

Software Development with C

Coding Guidelines

Dipl.Ing. Alexander Neundorf

<neundorf@eit.uni-kl.de>

30.06.2008

Outline

- Motivation
- Some Examples
 - MISRA, JSF++,...
- Coding Guidelines for the Lab
- Documentation

Motivation

- Coding guidelines:
 - General guidelines on what to do and what not to do
 - Language features to avoid
 - Code formatting
- *Why ?*
 - Fewer bugs:
 - avoid typical pitfalls
 - Better maintainability:
 - code should be easy to read and understand by other developers
 - ***Code is read much more often than it is written !***

Example: MISRA ^(TM) C



- Motor Industry Reliability Association
- 1998: MISRA C 1.0, 127 rules, based on ISO C89
- C coding guide for safety critical systems
- [http://computing.unn.ac.uk/staff/cgam1/teaching/0703/misra rules.pdf](http://computing.unn.ac.uk/staff/cgam1/teaching/0703/misra%20rules.pdf)
- Tool support available:
 - PolySpace, Gimpel PC-lint, Green Hills, Tasking, ...



MISRA C Examples

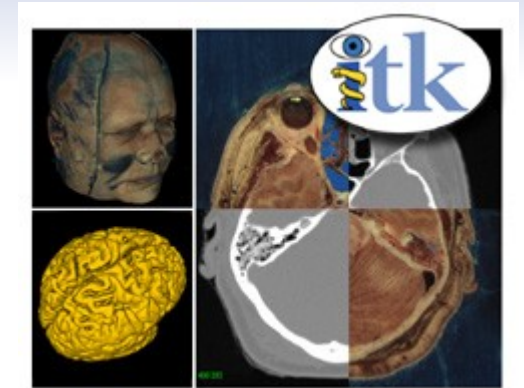
- §1: Conformance to ISO 9899, no extensions
- §19: No octal constants
- §30: Initialize all automatic variables
- §50: Don't tests floats for equality
- §52: No unreachable code
- §56, 57, 58: No break, continue, goto
- §59: Always use curly braces { }
- §60: Always have an else clause after if
- §70: No recursions
- §107: Don't dereference the NULL pointer
- §118: No dynamic heap memory (`malloc`)

Joint Strike Fighter C++



- 2005: C++ coding standard for JSF
- Safety critical, hard real-time system
- <http://www.research.att.com/~bs/JSF-AV-rules.pdf>
- 220 rules, e.g.:
 - §41: Max line length is 120
 - §43: No tabs
 - §69: Member functions are const by default
 - §85: When overloading == and !=, one will be defined using the other
 - §208: No C++ exceptions
 - §216: Don't optimize code prematurely
 - §217: Prefer compile/link errors over runtime errors

More Examples



- Insight Segmentation and Registration Toolkit
<http://www.vtk.org/Wiki/images/c/c6/ITKStyle.pdf>
- Linux kernel: <http://lxr.linux.no/linux/Documentation/CodingStyle>
- MS Win32: [http://msdn.microsoft.com/en-us/library/aa378932\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa378932(VS.85).aspx)
- Qt: http://techbase.kde.org/Policies/Kdelibs_Coding_Style

- Two “**Must read**”s:
 - *How to write unmaintainable code:* <http://mindprod.com/jgloss/unmain.html>
 - *Designing Qt-Style C++ APIs:* <http://doc.trolltech.com/qq/qq13-apis.html>

RTS TUKL Coding Style

- Code must be easy to read -> code must look good !
- Make clear what goes on !
- In doubt, prefer explicit and verbose !
- No clever tricks – Keep It Simple, Stupid (KISS) !

The Details....

Documenting Code

- Problems with documentation:
 - No fun
 - “Far away” from the code, takes effort to edit it
 - Easily forgotten to adapt to changes
- “*Literate Programming*” - Documentation embedded in the code
 - Still no fun
 - Embedded in the code, so editing takes less effort
 - Forgetting to adapt the documentation a lot harder
- Several tools, e.g. Doxygen

Doxygen (<http://www.doxygen.org>)

- Portable, supports C++, C, Java, Fortran, VHDL and more
- Generates HTML, RTF, PDF, Latex and more

- Documenting a function:

```
/** Open the file with the given \a filename in the given \a mode.
 * \param filename The file to open
 * \param mode A combination of 'w', 'b', 'r' and '+'
 * \return On success a FILE* pointer is returned, otherwise NULL.
 * \sa fclose() */
FILE* fopen(const char* filename, const char* mode);
```

- Documenting a struct:

```
/** struct stat is used to return information about a file. */
struct stat
{
    uid_t st_uid;    /**< The user id of the owner of the file. */
    off_t st_size;  /**< The size of the file in byte. */
    ...
};
```

Summary

- Be happy if you work on a project which has a coding style
- Follow it !

Questions, Comments & Complaints

