value of tm\_mday is not set until tm\_mon and tm\_year are determined.  to:  ... not restricted to the ranges indicated above. If the local time to be used for the conversion is one that includes Daylight Saving Time adjustments, a positive or zero value for tm\_isdst causes the mktime function to perform the conversion as if Daylight Saving Time, respectively, is or is not in effect for the specified time. A negative value causes it to attempt to determine whether Daylight Saving Time is in effect for the specified time; if it determines that Daylight Saving Time is in effect it produces the same result as an equivalent call with a positive tm\_isdst value, otherwise it produces the same result as an equivalent call with a tm\_isdst value of zero. 389) On successful completion, the components of the structure are set to the same values that would be returned by a call to the localtime function with the calculated calendar time as its argument.  and changing footnote 389 to read:  If the broken-down time specifies a time that is either skipped over or repeated when a transition to or from Daylight Saving Time occurs, it is unspecified whether the mktime function produces the same result as an equivalent call with a positive tm\_isdst value or as an equivalent call with a tm\_isdst value of zero.  Also, under "Returns" change:  If the calendar time cannot be represented  to:  If the calendar time cannot be represented in the time\_t encoding used for the return value  Option 3  A return of (time\_t)-1 means that the calendar time to be returned could not be represented. Implementations can return (time\_t)-1 for times in the spring-forward gap by defining local time and Daylight Saving Time in such a way that times in the spring-forward gap are converted to a value that cannot be represented, but they should not need to use this loophole to do so, and C23 should change:  If the calendar time cannot be represented, the function returns the value (time\_t)(-1).  to:  If the calendar time cannot be represented in the time\_t encoding used for the return value, or if the function does not succeed for some other reason, the function returns the value (time\_t)(-1).  Option 4  A return of (time\_t)-1 can mean other things than that the calendar time to be returned could not be represented, because [insert explanation here], and this should be clarified in C23 by changing:  If the calendar time cannot be represented, the function returns  the value (time\_t)(-1).  to:  If the calendar time cannot be represented in the time\_t encoding used for the return value, or if the function does not succeed for some other reason, the function returns the value (time\_t)(-1). | |  | |
| FR-241 | |  | | 7.29.3 | |  | | te | | Misspecification of the time conversion functions | | see accompanying document | |  | |
| GB-242 | |  | | 7.29.3 | | 2 | | ed | | This paragraph does not properly reflect the removal of asctime\_r and ctime\_r. | | Remove “asctime, ctime,”. Change “the following functions” to “asctime, ctime, gmtime and localtime”. | |  | |
| GB-243 | |  | | 7.29.3.2 | | 4 | | ed | | “asctime functions” (plural) should be “asctime function”. | | Change “asctime functions” to “asctime function”. | |  | |
| GB-244 | |  | | 7.29.3.4 | | 2 | | ed | | “converts” should be “convert” to agree with “functions”. | | Change “converts” to “convert”. | |  | |
| GB-245 | |  | | 7.29.3.5 | | 3 | | te | | Section 7.29.3.5 Para 3  AG Reference Bug 739 (https://austingroupbugs.net/view.php?id=739)  =======  Title: strftime %F conversion claims to provide ISO 8601 date format but does so only for a limited year range  The strftime %F conversion is described as:  %F is equivalent to "%Y-%m-%d" (the ISO 8601 date format)  However, if the year is between 0 and 999 this produces at most a three digit year, whereas ISO 8601 specifies a minimum of four digits for years in that range.  Also, if the year is outside the range 0 to 9999, according to wikipedia "To represent years before 0000 or after 9999, [ISO 8601] also permits the expansion of the year representation but only by prior agreement between the sender and the receiver. An expanded year representation [±YYYYY] must have an agreed-upon number of extra year digits beyond the four-digit minimum, and it must be prefixed with a + or − sign". | | There are three different solutions, depending on how much equivalence to ISO 8601 is to be claimed.  