Notes part 5

Clean C++20. Ch. 4. Basics of Clean C++

• Develop your software using the newest standard, currently C++20 – and never earlier than C++14 (pp. 65-66)
• Descriptive and self-explanatory naming eases readability (pp. 66-70)
– a function/subroutine should do one, precisely defined task represented by it's name
– if is tricky to decide a name, it is probably a design error
– blank lines for code groups indicates requirement of a sub-function
– avoid very exhaustive and too verbose names that creates chaos and less readability
– avoid cryptic abbreviations
• DDD – domain-driven design – code reflects the real-life domain it is designed for (p. 70-75)
• Avoid Hungarian notation (pp. 75-79) e.g.
– use `bool audioEnabled;` instead of `bool fEnabled; // f = a Boolean flag`
– don't use unnescessary comments
• Avoid disabling production code with comments (pp. 79-80)
– disabling code can be beneficial during *development* and *debugging*
• Avoid multiline block comments (pp. 80-85
– except as *file header* of a source code file *before* code
– a *one-line block comment* to categorize or group code might enhance readability e.g.
`// Event handler`
• Place license in a separate `license.txt` file (or `license.rtf` for installer) – not in the file header
• Doxygen – document generation from code (p. 86-89)
• Functions/subroutines should not be larger than 15 lines (pp. 89-100)
• Avoid flag parameters – use multiple self-explanatory functions/subroutines instead (pp. 100-102)
• Avoid output parameters (pp. 103-104)
• Avoid `return nullptr;` (pp. 104-107)
• Avoid C style, where applicable (p. 108-127)
- C-style null-terminated strings (`char*`)
- C-style I/O: e.g. `printf` and `scanf`
- C-style Memory Management
- C-style Pointers
- C Enumerations: `enum`
- C-style Structures: `struct`
- C-style Function Pointers
- C-style Preprocessor Directives: use #define and #ifdef scarcely
- C-style Type Casting: e.g. `(int)string` and `(int)bool`

• Avoid macros (p. 128-130)