C & Java Programming: Course overview

(Everything you always wanted to know (but were afraid to ask))

Emmanuel Fleury
<emmanuel.fleury@labri.fr>

LaBRI, Université de Bordeaux, France

September 1, 2017

Overview

Course Overview (C Programming)

Goal of this course (C Programming)

Reach autonomy in software development in C.

Build a software from scratch in 2 or 3 steps (Reversi game):

1. User-interface (usage, version, option parser) and internal mechanics;
2. An abstract data-type (bitboard) and efficiency of operations;
3. Backtracking algorithms and heuristics.

Course Overview (Java Programming)

Goal of this course (Java Programming)

Reach autonomy in software development in Java.

Discover software architecture:

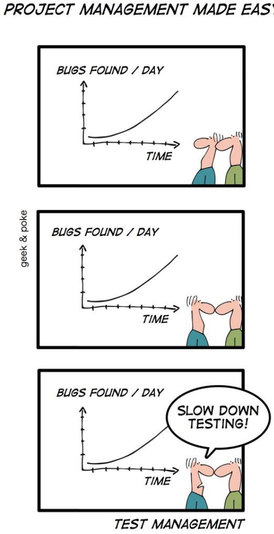
1. Modular programming / Object Programming;
2. Structure of an Object Oriented Program (inheritance, polymorphism);
3. Basic Knowledge of the Java Development Kit (JDK);
4. Java Generics, Reflection, Lambda expressions;
5. Exceptions and Execution model of Java

How Does It Work ? (for the C part)

- **Lectures**: Once a week, I talk about C and programming related topics (and I give the assignment for two weeks after);
- **Practice**: Once a week, you meet your teaching assistant and he helps you with the homework (but it is also required that you work on your assignments outside of these time slots);
- **Homeworks**: Assignments are given during the weekly lecture and result must be sent by e-mail to me, on Tuesday two weeks after (before midnight);
- **Projects**: An individual project will be assigned at the end of the 5 homeworks. It will be a feature to add to the software you were working on. You will be required to deliver both a report (written in LaTeX) and the source code of the project.

How Does Code Evaluation Works ?

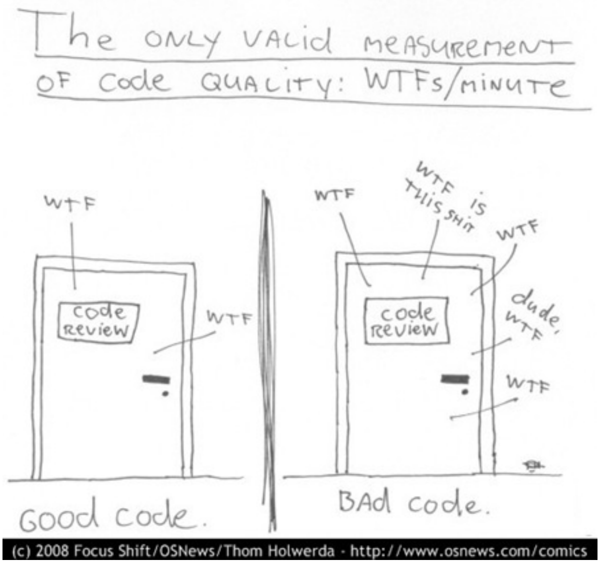
Automated tests (10 points)



GO WRITE SOME AUTOMATED TEST SOFTWARE, YOU BIG BABY. I ALREADY PAY YOU, SO IT'S FREE.
© 2007 Scott Adams, Inc./Dist. by UFS, Inc.

How Does Code Evaluation Works ?

Code reading (10 points)


The only valid measurement of code quality: WTFs/minute
WTF
code review
WTF
Good code.
WTF
WTF is this shit
WTF
code review
dude, WTF
WTF
BAd code.
(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics

Imposed Coding Style for Your Code

- All comments, variable and function names must be in English.
- Do not comment too much, comment only when needed !

```
int number_of_path = 10; /* Number of paths */
```

- Indentations are 2 or 4 spaces wide.
- Tabulations are 8 spaces wide.
- Lines must be 80 columns maximum (except if no other layout is possible).
- Do not use '`// ...`' comments, but '`/* ...*/`' comments.
- Use one space around (on each side) most binary/ternary operators, such as:
  `=  +  -  <  >  *  /  %  |  &  ^  <=  >=  ==  !=  ?  :`
- No space after (or before) unary operators such as:
  `&  *  +  -  ~  !  ++  --`
- No space around structure members operators: '`.`', '`->`'.
- Use a space after these keywords: '`if`', '`switch`', '`case`', '`for`', '`do`', '`while`'.
- Use a consistant way of placing braces all over your code.
- All macro names must be capitalized: `#define CONSTANT 0x12345`
- Insert linebreaks only to highlight the logical blocks of your program, not because you want to.
- ... many others ... (be reasonable and logic).

Code Example

```c
/* usage(status): Display the usage of the program and quit. */
static void usage (int status)
{
  if (status != EXIT_SUCCESS)
    fprintf(stderr,
            "Try `%s -h' for more information.\n", program_name);
  else
    {
      fprintf(stdout, "Usage: %s [OPTION] FILE...\n", program_name);

      fputs("Solve Sudoku puzzle's of variable sizes (1-8).\n"
            "\n"
            "  -o, --output=FILE  write result to FILE\n"
            "  -v, --verbose      verbose output\n"
            "  -V, --version      display version and exit\n"
            "  -h, --help         display this help\n", stdout);
    }

  exit (status);
}
```

Miscellaneous Rules. . .

- **Know your tools** (text-editor, compiler, debugger).
- **Use Subversion/Git** to track your code and deliver your work.
- **Check your program** (write tests, use gdb and valgrind).
- Write **efficient**, **correct** and **robust** code.
- Do not try to write **complex programs**, or you will **fail**.
  Simplicity is good (KISS principle (Keep It Stupid Simple)).
- Do not read/write **dynamic memory without checking it**.
- **Check function return code**, when needed.
- Try to **avoid code duplication** (DRY = Don't Repeat Yourself).
- When you do not know, **search on the Web** !
- **Do not confuse languages** (C/C++, Java/Javascript, . . .).
- **There is no universal rule !** You have to break it sometimes.
- . . .

Cheating Is Not An Option!

Allowed

- Work on the concepts with the others and exchange ideas.
- Look at the code of other students.
- Look at the code of other software which are not from this course.
- Look at Stack-Overflow or other programming-related websites.

Forbidden

- Look or use a full source code from past years.
- Copy/paste code from other sources than you own code.
- If you want to use code from another source, it shouldn't be more than 15 lines of code long (and, NO COPY/PASTE, type it).
- If you already have more than 15 lines from one source, you cannot use it anymore. Find another source.

More Information and Help

- **Course Website**:
  `http://www.labri.fr/~fleury/courses/programming/`
- **IRC Channel**:
  Server: **`irc.freenode.net`**, Channel: **`#mastercsi`**
  Server: **`irc.freenode.net`**, Channel: **`#ubdx-info`**
- **Discussion with the others are okay but...**
  Do not copy code without understanding it !
- **If you find the code of previous years students,**
  DON'T LOOK AT IT ! CHEATING IS BAD !
- **DON'T PUT THE CODE OF THIS PROJECT ON INTERNET !**
  I did spend a lot of time setting it and tuning it for pedagogical purpose.

  If you give it away, it will ruin all my attempts to teach next students something !