Computer Programming Good Practices

for physicists and PhD candidates

Vincent LAFAGE

[1]IJCLab, Laboratoire de Physique des 2 Infinis Irène Joliot-Curie
Université Paris-Saclay

vendredi 7 mai 2021

Preface

- best practices ?
- better practices ?
- good practices ?

looks very moral : a lot of principles indeed... and much casuistry as well.

We will take a more hygienic / prophylactic approach :
bugs are more like germs than demons !
They grow in unclean programming environment.
⇒ A living code needs regular cleaning.

This is NOT a talk about Software Quality
⇒ No ISO formalism, no administration, no formal process

This is about Self-Defense !

More guidelines than hard rules...

We will also address ethical implications later.

Software engineering

a craft

- 1968 NATO Software Engineering Conferences
  `http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF`
- term probably coined by Margaret Hamilton, lead Apollo flight software designer

Hippocrates
Vīta brevis, ars longa, occāsiō praeceps, experīmentum perīculōsum, iūdicium difficile. [1]

Peter Norvig *Teach Yourself Programming in Ten Years*
`https://norvig.com/21-days.html`

1. Life is short, and art long, opportunity fleeting, experimentations perilous, and judgment difficult.

Prototype code

90 % of our code has not much life expectancy.

- − it's of no consequence
- \+ it's an excellent playground to train our practice

Train regularly on recoding small examples : programming kata

Know your tool

execute typical instruction	1 ns
fetch from L1 cache memory	0.5 ns
branch misprediction	5 ns
fetch from L2 cache memory	7 ns
Mutex lock/unlock	25 ns
fetch from main memory	100 ns
send 2K bytes over 1Gbps network	20 000 ns
read 1MB sequentially from memory	250 000 ns
fetch from new disk location (seek)	8 000 000 ns
read 1MB sequentially from disk	20 000 000 ns
send packet US to Europe and back	150 000 000 ns

Separate data & processing

Excel™ (any spreadsheet) is an *excellent prototyping tool* : all is included (data, processing and display)
...but it is a *bad industrial tool* :

- the implied code is hidden
- there's no check (no compiler)
- easy to use without applying adequate scrutiny, oversight and validation

⇒ Architect your code with a clear cut entrance and exit

⇒ Use standard file formats (`.txt`, `.csv`, `.hdf5`, `.cdf`, `.fits`, `.root`)

⇒ Use databases (`PostgreSQL`)

Code Comments

what is a good comment ?

- Usefulness : is it a comment or a paraphrase ?
- non Duplicity : is it a comment or a paraphrase ?
- Clarity : is the comment really clearer than the code ?
- Conciseness / Brevity
- Objectivity
- Insight : comment should explain/illustrate the Finality of the code
- $\Rightarrow$ **Comment Only What the Code Cannot Say**
- *Don't comment bad code — rewrite it.*
- *A comment is of zero (or negative) value if it is wrong.*
- *Comment never get checked by the compiler, nor by the human...*

Noisy Code

```
/*
 * Copyright (c) 1995, 2008, Oracle and/or its affiliates. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *   - Redistributions of source code must retain the above copyright
 *     notice, this list of conditions and the following disclaimer.
 *
 *   - Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in the
 *     documentation and/or other materials provided with the distribution.
 *
 *   - Neither the name of Oracle or the names of its
 *     contributors may be used to endorse or promote products derived
 *     from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
 * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/**
 * The HelloWorldApp class implements an application that
```

Comments

A delicate matter, requiring taste and judgement. I tend to err on the side of eliminating comments, for several reasons. First, if the code is clear, and uses good type names and variable names, it should explain itself. Second, comments aren't checked by the compiler, so there is no guarantee they're right, especially after the code is modified. A misleading comment can be very confusing. Third, the issue of typography : comments clutter code.

Rob PIKE, *"Notes on Programming in C"*

`http://www.literateprogramming.com/pikestyle.pdf`

A common fallacy is to assume authors of incomprehensible code will somehow be able to express themselves lucidly and clearly in comments., Kevlin HENNEY

Comments

There is a famously bad comment style :

```
i=i+1;              /* Add one to i */
```

and there are worse ways to do it :

```
/**********************************
 *                                *
 *          Add one to i          *
 *                                *
 **********************************/

i=i+1;
```

Don't laugh now, wait until you see it in real life.

Versioning

git 101

`git` Use cases for the newbie :

- Getting some code from colleagues, collaboration
  - `git clone` *URL*
- Versioning your own code on your own machine
  1. `git init`
  2. type a first version of your text/source file
  3. `git add` *filename*
  4. compile it, test it, check it, execute it... until it passes
  5. `git commit -m''`*message*`''`
  6. improve your file
  7. back to step 4
- Later, you'll share it on a central repository such as a `gitlab` instance

Commit often, commit early

Start using it for your logbook, your diary. Practice every day.

Later, when you feel at ease, use branches...

