ISO/IEC/JTC 1/SC 22/WG 23 N0849

7 December 2018

Collated comments on TR 24772-3 (N0837) with proposed resolution

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PL 001 | |  | | 03.01 | |  | | ge | | Rather than copy-pasting whole section from the C standard maybe it's just better to reference it | |  | | It’s a point of view, but in general I prefer to have all the information in a single document. I’m not inclined to make any changes | |
| PL 002 | |  | | 04 | |  | | te | | Actually according to the C standard there is difference in the semantics:  (based on N1570) 6.2.5 Types: 20. A pointer type describes an object whose value provides a reference to an entity of the referenced type.   It means. That pointer is a type, while reference is the pointer's value property | | “Unlike some other languages, in C the terms ‘pass by reference’, ‘pass by pointer’, ‘pass by address’ have the same meaning” | | I think the observation is debatable, but I have no objection to adopting the proposed change | |
| PL 003 | |  | | 05 | | Table, point 8 | | te | | should actually also mention, that it can be intended to wrap-around the result, which is also fine and this is why you can do it on unsigned integers without UB | | Check, that the result of an operation on an unsigned integer value will not cause wrapping unless it can be shown that wrapping cannot occur or it’s intended behavior. | | I’m inclined to reject this, as there’s no way to determine whether the programmer expected the wrap-around to occur or not. Always avoiding wrap-around is a safer option | |
| PL 004 | |  | | 06.02.2 | | Point 2 | | te | | Being aware of the promotion/conversion rules won't prevent from actually doing mistakes with them as they are easy to be skipped. | | “Enable all compiler warnings regarding implicit conversions or use static analysis tools to show the warning” | | I’d be happy with the suggested text being added after that already there | |
| PL 005 | | 1 | | 06.03.1 | |  | | ed | | "C supports a variety of sized for integers such as ... . Each may either be signed or unsigned".  As to my understanding this means, that it can be signed and unsigned as well. | | C supports a variety of sized for signed integers ... Each signed integer has it's unsigned counterpart. | | I don’t see that this makes any difference to the meaning of the para. I’m inclined to reject (PS the text says ‘sizes’ not ‘sized’).  Maybe ‘Each integer type may be signed or unsigned’ in case its felt the sentence may imply an integer value can change signedness | |
| PL 006 | | 2 | | 06.08.1 | |  | | ed | | “unspecified” | | “undefined” | | Agreed (embarrassed that that slipped through!) | |
| PL 007 | |  | | 06.12.02 | | Point 2 | | te | | 1. To all my knowledge the pointer arithmetic is there actually to prevent you from doing the calculation error. 2. index operator do just the same thing as we can manually with pointer arithmetic. | | Remove the point | | NO!! Pointer arithmetic is a known source of programmer errors. Yes, internally indexing uses the same mechanism, but the expression of intent is far clearer | |
| PL 008 | |  | | 06.14.02 | |  | | te | | If the document recommends to malloc with the macro that does automatic casts, then most probably it should recommend freeing memory with macro, that automatically sets the pointer to the null value | | Use macro to free and set pointer to NULL example macro definition:  #define freeAndNull(P) free((P));(P)=NULL; (or any similar macro) | | I can see the logic of this, but I’m a bit worried about the implications of macro’ing in a compound statement. I have a nasty feeling there may be some contexts where it either won’t work or the results may be other than expected. I’m inclined to reject, or add to bullet 2 ‘A macro could be used to ensure that pointer freeing and setting to NULL are always performed together’ without providing actual code | |
| PL 009 | |  | | 06.26.02 | |  | | te | | using // instead of /\*\*/ - I do not see how this can help. Especially with the given example. With /\*\*/ You exactly see where the comment ends + even the simplest editors colors the comments differently than code, so it’s hard to miss that | | Remove the guidance | | I wouldn’t argue too hard to retain this advice. I’m inclined to agree | |
| PL 010 | |  | | 06.29.02 | |  | | te | | Do not modify a loop control variable within a loop.  Usefulness of this holds true for "for" loops. This guidance is more than misleading in case of while or do while loops. | | Do not modify a loop control variable within a “for” loop | | Agreed | |
| CA 011 | |  | | 06.32.02 | |  | | ed | | Typo | | "Follow the g guidance contained in ...."  ->  "Follow the guidance contained in ...." | | The error is in 6.36.2 (not 6.32.2) – Yes fix it | |
| PL 012 | |  | | 06.39.02 | |  | | te | | If I understand correctly paper proposes to use garbage collectors. This is in my opinion wrong, as a) this is not supported by the C language standard (language level). Even if some compiler/lib supports some kind of garbage collecting for C it will be not standard C code. I am not sure it’s good idea to recommend non standard approaches towards coding guidelines. Instead document should propose how to use language features in such a way, that it’s safe b) using garbage collector will cause other kinds of vulnerabilities especially with real-time applications (as it's noticed in the document) | | Remove the guidance | | I’m not particularly wedded to this guidance and wouldn’t object to its removal.  Alternatively we could weaken it, say ‘Consider the use of any garbage collector that may be available to replace the …; | |
| PL 013 | | 1 | | 06.46.01 | |  | | te | | Parameter passing is either pass by reference or pass by value. This is wrong. It's always pass by value. | | Remove first sentence in 6.46.1 | | I disagree that parameters are always passed by value. That may be the mechanism, but the value may be a pointer – see 6.2.5#20 of C17 “*A pointer type describes an object whose value provides a reference to an entity of the referenced type*”  We could modify the first sentence to ‘*A parameter in C is either a value or a pointer to a value*.’ | |
| PL 014 | |  | | 06.51.02 | |  | | te | | I would add one guidance in here. Apply coding standards so that macros are easily differentiable from the inline function | | → Use coding guidelines to make function-like macros differentiable from inline functions e.g. function-like macros should be named with capital letters | | I’d suggest adding a bullet after the second ‘Apply a naming convention to distinguish between inline functions and function-like macros’ | |
| PL 015 | |  | | 06.54 | |  | | ed | | The section is too generic. | | Leaving only first point from 6.54.2 would suffice | | I’d argue for leaving this as is – but would be willing to remove it if required | |
| CA 016 | |  | | 06.56.02 | |  | | ed | | Typo | | "Follow the g guidance contained in ...."  ->  "Follow the guidance contained in ...." | | Yes, fix it | |

ISO\_IEC PDTR 24772-3 - JTC001-SC22-N5312\_PKN.doc: Collation successful

ISO\_IEC PDTR 24772-3 - JTC001-SC22-N5312\_SCC.doc: Collation successful

Collation of files was successful. Number of collated files: 2

SELECTED (number of files): 2

PASSED TEST (number of files conformed to CCT table model): 2

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

CCT - Version 2018.2