Suggested resolution:  Option 1 - only claim ISO 8601 equivalence for years 1000 to 9999  Change the %F description to:  %F is equivalent to "%Y-%m-%d" (the ISO 8601 date format, when the year is between 1000 and 9999 inclusive)  Option 2 - only claim ISO 8601 equivalence for years 0 to 9999  Change the %F description to:  %F is equivalent to "%Y-%m-%d", except that the stored year is filled  as needed with leading zeros so that if the year is between 0 and 999 inclusive, four digits are stored. (This provides the ISO 8601 date format when the year is between 0 and 9999 inclusive.)  Option 3 - full ISO 8601 equivalence  Since "An expanded year representation [±YYYYY] must have an agreed-upon number of extra year digits beyond the four-digit minimum", there needs to be a way for that agreed-upon number to be used in the strftime format string. This would require adding field widths: the wording could be adapted from POSIX.1-2017. The requirement that the year must be prefixed with a + or − sign could be handled either by adding the + flag from POSIX.1-2017 or by stating the need for a + sign for years > 9999 in the description of %F. | |  | |
| GB-246 | |  | | 7.30.1.2, 7.30.1.4 | |  | | ed | | In paragraph 3 of each of 7.30.1.2 and 7.30.1.4, “pointed to by s” should have “s” in a fixed-width font. (For 7.30.1.6, c32rtomb, it’s already OK.) | | Change “s” to a fixed-width font in both those sentences (c8rtomb and c16rtomb). | |  | |
| US 64  -247 | |  | | 7.31.2.1 | | paragraph 8 | | ed | | In the a,A bullet, in the text beginning with, “If the precision p is present”, the capitalization and italics for the p’s is wrong. The (formatting) precision is denoted *P* (upper case, italic); the format of the (decimal floating) type is denoted by *p* (lower case, italic). | | In the a,A bullet, change the capitalization and italics of the p’s in the text beginning with, “If the precision p is present” to get “If the precision *P* is present (in the conversion specification) and is zero or at least as large as the precision *p* (5.2.4.2.2) of the decimal floating type, the conversion is as if the precision were missing. If the precision *P* is present (and nonzero) and less than the precision *p* of the decimal floating type …”. | |  | |
| GB-248 | |  | | 7.31.6.3.2 | | 4 | | ed | | The formatting of the return values (size\_t)(-2) and (size\_t)(-1) is inconsistent. | | Either use a fixed-width font for the whole of the first return value, or variable-width (with a minus sign, not a hyphen) for the operand of the cast in the second return value; a fixed-width font would probably be most appropriate in all such cases. In either case, it would be desirable to ensure consistency with the return values of all other functions with similar lists of return values, and individual return values (e.g. wcrtomb and mbsrtowcs show similar inconsistency; see also uchar.h functions). | |  | |
| FR-249 | |  | | 7.33.6 and 7.33.14 | |  | | te | | Add features to future library directions | | see accompanying document | |  | |
| US 65  -250 | |  | | A 4.2 | |  | | ed | | Page 468: A.4.2: "n-wchar-sequence:" should be shifted left. | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| US 66  -251 | |  | | A 5.2 | |  | | ed | | Page 469: A.5.2: "d-wchar-sequence:" should be shifted left. | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-252 | |  | | A.2.2 | |  | | ed | | A stray string “[-2ex]” appears after the struct-or-union syntax. A stray string “[-7ex]” appears after the function-specifier syntax. | | Remove those strings. | |  | |
| GB-253 | |  | | A.2.3 | |  | | ed | | Stray strings “[-6ex]” appear in four places. | | Remove those strings. | |  | |
| US 1  -254 | |  | | Abstract  Pg vi | |  | | ed | | Page vi: "to from" in "Missing macros were added to from <float.h> and <limits.h>." | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-255 | |  | | Annex B | |  | | ed | | None of the subclauses list the “\_\_STDC\_VERSION\_\*\_H\_\_” macros. | | Add those macros to the library summary for each header that has such a macro. | |  | |