Commit comments

The Conventional Commits : a lightweight convention on top of commit messages. easy set of rules for creating an explicit commit history.

```
<type>[optional scope]: <description>
[optional body]
```

type :

- `feat` : introduces a new feature to the codebase
- `fix` : patches/squashes a bug in your codebase
- `perf:` improves performance (less memory, less IO, more speed...)
- `refactor:` rewrite/restructure your code, however does not change any behaviour (nor performance)
- `tests` : add missing tests or correcting existing tests
- `style` : do not affect the meaning (white-space, formatting, missing semi-colons, etc)
- `build` : affect build components like build tool (`Makefile`), dependencies, project version,
- `docs` : affect documentation only
- `conf` : affect configuration only
- `chore` : miscellaneous (for instance, related to the versioning system)
- `ci` : continuous integration
- `pack` : packaging specification (such as `rpm` or `deb` specification files)
- `ops` : affect operational components like infrastructure, deployment, backup, recovery, ...
- `edit` : modify a feature
- `del` : delete a feature
- DRY : don't mention the file(s), the date, the author(s)

Versioning comment

examples

```
feature : compute Archimedes's constant  integrating disc area by M
fix : simplify assignment
fix : adapt OpenMP directive to older compilers
fix : adapt assignment to more generic shell
build : deal with older c++ compiler
fix : align type of loop index with type of bound
fix : align type of loop index with type of bound
style : clarify the meaning of printed report
fix : align type of litteral index with type of assignment variable
style : turn to standard C-style loop and align type of litteral wi
style : clarify the meaning of printed report
fix : align type of loop index, type of index bound and type of lit
```

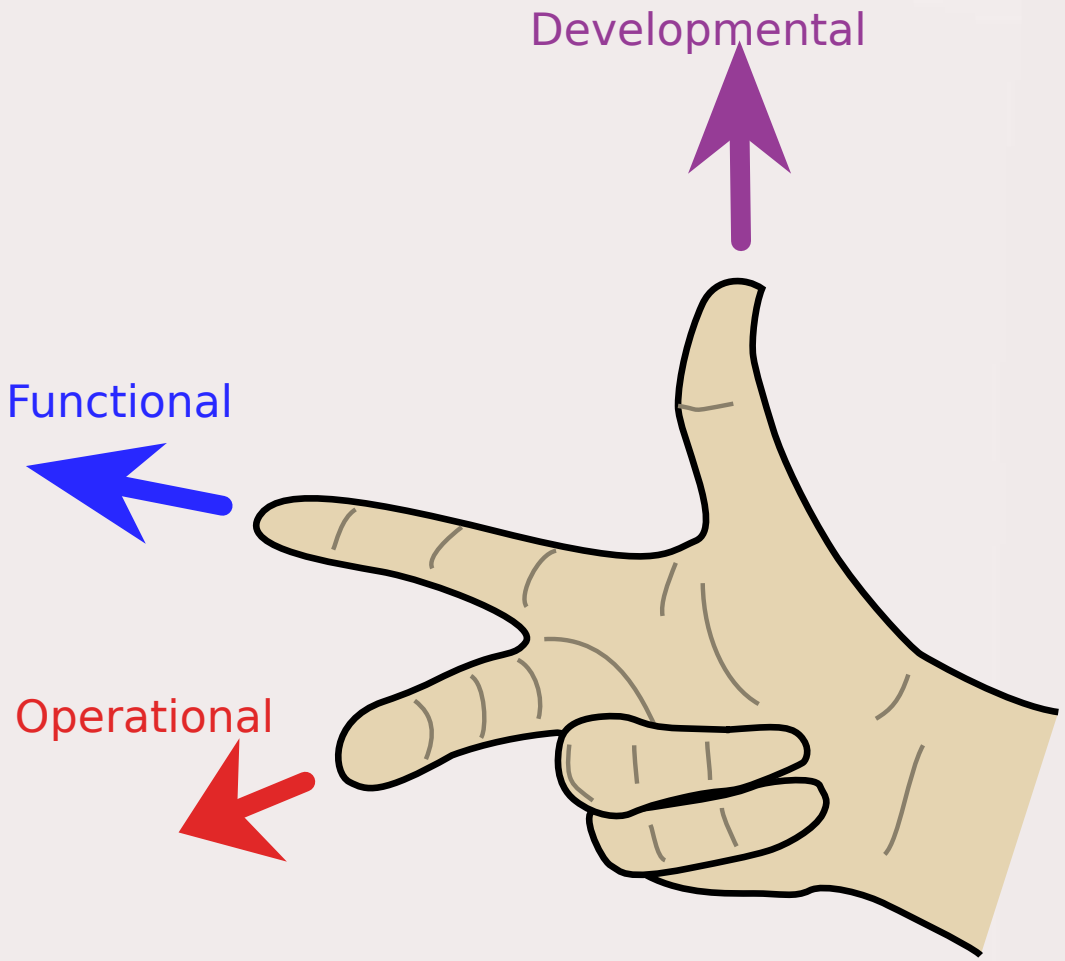
Versioning comment

examples

```
# Commit message with no body
feat: allow provided config object to extend other configs

# Commit message with scope
feat(lang): add polish language

# Commit message with ! to draw attention to breaking change
refactor!: drop support for Node 6

# Commit message with both ! and BREAKING CHANGE footer
refactor!: drop support for Node 6

BREAKING CHANGE: refactor to use JavaScript features not available in
```

