|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PL 001 |  | 03.01 |  | | ge | Rather than copy-pasting whole section from the C standard maybe it's just better to reference it |  | Reject  It was a committee decision that it was beneficial to have relevant definitions included in this document |
| 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” | Accept |
| 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. | Accept in principle  Changing the addition to “… or document and verify the intended behaviour” |
| 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” | Accept  Make a new bullet:  “Enable compiler warnings regarding implicit conversions or use static analysis tools that provide such warnings” |
| 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. | Accept in principle  Clarified wording by making sure that the text refers to types rather than individual objects |
| PL 006 | 2 | 06.08.1 |  | ed | | “unspecified” | “undefined” | Accept |
| 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 | Reject  Pointer arithmetic is a known source of program errors. While indexing uses the same mechanisms, the intent is clearer – which is why it says ‘consider a ban’ rather than ‘prohibit’ |
| 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) | Accept in principle  Replace the start of bullet 2 with ‘Use a macro to call free and set the pointer to NULL” |
| 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 | Accept |
| 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 | Accept |
| CA 011 |  | 06.32.02 |  | | ed | Typo | "Follow the g guidance contained in ...."  ->  "Follow the guidance contained in ...." | Accept – but the error is in 6.36.2 (not 6.32.2) |
| 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 | Accept |
| 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 | Accept  Additionally change the start of the sentence to ‘There isn’t a guarantee that the parameters being passed to a function will …’ Also wording added that unexpected NULL pointers are a particular problem |
| 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 | Accept in principle  Text added to the third bullet ‘and use a naming convention that clearly identifies it as a macro” |
| PL 015 |  | 06.54 |  | ed | | The section is too generic. | Leaving only first point from 6.54.2 would suffice | Accept |
| CA 016 |  | 06.56.02 |  | | ed | Typo | "Follow the g guidance contained in ...."  ->  "Follow the guidance contained in ...." | Accept |