| FR-256 | |  | | Annex B.29 | |  | |  | | The type char8\_t is missing in the list. | | Add it | |  | |
| GB-257 | |  | | Annex E | | 2 | | ed | | Since BOOL\_WIDTH is an exact value, this should be noted here. | | Add a comment “// exact value” to the definition of BOOL\_WIDTH. | |  | |
| FR-258 | |  | | Annex M | |  | | ed | | missing mention of new interface timegm | | add to the same line as for timespec\_getres | |  | |
| GB-259 | |  | | Annexes C, E, I | |  | | ed | | In Annexes not divided into clauses, the page footer shows a clause number from the previous Annex (for example, B.31 on page 498 for Annex C). | | Fix the footers. | |  | |
| GB-260 | |  | | B.16 | |  | | ed | | The stdatomic.h summary is missing ATOMIC\_CHAR8\_T\_LOCK\_FREE and atomic\_char8\_t. | | Add those to the summary. | |  | |
| GB-261 | |  | | B.19 | |  | | ed | | The stddef.h summary is missing the unreachable macro. | | Add that to the summary. | |  | |
| GB-262 | |  | | B.22 | |  | | ed | | The stdlib.h summary is missing once\_flag and call\_once. | | Add those to the summary. | |  | |
| GB-263 | |  | | B.5 | |  | | ed | | The fenv.h summary is missing femode\_t, FE\_DFL\_MODE and FE\_TONEARESTFROMZERO. | | Add those to the summary. | |  | |
| GB-264 | |  | | B.9 | |  | | ed | | The limits.h summary is missing BITINT\_MAXWIDTH. | | Add that to the summary. | |  | |
| GB-265 | |  | | Bibliography | |  | | ed | | The entries for IEEE standards are inconsistent about use of a hyphen or an en dash to separate the standard number and date. | | Make items 9, 10 and 11 consistent in whatever way is considered appropriate. (The standards themselves appear to use a hyphen, so indicating changing en dash to hyphen in items 9 and 10.) | |  | |
| US 89  -266 | |  | | Bibliography | |  | | ed | | Page 672+: The footer on the Index is "Bibliography". | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-267 | |  | | F.10.1.1, F.10.1.8, F.10.3.11, F.10.5.3 | | 1 | | ed | | The return value +0 for acos(1), acospi(1), log(1), lgamma(1) and lgamma(2) is in a different font from other such return values. | | Change “+0” to “$+0$” in the LaTeX source, in each of those five places. | |  | |
| GB-268 | |  | | F.10.3.7 | | 4 | | te | | The example implementation of frexp is not valid because it would raise spurious “invalid” exceptions for infinities and quiet NaNs. | | Change “value == 0” to “value == 0 || !isfinite(value)”. | |  | |
| GB-269 | |  | | F.10.4.2 | | 1 | | ed | | Although not strictly required (it follows from the general semantics), consistency with the descriptions of other functions would indicate stating explicitly the values of compoundn with first argument a positive infinity and second nonzero. | | Add two more bullet points: “compoundn(+∞, n) returns +∞ for n > 0.” and “compoundn(+∞, n)” returns +0 for n < 0.”. | |  | |
| US 68  -270 | |  | | F.10.4.7 | |  | | te | | Page 524: F.10.4.7 (powr): Add to extra "powr(+1, y)": returns a NaN and raises the "invalid" floating-point exception for infinite y. | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-271 | |  | | F.10.4.7 | | 1 | | ed | | The second powr(+1, y) bullet point is incomplete. | | Change that bullet point to “powr(+1, ±∞) returns a NaN and raises the “invalid” floating-point exception.”. | |  | |
| US 69  -272 | |  | | F.10.6.8 | |  | | ed | | Page 527: F.10.6.8 (roundeven): Paragraph 1 is on the wrong line. | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-273 | |  | | F.10.6.8 | | 1 | | ed | | There is excess white space at the top of this paragraph before the bullet points start. | | Remove that white space. | |  | |
| US 70  -274 | |  | | F.10.7.1 | |  | | ed | | Page 528: F.10.7.1 (fmod): "returns x for x finite x." -> "returns x for x finite." | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| US 71  -275 | |  | | F.10.8.1 | |  | | ed | | Page 528: F.10.8.1 (copysign): "specified in the Appendix to IEC 60559." -> "specified in IEC 60559." | | Either do the change or delete the entire line. | |  | |