Development space

degrees of freedom



Know your tool

editor

Beyond the `emacs` *vs.* `vi` war

« Je suis de la religion de ma nourrice et de mon roi », Descartes

- `atom`
- `vscode`
- `eclipse`
- `sublime` (hassleware)
- `Code::Blocks` IDE
- …

avoid

- `notepad`
- `nedit`
- `gedit`

Know your tool

editor

Beyond cut&paste and search

- auto indent
- colorized syntax for your language(s)
- rectangular cut&paste
- regular expression
- delete trailing whitespace
- untabify (turn tabs to spaces)
- refactoring menu :
  - rename identifier in the full code

Lines

size matters

66 characters for Jurassic Fortran was certainly not enough

but it is the typical size of a newspaper column,

for a good reason : our average brain buffer size / eye coordination capability

$\Rightarrow$ support your brain !

132 is way too much :

modern screens can take it (even more for your extra-wide screen)

can your team video projector take it ?

80 character is a nice constraint

Indent complexity

Use it as a visual design ; illustrate the structure : supports your brain !

Spacing

One whitespace is usually enough between two keywords or identifiers
One blank line is enough to separate parts of code

$\Rightarrow$ no more than one blank line

Typography

We usually read a long time before we code
$\Rightarrow$ we take a lot of typographic standard for granted

- space after comma, not before
- space after closing parenthesis, not before
- space before opening parenthesis, not after
- no space before double punctuation, one after (English typography)
- ...

$\Rightarrow$ support your brain !

Casing

Roman empire had only upper case : they have fallen twice !
I wouldn't jump to conclusion...
We usually read much more lower case than upper case
$\Rightarrow$ support your brain !

Style

To be, or not to be, that is the question :
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take Arms against a Sea of troubles,
And by opposing end them ?

William Shakespeare
Hamlet

Continuing existence or cessation of existence : those are the scenarios. Is it more empowering mentally to work towards an accomodation of the downsizings and negative outcomes of adversarial circumstance, or would it be a greater enhancement of the bottom line to move forwards to a challenge to our current difficulties, and, by making a commitment to opposition, to effect their demise ?

