These are some guidelines for writing good code. They’re based on Tom Gelb’s *Software Inspection*, 1993, pp. 423-427.

**Semantics** The code should implement the specification, as defined in the assignment.

**Simplicity** The code should accomplish its task (implement the semantics) as simply as possible, so it can be easily understood.

**Confined** The code should confine itself to the semantics, and not implement other functionality.

**Perform** The code should implement the specification with sufficiently efficient algorithms and techniques so that it performs fast enough. Performance requirements may include: no busy waiting, efficient memory use, and good scaling.

**Robust** The code should handle errors well. It should try to recover from system errors. Unrecoverable system errors and input errors should produce comprehensible error messages.

**Non-repetitive** Basic concepts (in code or comments) should be stated only once. Subsequent use of the concept should refer to the initial statement.

**Symbolic** The code should use symbolic constants, not hard-coded values.

**Well Commented** Comments in the code should accurately, thoroughly, clearly and concisely describe the code. If needed, they should document the code’s history and logic. Their structure and layout should meet appropriate standards.

**Layout** The code layout (indentation, spacing, etc.) should help make the code easy to understand, and follow any appropriate standard.

**Resourceful** Where suitable, the code should use data structures and algorithms that are available in the language or its libraries.

**Naming** Objects (variables, classes, methods, etc.) should be given names that ease comprehension of the code.

**Tested** The code should be well tested. Test cases should be packaged with the code.

**Usable** The code’s interface(s) (which may be function calls, command line operations, a GUI or a combination of the above) should meet the needs of the target users.

**Modular** The code should be modular, to support reuse, maintenance and future modifications.

**Paradigm** The code should employ its programming language(s) in their recommended paradigms.

**Some comments**

Guidelines **Semantics**, **Simplicity**, **Confined** and **Tested** are concerned with software correctness. If informal correctness proofs are appropriate, they should appear in the comments.

Guideline **Perform** relates to **Resourceful**, because reusing good code in libraries can help make efficient algorithms.

Guidelines **Non-repetitive** and **Symbolic** recommend non-redundant expression of ideas. Redundant ideas are bad because to keep them identical modifications must be made in multiple places. This can be difficult or impossible when the locations become widely separated.