| GB-276 | |  | | F.10.8.7 | | footnote 455 | | ed | | The second sentence in the footnote seems incorrect in that context; that expression seems like a valid implementation even in the FE\_SNANS\_ALWAYS\_SIGNAL case. That incorrect sentence was in the initial integration of TS 18661-1, and appears to be a misapplication of the footnote that N2314 said was to be added to C11 footnote 374 (on the fmax sample implementation; footnote 456 in the CD, i.e. the next footnote). | | Move this sentence to the end of the next footnote, on fmax. | |  | |
| GB-277 | |  | | F.10.8.7 | | footnote 455 | | ed | | The expression “\*x \* 1e0” should be in a fixed-width font, as a C source expression. | | Change it into a fixed-width font. | |  | |
| US 67  -278 | |  | | F.3 | |  | | ed | | Page 508: F.3#8: "854" -> "854-1987". | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-279 | |  | | F.8.4, F.8.5 | |  | | te | | What evaluation rules apply to initializers for objects with automatic storage duration declared with the constexpr storage-class specifier? Are the same rules applied as for static and thread storage duration (so allowing such an initializer that might raise exceptions if evaluated at execution time, e.g. “constexpr double x = (double)(1.0 / 3.0);”? Or are execution-time evaluation rules applied, resulting in a constraint violation because (double)(1.0 / 3.0) is not considered to evaluate to a constant? And does this differ depending on whether the FENV\_ACCESS pragma is in effect, the FENV\_ROUND or FENV\_DEC\_ROUND pragma is in effect, or both? | | Make some suitable change (both to normative text, if any such initializers are to be accepted that would raise exceptions at execution time, and adding examples, in any case) to clarify the handling of such initializers, following any recommendations from the C floating-point group. | |  | |
| US 72  -280 | |  | | G.6.1.1 | |  | | ed | | Page 539: G.6.1.1 (cacos): Replace "pi" with pi symbol. | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-281 | |  | | G.6.1.1 | | 1 | | ed | | In the sixth bullet point, “pi” should be “π”. | | Change “pi” to “π”. | |  | |
| GB-282 | |  | | G.6.2.6 | | 1 | | ed | | In the seventh bullet point, “sin” should not be in italics. | | Change it out of italics (use \sin in LaTeX). | |  | |
| US 75  -283 | |  | | H.11 | |  | | ed | | Page 556: H.11#6: Bullet list has two items too far to the left. | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| US 76  -284 | |  | | H.11 | | paragraph 6 | | ed | | In the first instance of \_DecimalN, the “N” is not italicized per usual style. | | In the first instance of \_DecimalN italicize the “N”. | |  | |
| GB-285 | |  | | H.11.2 | | 1 | | ed | | “sublause” should be “subclause”. | | Change “sublause” to “subclause”. | |  | |
| GB-286 | |  | | H.12.2 | | 2 | | ed | | Since there are no wide string analogs of the new functions being added (no wcstofN analogous to strtofN, for example), a footnote should be added to that effect, like that for the strfrom functions. | | Add a footnote “As with the strfromd functions, there are no wide string analogs of these functions; see 7.31.4.1.1. (Alternatively, such wide string analogs could be added.) | |  | |
| GB-287 | |  | | H.12.2 | | 3 | | te | | The semantics described in this paragraph, for strtodN accepting hexadecimal floating constants, are incompatible with the base standard: an implementation following those semantics would not conform to the base standard, in that there are function inputs that have defined behavior both with the base standard and with the Annex in effect, but where the results required to be returned are different in the two cases. This is contrary to the normal practice, where optional Annexes only define things that were undefined or unspecified behavior without the Annex or that in the absence of the Annex would be rejected as a syntax violation. The example “0x1.8p+4” given in 7.24.1.6 illustrates this issue: the semantics required by the normative text in 7.24.1.6 are as shown for the example, whereas H.12.2 paragraph 3 would require it to be accepted in its entirety and converted to the number 24 (quantum exponent unclear, but should be 0, as per the previous comment). | | This might sensibly be addressed by requiring the support for hexadecimal floating constants unconditionally in strtodN, with the relevant text (including anything relating to quantum exponent, and an update to the Example) moved to 7.24.1.6. However this is addressed, the C floating-point group should propose appropriate wording for the exact changes. | |  | |