Tom Burton
Long Words Bother Me

Naming Identifiers

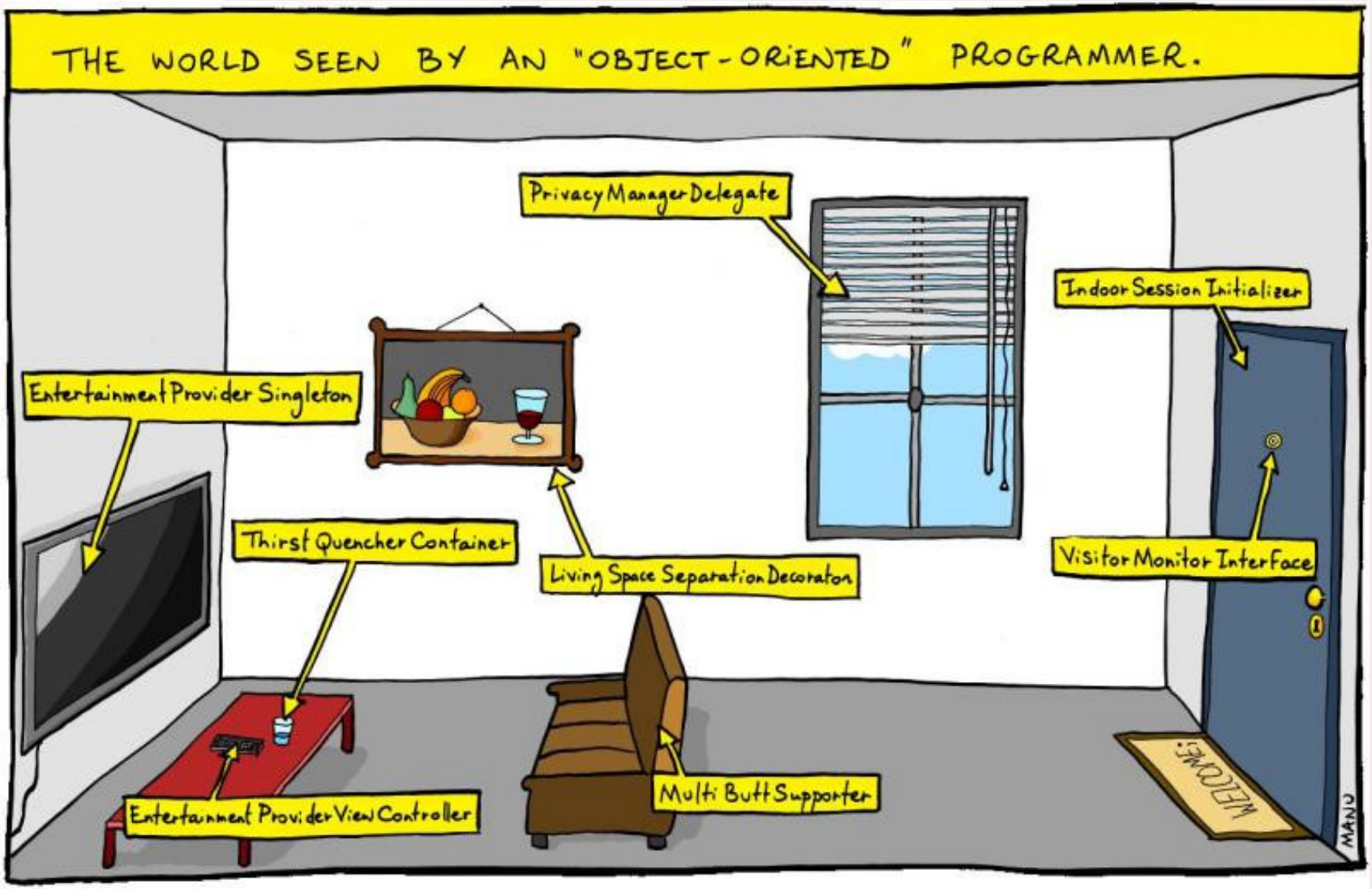
Avoid Lego naming

Put/stuff more meaning in the name so as to render comment useless
Take care : Clearer, not longer

