**4-2 Journal: Best Coding Practices**

Winnie Kwong

Southern New Hampshire University

CS-350 Emerging Sys Arch & Tech

Professor Prasad

March 31st, 2024

**Best Practices: Embedded C**

Some of the best practices in coding for embedded C are to be descriptive, use input validations, and use testing and debugging. Garcia recommends using descriptive variables and function names to make the code more readable and maintainable. (Garcia, 2024). Using descriptive variables and function names enables the developer to grasp the purpose of the variable or function, especially when the developer is self-documenting or collaborating with others. According to Soham, one of the best practices is to validate and sanitize input data before processing it. (Soham, 2024). Using input validation and sanitization is essential because it secures vulnerabilities and can simplify the code to mitigate error handling, making the system more stable and preventing crashes. Lastly, it is crucial to thoroughly test and debug to identify and address security vulnerabilities. Testing and efficient debugging can reduce the risk of attacks and determine weaknesses that unauthorized users can exploit. Using best practices such as descriptive variables and functions, input validations, and testing and debugging enables the developers to meet industry standards when coding in embedded C.

**Pitfalls: Embedded C**

Although there are many best practices in coding, there are pitfalls, such as not staying up to date with emerging technologies, mixing integers in arithmetic operations, and large if-else and case statements. Pandey states, "Failing to keep up with emerging technologies might result in out-of-date products and lost opportunities." (Pandey, 2023). Suppose a developer falls behind with emerging technologies. In this scenario, the developer becomes at fault for missing possible tools and libraries that could have made the code easier to simplify or even automate. When mixing signed and unsigned integers in arithmetic operations, incorrect comparisons can hinder the results. Bhageria argues that one of the C programming language's most common pitfalls is mixing signed and unsigned integers in arithmetic operations. (Bhageria, 2020). Suppose a developer has chosen the wrong integer. The results may accidentally be less than expected because the integer may have been converted with a negative value or cause difficulties with the overflow and underflow. Another pitfall is to include large if-then-else and case statements. The larger the if-then-else and case statements, the more difficult it will be to debug because of the complex block. It also reduces the readability and maintainability, causing a rise in error handling. By not staying up to date with emerging technologies, mixing integers in arithmetic operations, and large if-else and case statements, the developer will face many challenges that can potentially harm the performance of the code.

**References**

Bhageria, V. (2022, April 18). Most common pitfalls in C Programming Language and how to avoid them. *NerdyElectronics*. https://nerdyelectronics.com/most-common-pitfalls-in-c/

Garcia, Y. (2024, February 15). Best Practices for Embedded C Programming: Ensuring Efficiency, Reliability, and Maintainability. *LinkedIn.* <https://www.linkedin.com/pulse/best-practices-embedded-c-programming-ensuring-yamil-garcia-ibwje/>

Pandey, S. (2023, November 8). Common mistakes to avoid in embedded software development. *HashStudioz Technologies*. <https://hashstudioz.com/blog/embedded-software-development-mistakes/>

Soham. (2024, March 12). Building secure embedded C Programs: best practices and code examples. *Medium*. <https://medium.com/@juhinsohamdas/building-secure-embedded-c-programs-best-practices-and-code-examples-a11bfa69f2c0>