| GB-288 | |  | | H.12.2 | | 3 | | te | | The quantum exponent resulting from conversion of hexadecimal numbers is not clearly specified. | | Specify a preferred quantum exponent of 0 if the hexadecimal number can be exactly represented in the decimal type and the least possible if not, as for conversions from non-decimal floating types, in order to meet IEC 60559 requirements; the C floating-point group should propose appropriate wording for this. | |  | |
| US 77  -289 | |  | | H.12.3.1 | | heading | | ed | | There is a space before the “N” in the function name. | | Remove the space between “strfromencf” and “N”. | |  | |
| US 80  -290 | |  | | H.12.4 | | paragraph 1 | | ed | | The “N” in “for each N” is not italicized per usual practice. | | Italicize the “N” in “for each N”. | |  | |
| US 81  -291 | |  | | H.13 | |  | | ed | | Page 576: H.13: Type-generic macros <tgmath.h>: last table: “fmul(dc, d)” Move first "undefined" down one line. | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-292 | |  | | H.13 | | 6 | | ed | | The example for f32xsqrt(n) should call f32xsqrtf64x: n is an int, which is mapped to double, there is no corresponding function whose parameters have type exactly double, so \_FloatNx types are preferred, and \_Float64x exists (as remarked for a subsequent example), and can represent all double values (since those are the same as \_Float64 values). | | In that example, change “f32xsqrtf64” to “f32xsqrtf64x”. | |  | |
| GB-293 | |  | | H.13 | | 6 | | ed | | The table of examples for “Macros that round the result to a narrower type” includes an entry “undefined” where the final “d” is in a different font | | Change that “d” to the same font as the rest of the word. | |  | |
| GB-294 | |  | | H.13 | | 6 | | ed | | In the first table of example, “pow32x” should be “powf32x”. | | Change “pow32x” to “powf32x”. | |  | |
| US 82  -295 | |  | | H.13 | | paragraph 5 | | ed | | In the second level bullet beginning “For prefix f:”, the first line has a bad line break. | | In the second level bullet beginning “For prefix f:”, fix the bad line break at the end of the first line. | |  | |
| US 83  -296 | |  | | H.13 | | paragraph 6 | | ed | | In the text right after the program text box, the first line has a bad line break. | | In the text right after the program text box, fix the bad line break at the end of the first line. | |  | |
| US 84  -297 | |  | | H.13 | | paragraph 6, table following “Macros that round the result to a narrower type…” | | ed | | In the second column, “undefined” should be in the same row as “fmul(dc, d)” in the first column. | | In the second column, lower “undefined” to be in the same row as “fmul(dc, d)” in the first column. | |  | |
| GB-298 | |  | | H.2, H.2.1 | |  | | ed | | The initial paragraphs in these subclauses should be numbered. | | Add paragraph numbers (adjusting numbers of subsequent paragraphs accordingly). | |  | |
| GB-299 | |  | | H.2.1 | | 1 | | ed | | “encodng” should be “encoding”. | | Change “encodng” to “encoding”. | |  | |
| GB-300 | |  | | H.2.1, H.2.3, H.3 | |  | | ed | | In the tables in H2.1., and the Example below (paragraph 1), and in H.2.3 paragraph 1, and in H.3 paragraph 4 and paragraph 7 (MIN\_EXP, MIN\_10\_EXP, MAX\_EXP, MAX\_10\_EXP, MAX, MIN, TRUE\_MIN descriptions), “max” and “min” should not be in italics. | | Change them out of italics throughout the places listed (use LaTeX \max and \min). | |  | |
| US 85  -301 | |  | | H.2.2 | | paragraph 2 | | ed | | Bad line break in second line. | | Fix the bad line break in para 2, second line. | |  | |
| GB-302 | |  | | H.2.4 | | 5 | | ed | | “sublcause” should be “subclause”. | | Change “sublcause” to “subclause”. | |  | |
| US 74  -303 | |  | | H.3 | |  | | ed | | Page 550: H.3: \*\_EPSILON: Add "normalized" before "value". | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| US 73  -304 | |  | | H.3 | |  | | te | | Page 550: H.3: \*\_MAX\_EXP: "maximum negative integer" -> "maximum positive integer" | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-305 | |  | | Index | |  | | ed | | “\_DECIMAL\_DIG”, “\_H\_\_”, “\_r” should not be indexed under “identifier prefix” (they are suffixes not prefixes). “\_\_\_Noreturn\_\_” should not be indexed under “pragma” (it’s an attribute). | | Relocate those index entries to more appropriate categories. | |  | |