don't get your code in turkish/japanese/german : agglutinative language

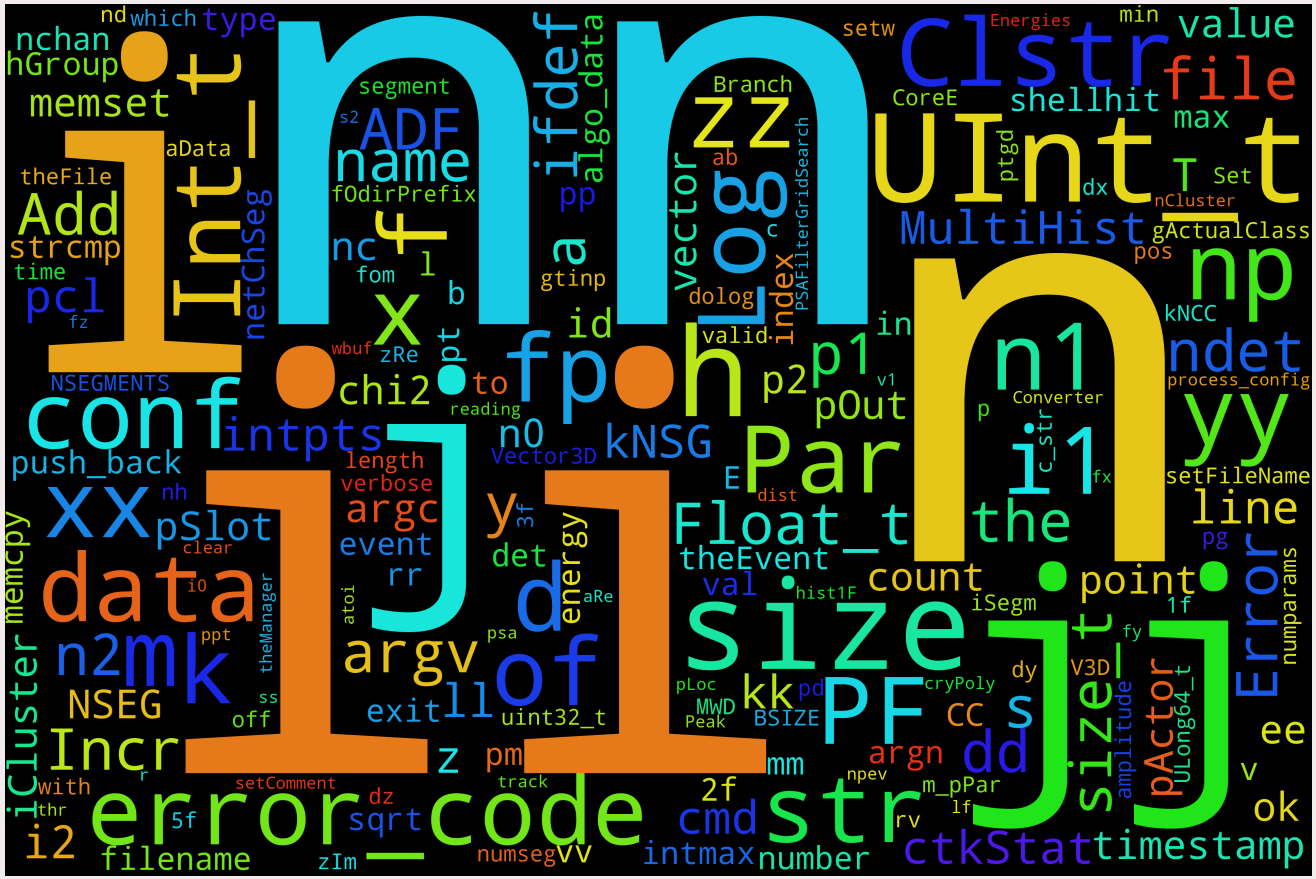
Object Oriented naming pitfalls

Avoid Lego naming



Underabstraction

What is the code about ?



removing comments
ignoring language keywords
as well as `std` functions and types
use wordcloud generator

⇒ does not reveal much intent...

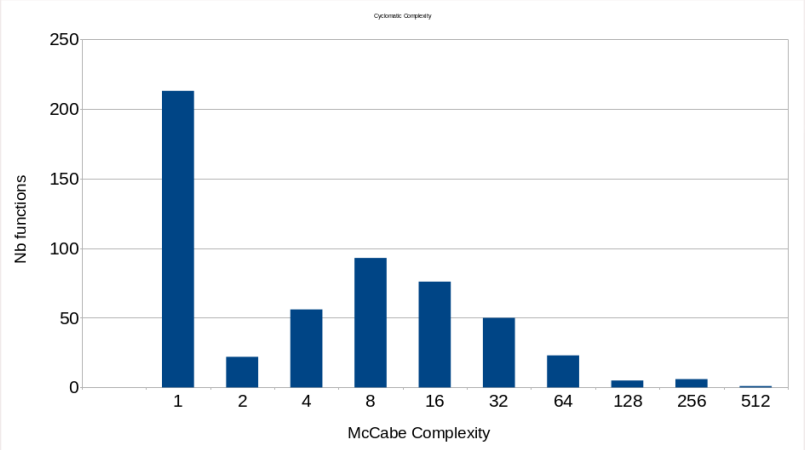
3 139 `for` loops (1 415 distincts, 620 duplicates)

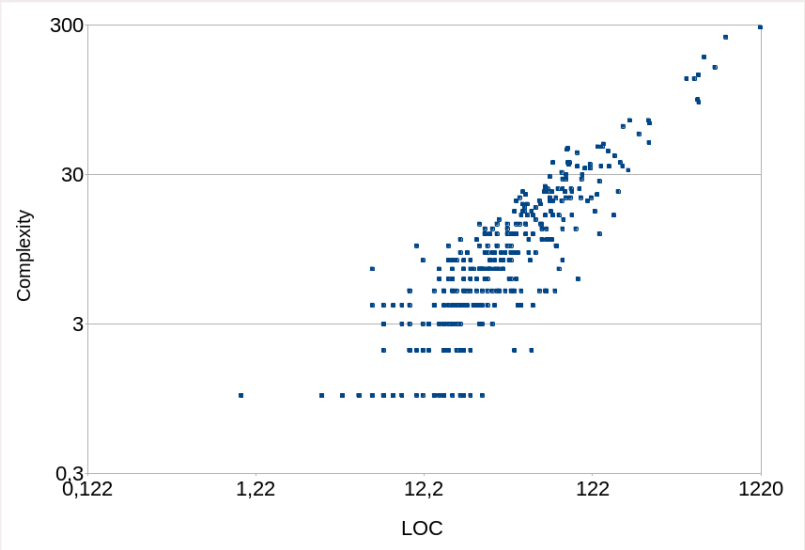
"There are only two hard things in Computer Science : cache invalidation and naming things."

— Phil Karlton

Cyclomatic complexity

⇒ « *A measure of the decision complexity of the functions* »
⇒ « *number of possible paths in a function flow* »
⇒ each `if`, `for`, `while` or `switch` increases cyclomatic number.





Essence of OOP

The Law of Demeter (LoD) or principle of least knowledge is a design guideline for developing software, particularly object-oriented programs. In its general form, the LoD is a specific case of loose coupling.

- Each unit should have only limited knowledge about other units : only units "closely" related to the current unit.
- Each unit should only talk to its friends ; don't talk to strangers.
- Only talk to your immediate friends.

$\Rightarrow$ limit long range coupling

Build

a recipe to cook your executable

write the recipe : the `Makefile` ; a part of the documentation

- First, most coder get their first `Makefile` from colleagues
- Then, don't stop, improve your `Makefile`

— use standard names for `Makefile` variables : `CC`, `CPP`, `CXX`, `FC`, `CFLAGS`,...

