https://inf.ethz.ch/personal/wirth/Oberon/index.html

Niklaus Wirth School of Programming Style

A Tutorial Excerpted from Niklaus Wirth Recommendations
A derivative of Programming in Modula-2 (1982)
Niklaus Wirth (rev. 5.10.2015)

Pages 20 - 21. Avoid hardwired literals scattered all over the code.

```
CONST N = 16;
EOL = 0DX;    (*WS: Note that a cryptic hex constant is given a descriptive name,*)
empty = {};
M = N-1;
```

Constants with explicit names aid in making a program readable, provided that the constants are given suggestive names. If, e.g., the identifier N is used instead of its value throughout a program, a change of that constant can be achieved by changing the program in a single place only, namely in the declaration of N. This avoids the common mistake that some instances of the constant, spread over the entire program text, remain undetected and therefore are not updated, leading to inconsistencies.

Page 45. Comments in the code.

The definition is the relevant extract from the module listed here, implementing a fifo (first in first out) queue. This fact, however, is not evident from the definition alone; normally the semantics are mentioned in the form of a comment or other documentation. Such comments will usually explain what the module performs, but not how this is achieved.

Page 46. Clarity of the code.

program clarity is enormously important, and to demonstrate the correctness of a program is ultimately a matter of convincing a person that the program is trustworthy. [...] The only salvation lies in structure. A program must be decomposed into partitions which can be considered one at a time without too much regard for the remaining parts. [...] the programmer who has the courage to restructure when a better solution emerges is still much better off than the one who resigns and elaborates a program on the basis of a clearly inadequate structure, for this leads to those products that no one else, and ultimately not even the originator himself can "understand".

Page 64. Decoupling.

The second, but not less important reason is to make it possible to change (improve) the innards of imported modules without having to change (and/or recompile) the importing modules. This effective decoupling of modules is indispensible for the development of large programs, in particular, if modules are developed by different people, and if we regard the operating system as the low section of a program's module hierarchy. Without decoupling, any change or correction in an operating system or in library modules would become virtually impossible.

Proposed Work Steps Based on Niklaus Wirth Recommendations.

1. Avoid hardwired literal constants like 12, 09X, or 0DX.

2. Use named CONST instead, like CTRL_REG, TAB, EOL.

3. Define such constants in "definition modules" (a.k.a. header files) and import these.

4. Develop either one such module SysDef.Mod, or a few topical modules. For example:

    SysDef.Mod for Oberon System definitions.

    DisplayDef.Mod for display definitions (resolution, timing, etc).

    BoardDef.Mod for board related stuff (peripheral addresses etc).

5. Keep the number of the topical modules to a reasonable minimum. A single SysDef.Mod may be sufficient.

6. Develop Utils.Mod to hide the coding idiosyncracies of the Oberon System. Hide the idiosyncracies in procedures whose names make sense, like **`GetArg`** hiding the following coding "idiom":

    Texts.OpenScanner(S, Oberon.Par.text, Oberon.Par.pos); Texts.Scan(S);

7. Unify handling of the white space characters. E.g., System.Directory is skipping the spaces, but not the tabs. This is hardly acceptable nowadays. Fix all places where the command arguments are obtained from the text, either directly or indirectly.

8. Review the System and identify murky acronyms, such as vwr, dsc, par, etc. Wrap these into reasonably named procedures. (The previous bullet.)

9. Identify back door kludges, such as "!" after the file name or Unix-like switches implemented deep inside the code with little documentation. Discuss with the developers whether kludges are really a good idea and why they are needed. Propose implementing new commands rather than adding "options" to existing ones.

10. Find and document all undocumented features. Ask the developers to reveal their secret undocumented features and make them documented.

11. Implement a coherent programming style developed by Oberon Microsystems. (I am not aware of any official ETH Oberon coding style. The OMS code is of significantly higher quality because OMS worked out their programming style and they used it.)

12. Extend the language syntax by allowing an underscore in the names. This will enable the C-style constant declarations (all capital, like CTRL_REG_1, etc.). Persuade the language maintainers to allow the underscore as the "undocumented feature". If met with resistance, ask them to allow the underscores in the CONST identifiers.

13. Restructure the code appearance by breaking the multiple instructions per line. Find a compromise between well formatted, readable code, and the traditional ETH piled up code.

    Examples: I "unrolled" Trap and Deferred in System.Mod. Comparing the unrolled code with the piled up code made it clear that unrolling improves the code clarity. It is a cheap fix with a handsome payoff.

14. Find like-named variables different only by capitalization. Rename variables to fix this, if the confusion is bothersome.

    Example: *W* and *w* are both used in System.Trap. These two letters look very similar in Courier. One of them is local, the other global. Using the same letter is very confusing to the human reader. Neither of these single letters convey any meaning. Fixing this kind of mishap will much improve the coding quality.

15. Change the meaningless variable names to meanigful ones as much as possible without breaking the official documentation. Single letter names are particularly notorious.

    Explain: Oberon System was originally written with variable names like par, vwr, or dsc. They may be OK for local use. Unfortunately, these names gained the System-wide significance. Such names are counterproductive, if we we agree that the goal of programming is to convey meaning to human readers, as officially stated on page 46 of PIO.pdf. The Oberon System code is contradicting such a promise due to using nondescriptive variable names, the names which differ only by capitalization, or lack of a style guideline.

    In principle all this should be very easy to fix. However, introducing meaningful names would break the entrenched Oberon tradition.

14. Add comments to the Oberon source, providing enough information that the source can be understood without memorizing the Oberon books. Provide references to the page numbers of the Oberon System book (online edition), as well as short explanations of what is going on in the code.

15. Do not be afraid of expanding the code. Do not try to minimize the number of lines. Provide clarity and quality rather than a low line count.

16. Provide a run time version identification and the author for every module..

17.Write both the comments and the documentation in good English style.

18. Implement Stack Guard Pages. E.g, assume we have 1 MB. Declare stack to be 128 kB, but memory map onto the whole 1 MB. Move the remaining RAM (1 MB - 128 kB) to start at 2 MB. This will leave a gap. Map this gap to a "memory trap" which will raise an interrupt when addressed.

19. Concerning the new facilities System.RunRsc and System.RunBin, consider the changes:

    a) maxCode and maxBin declared in SysDefs.Mod.

    b) Work tables created on the heap because the stack is rather small.