| GB-306 | |  | | Index | |  | | ed | | The index has many entries that are inappropriate, or inappropriately described, as identified for J.6.1 and J.6.2 above. For example, the entry for pp\_param (local notation in an example, not a pragma), generic\_\*\_type (not types, just local notation), frompfp (no such functions), and fragments of function and macro names (could be indexed if described appropriately as function or macro families, but not when described as functions or macros). Likewise for some uppercase variants such as ADDD, ADDF, XADDD, XADDF (macro name fragments). | | Remove or change descriptions for all entries described as problematic for J.6.1 and J.6.2 above. | |  | |
| US 93  -307 | |  | | Index | |  | | ed | | Page 711: The following are missing from index for "identifier suffix":  \_DECIMAL\_DIG  \_DIG  \_EPSILON  \_MANT\_DIG  \_MAX\_10\_EXP  \_MAX\_EXP  \_MIN\_10\_EXP  \_MIN\_EXP  \_SNAN  \_TRUE\_MIN | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| US 92  -308 | |  | | Index | |  | | ed | | Page 601: The entry "\_EXT\_\_" should be removed. | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| US 91  -309 | |  | | Index | |  | | ed | | Page 693: There are 2 index entries for "IEC 60559". I assume that they should be merged together. | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| US 90  -310 | |  | | Index | |  | | ed | | Pages 674 & 710: There are several suffix terms in the index that are listed as prefix terms:  \_DECIMAL\_DIG identifier prefix,  \_H\_\_ identifier prefix,  \_r identifier prefix, | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-311 | |  | | Introduction | | 7 | | ed | | [www.open-std.org](http://www.open-std.org/) now uses https so the link should be updated accordingly. Also, the Rationale and the log of Defect Reports may not be the best examples of additional information present there, since the Rationale hasn’t been updated for C11 or later standard revisions and reported issues are no longer referred to as Defect Reports because they do not generally meet current ISO requirements for Defect Reports. | | Change http to https. Make other changes to the examples of additional information based on discussion of what might be the best examples to give. | |  | |
| FR-312 | |  | | Introduction | | p7 | | ed | | There is no related “Rationale” document | | remove reference to such a document from the text | |  | |
| GB-313 | |  | | J.1 | | 1 | | ed | | Item 56 has a stray “c” before “7.31.6.4.1”. | | Replace that “c” by a space. | |  | |
| GB-314 | |  | | J.2 | | 1 | | ed | | Item 54 does not reflect changes to the definition of constant expressions. | | Change “enumeration constants” to “named and compound literal constants of arithmetic type”. | |  | |
| GB-315 | |  | | J.2 | | 1 | | ed | | Item 53 does not reflect changes to the definition of constant expressions. | | Insert “a named constant, a compound literal constant,” at the start of the list. | |  | |
| GB-316 | |  | | J.2 | | 1 | | ed | | Item 52 does not reflect changes to the definition of constant expressions. | | Change “enumeration constants” to “named and compound literal constants of integer type”. Change “floating constants” to “floating, named, and compound literal constants of floating type”. | |  | |
| GB-317 | |  | | J.3.12 | | 1 | | ed | | Item 43 contains a stray “)”. | | Remove that “)”. | |  | |
| US 86  -318 | |  | | J.6.1 | |  | | ed | | Pages 604, 606, 675, 686: What are "\_\_suffix\_\_" and "\_\_prefix\_\_" that only appear in J.6.1 Rule based identifiers and the index? (page 168 uses "prefix" and "suffix"). | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-319 | |  | | J.6.1 | | 2 | | ed | | \_EXT\_\_, \_Float, FP\_FAST\_D, FP\_FAST\_F, stdc\_, strfromencbind, strfromencdecd, strfromencf, strto, strtoencbind, strtoencdecd, strtoencf are fragments of identifiers or keywords (of no significance by themselves) and should not appear in this list. | | Remove those entries from this list. | |  | |
| GB-320 | |  | | J.6.1 | | 2 | | ed | | \_\_alignas\_is\_defined and \_\_alignof\_is\_defined were removed so should not appear in this list. | | Remove \_\_alignas\_is\_defined and \_\_alignof\_is\_defined from this list. | |  | |