— use standard metasyntactic variables for `Makefile`

— expand the targets : get your `Makefile` to produce your doc with `Doxygen`, your changelog with `git` and semantic versionning

Know your tool

Compiler's Options

```
g++ -std=c++11
    -Ofast -march=native -mtune=native
    -g3
    -Wall -Wextra --pedantic
    -Wformat=2 -Wwrite-strings -Wunreachable-code
    -Wshadow -Wstack-protector
    -o montecarlo++ montecarlo.cxx
```

The compiler is your friend listen to warnings!

... and asks the compiler for advice

```
...
-Wall -Wextra --pedantic
-Wformat=2 -Wwrite-strings -Wunreachable-code
-Wshadow -Wstack-protector
...
```

Software Architecture

a craft & an art

You spend time in your software : what kind of place is it to live in ?

- a dormitory ?
- a parking lot ?
- an attic ?
- a workshop ?
- a storage room ?
- a patio ?

Would you invite people to share this place ? How do you feel (re)entering it ?
Marcus VITRUVIUS Pollio, *De architectura* :

all buildings should have three attributes : *firmitas*, *utilitas*, *venustas*[2]

2. stability, utility, beauty

Unit Testing

Write the properties your function is supposed to
Code these assertions
then program the function

Build a set of test cases

Floating (point) world

Revisiting "What Every Computer Scientist Should Know About Floating-point Arithmetic"

- Numbers : real, decimal, binary, floating point...
- When computations don't turn out as expected...(why, how)
  - global errors
  - local errors
  - composing errors
- Heuristics for accuracy :

  how a rough estimate can save epsilons
- How to reconcile adimensionalisation and performance
- How to reconcile abstraction and accuracy : functions of a complex variable
- Why are geometrical computations so hard
- The hidden side of functional programming : towards total functions

$\mathbb{D} = \left\{ \frac{n}{10^p}, n \in \mathbb{Z}, p \in \mathbb{N} \right\} = \mathbb{Z}[1/10]$ (decimal)
$\mathbb{B} = \left\{ \frac{n}{2^p}, n \in \mathbb{Z}, p \in \mathbb{N} \right\} = \mathbb{Z}[1/2]$ (binary)
$\mathbb{B} \subset \mathbb{D}$ mais $\mathbb{D} \not\subset \mathbb{B}$: $\frac{1}{5} \in \mathbb{D}$, $\frac{1}{5} \notin \mathbb{B} \Rightarrow 0.1 + 0.2 \neq 0.3$ $\left(\frac{1}{5} = 0.00\overline{1100}_2 \ldots \right) \Rightarrow$ not good for financial computations...

- closure :
  $\forall (x, y) \in \mathbb{B}^2, \quad x + y \in \mathbb{B}$,
  $\forall (x, y) \in \mathbb{B}^2, \quad x \times y \in \mathbb{B}$
- commutativity :
  $\forall (x, y) \in \mathbb{B}^2, \quad x + y = y + x$,
  $\forall (x, y) \in \mathbb{B}^2, \quad x \times y = y \times x$
- associativity :
  $\forall (x, y, z) \in \mathbb{B}^3, \quad x + (y + z) = (x + y) + z$,
  $\forall (x, y, z) \in \mathbb{B}^3, \quad x \times (y \times z) = (x \times y) \times z$
- distributivity :
  $\forall (x, y, z) \in \mathbb{B}^3, \quad x \times (y + z) = x \times y + x \times z$
- total order :
  $\forall (x, y, z) \in \mathbb{B}^3, \quad x \leq y$ and $y \leq z \Rightarrow x \leq z$ (transitivity) ;
  $\forall (x, y) \in \mathbb{B}^2, \quad x \leq y$ and $y \leq x \Rightarrow x = y$ (antisymmetry) ;
  $\forall x \in \mathbb{B}, \quad x \leq x$ (reflexivity) ;
  $\forall (x, y) \in \mathbb{B}^2, \quad x \leq y$ or $y \leq x$ (totality).
- topology :
  $\mathbb{B} \subset \mathbb{D} \subset \mathbb{Q}$ are dense in $\mathbb{R} \Rightarrow$ arbitrarily close approximations to the real numbers

