Code Inspection Checklist for 'C' code

1. Is the design implemented completely and correctly?
2. Are there missing or extraneous functions?
3. Is each loop executed the correct number of times?
4. Will each loop terminate?
5. Will the program terminate?
6. Are all possible loop fall-throughs correct?
7. Are all CASE statements evaluated as expected?
8. Is there any unreachable code?
9. Are there any off-by-one iteration errors?
10. Are there any dangling ELSE clauses?
11. Is pointer addressing used correctly?
12. Are priority rules and brackets in arithmetic expression evaluation used as required to achieve desired results?
13. Are boundary conditions considered? (e.g., null or negative values, adding to an empty list, etc.)
14. Are pointer parameters used as values and vice-versa?

Interfaces

15. Is the number of input parameters equal to then number of arguments?
16. Do parameter and argument attributes match?
17. Do the units of parameters and arguments match?
18. Are any input-only arguments altered?
19. Are global variable definitions consistent across modules?
20. Are any constants passed as arguments?
21. Are any functions called and never returned from?
22. Are returned VOID values used?
23. Are all interfaces correctly used as defined in the Software Design Description?

Data and Storage

24. Are data mode definitions correctly used?

25. Are data and storage areas initialized before use, correct fields accessed and/or updated?

26. Is data scope correctly established and used?

27. If identifiers with identical names exist at different procedure call levels, are they used correctly according to their local and global scope?

28. Is there unnecessary packing or mapping of data?

29. Are all pointers based on correct storage attributes?

30. Is the correct level of indirection used?

31. Are any string limits exceeded?

32. Are all variables EXPLICITLY declared?

33. Are all arrays, strings, and pointers initialized correctly?

34. Are all subscripts within bounds?

35. Are there any non-integer subscripts?

Maintainability and Testability

36. Is the code understandable (i.e., choice of variable names, use of comments, etc.)

37. Is there a module header?

38. Is there sufficient and accurate commentary to allow the reader to understand the code?

39. Does the formatting and indenting style add to the readability of the code?

40. Are coding conventions followed?

41. Is tricky or obscure logic used?

42. Is the code structured to allow for easier debugging and testing?

43. Is the code structured so that it could be easily extended for new functions?

44. Are there any unnecessary restrictions to extensions due to code structure?

Error Handling

45. Are all probable error conditions handled?

46. Are error messages and return codes used?

47. Are they meaningful and accurate?

48. Are the default branches is CASE statements handled correctly?

49. Does the code allow for recovery from error conditions?

50. Is range checking done where appropriate to isolate the source of an error?