| GB-321 | |  | | J.6.1, J.6.2 | |  | | ed | | \_\_pp\_param\_\_ and pp\_param are an example in text, not identifiers with any special significance, and should not appear in these lists. | | Remove \_\_pp\_param\_\_ and pp\_param from these lists. | |  | |
| GB-322 | |  | | J.6.2 | | 1 | | ed | | The names reserved for complex.h by 7.33.1 are only listed in their “double” forms, not the corresponding float and long double identifiers. | | Add the corresponding “float” and “long double” identifiers to the list. | |  | |
| GB-323 | |  | | J.6.2 | | 1 | | ed | | generic\_count\_type, generic\_return\_type, generic\_value type, QChar, QVoid and QWchar\_t are notation used for some type-generic function declarations and have no special significance as identifiers. | | Remove those entries from the list. | |  | |
| GB-324 | |  | | J.6.2 | | 1 | | ed | | The eight entries starting “frompfp” (note the first ‘p’) are entirely spurious; there are no such standard identifiers. | | Remove those entries from the list. | |  | |
| GB-325 | |  | | J.6.2 | | 1 | | ed | | acosd, acoshd, acospid, addd, addf, asind, asinhd, asinpid, atan2d, atan2pid, atand, atanhd, atanpid, canonicalized, cbrtd, ceild, compoundnd, copysignd, cosd, coshd, cospid, DEC, DECN\_, DECN, decodebin, decodebind, decodedec, decodedecd, decodef, divd, divf, encbind, encdecd, encf, encodebin, encodebind, encodedec, encodedecd, encodef, erfcd, erfd, exp10d, exp10m1d, exp2d, exp2m1d, expd, expm1d, fabsd, fdimd, Float, floord, FLT, FLTN\_, FLTN, fmad, fmaxd, fmaximumd, fmaximum\_magd, fmaximum\_mag\_numd, fmaximum\_numd, fmind, fminimumd, fminimum\_magd, fminimum\_mag\_numd, fminimum\_numd, fmodd, frexpd, fromfpd, fromfpxd, getpayloadd, HUGE\_VAL\_D, HUGE\_VAL\_F, hypotd, ilogbd, ldexpd, lgammad, llogbd, llquantexpd, llrintd, llroundd, log10d, log10p1d, log2d, log2p1d, logbd, logd, log1pd, lrintd, lroundd, modfd, muld, mulf, N, nand, nearbyintd, nextafterd, nextdownd, nextupd, powd, pownd, powrd, quantized, quantumd, remainderd, rintd, rootnd, roundd, roundevend, rsqrtd, samequantumd, scalblnd, scalbnd, sind, sinhd, sinpid, sqrtd, subd, subf, tand, tanhd, tgammad, truncd, ufromfpd, ufromfpxd, X\_, xaddd, xaddf, xdivd, xdivf, xfmad, xfmaf, xmuld, xmulf, xsqrtd, xsqrtf, xsubd, xsubf are fragments of identifiers (of no significance by themselves) and should not appear in this list. | | Remove those entries from the list (without affecting their formatting elsewhere in the document). | |  | |
| IE-326 | |  | | K.3.5.2.1 | | 7 | | te | | Adopt changes from N3059 <https://www.open-std.org/jtc1/sc22/WG14/www/docs/n3059.pdf> | | From N3059 Section 1.3: ~~To the extent that the underlying system supports the concepts, files opened for writing shall be opened with exclusive (also known as non-shared) access. If the file is being created, and the first 2 character of the mode string is not ’u’ , to the extent that the underlying system supports it, the file shall have a file permission that prevents other users on the system from accessing the file. If the file is being created and first character of the mode string is ’u’ , then by the time the file has been closed, it shall have the system default file access permissions.~~ | |  | |
| US 87  -327 | |  | | K.3.5.2.1 | | 7 | | te | | Adopt changes from N3059 <https://www.open-std.org/jtc1/sc22/WG14/www/docs/n3059.pdf> | | From N3059 Section 1.3: ~~To the extent that the underlying system supports the concepts, files opened for writing shall be opened with exclusive (also known as non-shared) access. If the file is being created, and the first 2 character of the mode string is not ’u’ , to the extent that the underlying system supports it, the file shall have a file permission that prevents other users on the system from accessing the file. If the file is being created and first character of the mode string is ’u’ , then by the time the file has been closed, it shall have the system default file access permissions.~~ | |  | |
| US 88  -328 | |  | | K.3.9.2.2 | | paragraph 3 | | ed | | Bad line break and inconsistent use of space in front of operator parentheses. | | Remove space after “sizeof”, which should fix the bad line break there. | |  | |