Decimal *vs.* binary ...and binary *vs.* floating

- closure :
  $\exists (x, y) \in \mathbb{F}^2, \quad x + y \notin \mathbb{F}$,
  $\exists (x, y) \in \mathbb{F}^2, \quad x \times y \notin \mathbb{F}$
  $\Rightarrow$ rounding and extension $\overline{\mathbb{F}} = \mathbb{F} \cup \{\pm\texttt{Inf}\} \cup \{\texttt{NaN}\} \cup \{\texttt{0\_}\}$ overflow, underflow, inexact
- commutativity :
  $\forall (x, y) \in \mathbb{F}^2, \quad x + y = y + x$,
  $\forall (x, y) \in \mathbb{F}^2, \quad x \times y = y \times x$
- associativity :
  $\exists (x, y, z) \in \mathbb{F}^3, \quad x + (y + z) \neq (x + y) + z$,
  $\exists (x, y, z) \in \mathbb{F}^3, \quad x \times (y \times z) \neq (x \times y) \times z$
- distributivity :
  $\exists (x, y, z) \in \mathbb{F}^3, \quad x \times (y + z) \neq x \times y + x \times z$
- total order :
  $\forall (x, y, z) \in \mathbb{F}^3, \quad x \leq y$ and $y \leq z \Rightarrow x \leq z$ (transitivity) ;
  $\forall (x, y) \in \mathbb{F}^2, \quad x \leq y$ and $y \leq x \Rightarrow x = y$ (antisymmetry) ;
  $\forall x \in \mathbb{F}, \quad x \leq x$ (reflexivity) ;
  $\forall (x, y) \in \mathbb{F}^2, \quad x \leq y$ or $y \leq x$ (totality).
  $\exists (x, y) \in \overline{\mathbb{F}}^2, \quad x \leq y$ and $y \leq x$ (NaN).
- topology :
  $\mathbb{B} \subset \mathbb{D} \subset \mathbb{Q}$ are dense in $\mathbb{R} \Rightarrow$ arbitrarily close approximations to the real numbers
  but
  $\mathbb{F}$: floating point numbers, finite parts of $\mathbb{B}$ (or $\mathbb{D}$) are dense nowhere

Formats

« computing is about representation »

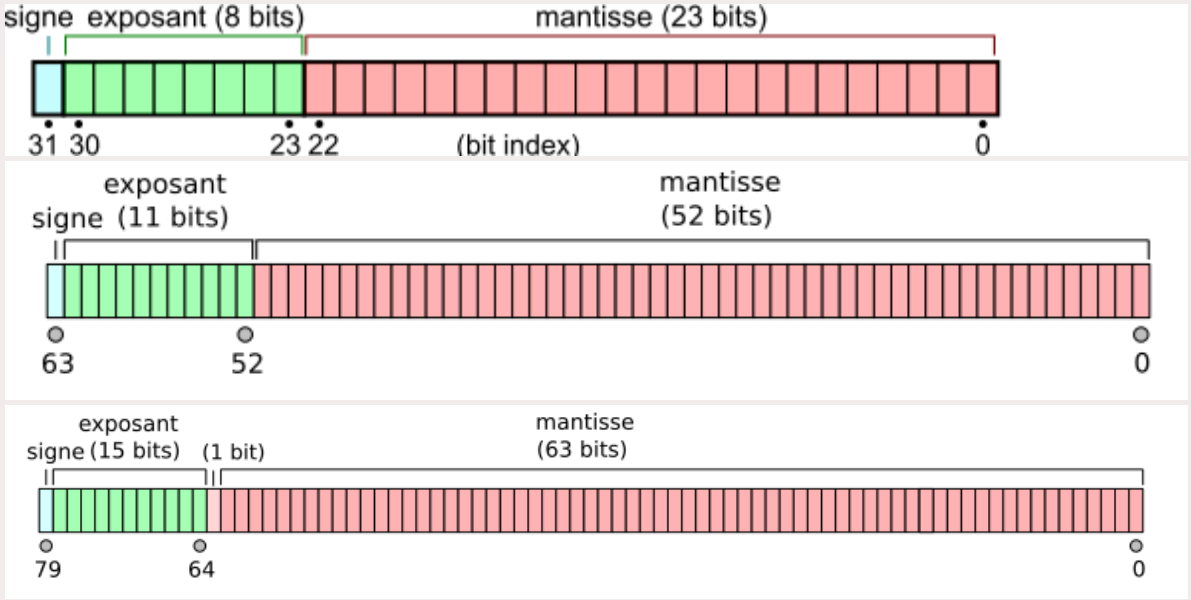
Scientific notation :

significand $\times$ base$^{\text{exponent}}$ significand $\in \mathbb{Z}$, exponent $\in \mathbb{Z}$

Standard form : mantissa, alias *normalized significand*

mantissa $\times$ base$^{\text{exponent}}$ mantissa $\in [1; \text{base}[$, exponent $\in \mathbb{Z}$

Trick, for base 2 : the most significant digit is always 1...



In the registers, we widen mantissa with three bits :

