Joseph Silva Jr.

07/23/21

CS 350 – Module 4 Journal

SNHU

1. **Describe best coding practices in Embedded C (1):**

* Know why you're using it – know why you are using a specific symbol of a code during writing. Ex: = or == during an if statement.
* Choose the best coding standard for your industry – “It reduces the amount of qualification work to satisfy your end users”.
* Use coding rules and follow recommendations – Rules are coding compliance and recommendations, which do not have to be followed all the time, can help keep a program run smooth.
* Describe the intent behind the rule – “The description elaborates on the headline and provides suitable context”.
* Update coding standards with care – Know when to update the coding and when not to because updating code standards might not always benefit the program for a long-term affect.
* Consider open vs. closed standards – “Open standards tend to be extremely dynamic. Guidelines change quickly. Closed standards, however, are better for safety-critical industries. They provide a stable reference to meet mandatory requirements”.
* Prioritize coding rules – Allows a programmer to go after fixing major problems when comparing a program’s code to common code standards to prioritize bug fixes and keep code quality up to par.
* Plan for rule deviations – “Having a plan for rule deviations will help you if you need to provide evidence to an auditor”.
* Educate programmers – gives the programmers the ability to understand and learn more about additional information or additional knowledge when developing programs.

1. **Describe common pitfalls in Embedded C (2):**

* [Mixing signed and unsigned integers in arithmetic operations](https://nerdyelectronics.com/embedded-systems/embedded-c/most-common-pitfalls-in-c/#1_mixing_signed_and_unsigned_integers_in_arithmetic_operations) – can have an output of the wrong results.
* [Overstepping Array Boundaries](https://nerdyelectronics.com/embedded-systems/embedded-c/most-common-pitfalls-in-c/#2_overstepping_array_boundaries) – Know the array length while working on arrays or else a corrupting the buffer or segmentation fault can occur.
* [Missing out the Base Condition in Recursive Function](https://nerdyelectronics.com/embedded-systems/embedded-c/most-common-pitfalls-in-c/#3_missing_out_the_base_condition_in_recursive_function) – “The problem with this function is it would loop infinitely, causing a segmentation fault it needs a base condition to stop the recursion”.
* [Using character constants instead of string literals, and vice versa](https://nerdyelectronics.com/embedded-systems/embedded-c/most-common-pitfalls-in-c/#4_using_character_constants_instead_of_string_literals_and_vice_versa) – there are two different things a character constant (‘ ‘) is an integer whose value is the character code while a string literal (“ “) is an unmodifiable array.
* [Floating point literals are of type double by default](https://nerdyelectronics.com/embedded-systems/embedded-c/most-common-pitfalls-in-c/#5_floating_point_literals_are_of_type_double_by_default) – “Besides rounding errors, mixing float variables with double literals will result in poor performance on platforms which don’t have hardware support for double precision”.
* [Forgetting to free memory](https://nerdyelectronics.com/embedded-systems/embedded-c/most-common-pitfalls-in-c/#6_forgetting_to_free_memory) -  “because when a process exits, almost all operating systems free all the allocated memory for you. Also note that getline could fail in many different ways, but in whichever way it fails, the memory it has allocated should always be freed (when you’ve finished using it) if line is not NULL. Memory can be allocated even if the first call to getline() detects EOF (which is reported by a return value of -1, not EOF)”.
* [Adding a semicolon to a #define](https://nerdyelectronics.com/embedded-systems/embedded-c/most-common-pitfalls-in-c/#7_adding_a_semicolon_to_a_define) – causes a syntax error.
* [Be careful with semicolons](https://nerdyelectronics.com/embedded-systems/embedded-c/most-common-pitfalls-in-c/#8_be_careful_with_semicolons) – missing a semicolon causes a syntax error.
* [Mistakenly writing = instead of == when comparing](https://nerdyelectronics.com/embedded-systems/embedded-c/most-common-pitfalls-in-c/#9_mistakenly_writing_instead_of_when_comparing) – know + is for assignment and == is for comparing during an if statement.
* [Copying too much](https://nerdyelectronics.com/embedded-systems/embedded-c/most-common-pitfalls-in-c/#10_copying_too_much) – undefined behavior can allow hackers to exploit these mistakes to change addresses for malicious code to be entered into the program.
* [Macros are simple string replacements](https://nerdyelectronics.com/embedded-systems/embedded-c/most-common-pitfalls-in-c/#11_macros_are_simple_string_replacements) – The macro will expand the wrong statement.

**CITATIONS**

1. Bellairs, R. (2018). *9 coding standards best practices*. Perforce. <https://www.perforce.com/blog/qac/9-coding-standards-best-practices>
2. Vivek BhageriaVivek is a Senior Embedded Engineer at Robert Bosch. He has been working on Embedded Systems for the past 10 years. He loves to share his knowledge and train those who are interested. Nerdyelectronics.com was started out of this interest.www. (2020, December 11). *Most common pitfalls in C Programming Language and how to avoid them*. NerdyElectronics. https://nerdyelectronics.com/embedded-systems/embedded-c/most-common-pitfalls-in-c/.