| GB-329 | |  | | L.3 | | 2 | | ed | | In the penultimate bullet point, “free” and “realloc” should be in fixed-width type. | | Change “free” and “realloc” to fixed-width type. | |  | |
| GB-330 | |  | | M.1 | | 1 | | ed | | The ptrdiff\_t entry has a wording issue. | | Change “of at least 16” to “of width at least 16”. | |  | |
| GB-331 | |  | | M.1 | | 1 | | ed | | The stdbit.h entry has some wording issues. | | Change “implementation’s” to “implementations’” (moving apostrophe after ‘s’). Change “efficiency” to “efficient”. | |  | |
| GB-332 | |  | | M.1 | | 1 | | ed | | The \_BitInt entry is badly worded. | | Change “added the \_BitInt the bit-precise integer types” to “added bit-precise integer types \_BitInt(N) and unsigned \_BitInt(N)”. | |  | |
| NEN/NL9-333 | | n/a | | n/a | | n/a | | te | | N3031 failed gain consensus at the last WG14 meeting not due to fundamental disagreements about the necessity of the functionality being introduced in that paper, but due to minor details (naming of the functionality, potential about potential performance, and the nuances of initializing and handling the “mbstate\_t” object). However, the situation as outlined in the motivation of that paper, paired with the ubiquity and centrality of C, means that correctly processing text is fragile and complex.  Especially when it comes to human names, this is fundamentally at odds with legislation that emphasizes the right of people to have their name represented correctly (e.g. GDPR; also confirmed in court recently); EU citizens with non-ASCII characters in their birth names are affected by this in a variety of services, from banking to airline tickets to online purchases. In addition to the motivation outlined in N3031, the immensely widespread occurrence of mojibake upon entering non-ASCII characters is testament to the fact this is not merely a problem of inadequately skilled programmers and programmers, but a fundamental shortcoming of the current standardization.    We also fundamentally believe the functionality would have made it into C23 at the previous meeting if the author had more time to address the wording needs at an earlier date (the changes requested came in some 2 days before the author was meant to tweak). We do not blame anyone for this (everyone is busy), but a fresh look after the comments were addressed in post may help. | | Consider taking a poll to adopt N3031 (or a suitably modified version or successor revision) for C23. Particularly, the functionality as it relates to <stdmchar.h> header and it’s encoding translation functionality. | |  | |
| FR-334 | |  | | overall | |  | | ed | | There are mis-classifications of identifiers | | see accompanying document | |  | |
| FR-335 | |  | | overall | |  | | te | | Diverging policies for version numbering | | see accompanying document | |  | |
| FR-336 | |  | | overall | |  | | ed | | Colors for code snippets | | see accompanying document | |  | |
| US 2  -337 | |  | | Pg vii | |  | | ed | | Page vii: "heads" in "The wording for freestanding heads for <string.h>" | | Please make any proposed change as necessary. for each of these issues (unless noted otherwise) | |  | |
| GB-338 | |  | | throughout | |  | | ed | | The formatting of footnote references is inconsistent throughout the draft; sometimes the footnote number appears before punctuation, sometimes after, sometimes preceded by a space, sometimes not. | | Change to follow usual practice consistently: no space before the footnote number, and, if adjacent to punctuation, coming after rather than before that punctuation. | |  | |

ISO\_IEC CD 9899\_AFNOR.doc: Collation successful

ISO\_IEC CD 9899\_ANSI.docx: Collation successful

ISO\_IEC CD 9899\_BSI.doc: Collation successful

ISO\_IEC CD 9899\_DIN.docx: Collation successful

ISO\_IEC CD 9899\_NEN.docx: Collation successful

ISO\_IEC CD 9899\_NSAI.docx: Collation successful

ISO\_IEC CD 9899\_SCC.doc: Collation successful

ISO\_IEC CD 9899\_SIS.docx: Collation successful

Collation of files was successful. Number of collated files: 8

SELECTED (number of files): 8

PASSED TEST (number of files conformed to CCT table model): 8

FAILED TEST (number of files conformed to CCT table model): 0

CCT - Version 2020.1