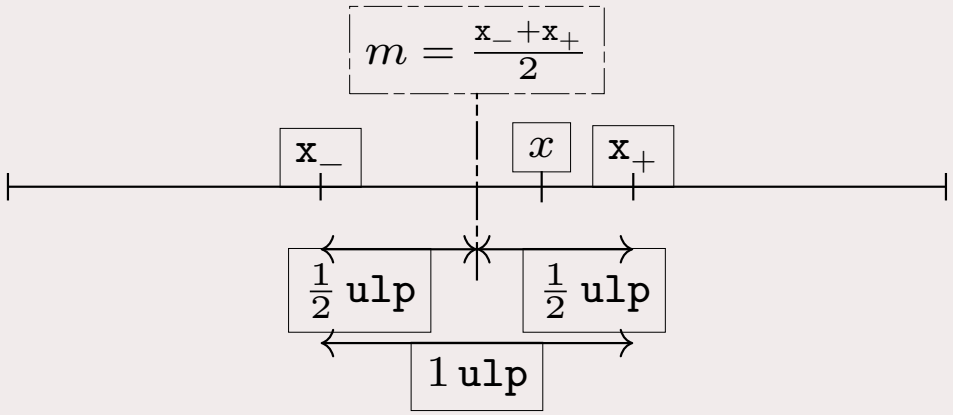
- guard bit
- round bit
- sticky bit

Problems

- apparent : rounding $\Rightarrow$ catastrophic `cancelation`
- apparent : conversion. Goes unnoticed or perceived as minor.
- apparent : `overflow`. Apparent, but not treated.
- less apparent : `underflow`, gradual `underflow` : denormal numbers

Rounding

$\forall x \in \mathbb{R}, \exists(\mathtt{x}_-, \mathtt{x}_+) \in \mathbb{F}^2 \,|\, \mathtt{x}_- \leq x \leq \mathtt{x}_+$ (closest representable neighbours)



$\Rightarrow$ correct rounding requires at least 2 extra bits beyond target accuracy
or even more (*table maker's dilemma*)
correct rounding, faithful rounding, happy-go-lucky rounding

rounding is non-linear but completely deterministic !

Conversion

- $\mathbb{D} \not\subset \mathbb{B}$: every decimal is not a binary
  $\Rightarrow$ conversion to binary relies on rounding
  $\frac{1}{5} = 0.2_{10} = 0.00\overline{1100}_2 \cdots \ominus 13421773 \times 2^{-26} = 0.2 + 2,98 \times 10^{-9}$

4 byte	`float`	$25.4E0 = 25.399999619\cdots$
8 byte	`double`	$25.4D0 = 25.39999999999999858\cdots$
10 byte	`long-double`	$25.4T0 = 25.399999999999999653\cdots$
16 byte	`quadruple`	$25.4Q0 = 25.399999999999999999999999999999877\cdots$

- $\mathbb{B} \subset \mathbb{D}$: every binary is a decimal
  However, converting a binary, usually from a computation, usually for display or storage, is not toward the exactly corresponding decimal : it would require too many meaningless decimal digits.
  $\frac{1}{8} = 0.001_2 = 0.125_{10} \ominus 0.1_{10} \cdots$
  $\Rightarrow$ conversion to decimal also relies on rounding

Can division by a constant be replaced by multiplication by this constant reciprocal ?
This replacement can induce an extra uncertainty.
Counter-example : dividing by 2 (has an exact representation) induces no uncertainty, and the reciprocal of 2 having an exact representation, multiplying by $\frac{1}{2}$ induces no uncertainty either.
Example : dividing by 5 or by 10, or even by 3 : one uncertainty coming from division operation, two uncertainties coming from multiplication operation and misrepresentation of operand
Counter-example : dividing by π : the inexact representation of π induces one uncertainty, the inexact representation of its reciprocal also induces one uncertainty (almost the same relative uncertainty : $0,37\,\mathtt{ulp}$ and $0,43\,\mathtt{ulp}$ respectively)

Catastrophic Cancellation ?

By way of exception in base 10 (not in binary) ! mantissa : 3 decimal digits
For $a = 3.34$ and $b = 3.33$

- $a \ominus b = 0.01 \Rightarrow$ **cancellation** (relative precision loss)
  but a **benign** one (the floating point result is exact : $a \ominus b = a - b$)
- $\begin{cases} a^2 - b^2 & = 0.0667 = 6.67 \times 10^{-2} \\ a \otimes a \ominus b \otimes b & = 0.1 = 1.00 \times 10^{-1} \end{cases}$
  50% of relative error on the result, or 333 `ulp`, no digit is even correct :
  catastrophic cancellation
- When does this occur ?
- How many digits are lost ?

Plus, there is an ***overflow*** risk
$\Rightarrow$ Let's factorize this !

$$(a \oplus b) \otimes (a \ominus b) = 6.67 \otimes 0.01 = 6.67 \times 10^{-2} \qquad \text{exact}$$

$\Rightarrow$ The Right Way™

Area of triangle

Heron of Alexandria, area S as a function of lengths a, b and c of edges

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

$p = \frac{a+b+c}{2}$ half-perimeter

Symmetric, but numericaly unstable, for needle-like triangles (when large and small values meet in the same formula)

Kahan Re-labelling : $a > b > c$

$$\frac{1}{4}\sqrt{[a+(b+c)]\,[c-(a-b)]\,[c+(a-b)]\,[a+(b-c)]}$$

Apparent Symmetry is lost, but the formula is way more robust

Originating from a determinantal expression

$$S = \frac{1}{4}\sqrt{\begin{vmatrix} 0 & a^2 & b^2 & 1 \\ a^2 & 0 & c^2 & 1 \\ b^2 & c^2 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{vmatrix}}$$