Advanced Computer Programming

Lecture 1

Alberto Ceselli
alberto.ceselli@unimi.it

Dipartimento di Informatica
Università degli Studi di Milano

October, 3rd, 2014

Is it worth to improve coding skills?

Is it really worth to improve coding skills?
http://www.bls.gov/ooh/computer-and-information-technology/computer-programmers.htm
http://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm
http://www.computerworld.com/article/2502348/it-management/it-jobs-will-grow-22–through-2020–says-u-s-.html

Which kind of language?

Which kind of language / coding style / coding paradigm?
http://bluebones.net/evolution/evo-prog-lang.png
i.e. ... it depends on your application!

Syllabus

Part A: enhancing programming paradigms

1- Data driven programming:
- performing complex tasks on data: functional programming (clojure)
- when data determine the flow of the program (awk)

2- Scaling up: from procedures and data structures to Generics (Java)
- ADTs; the issue of polymorphism
- Parametric types and Parametric programming
- Object Orientation (review)
- Generic programming

3- Meta programming (Java)
- Binding issues
- Delegates
- Reflection, Reification, Code annotation
- Applications

4- Exploiting concurrency (Java)
- shared memory concurrent programming

Syllabus

Part B: Programming in the small

- Realtime programming issues
- Programming in embedded systems
- Programming mobile devices

Part C: Programming in the large (Component based programming)

- Fundamentals of component based programming and pattern-based design
- Case study 1: creational patterns
- Case study 2: structural patterns
- Case study 3: behavioural patterns

Why data-driven programming?

Found on the net: "reformatting 1GB of textual feature data into a form Matlab and R can read"

Language	Time (min:sec)	Speed (vs. gawk)	Lines of code
mawk	1:06	7.8x	3
java	1:20	6.4x	32
c-ish c++	1:35	5.4x	42
python	2:15	3.8x	20
perl	3:00	2.9x	17
nawk	6:10	1.4x	3
c++	6:50	1.3x	48
ruby	7:30	1.1x	22
gawk	8:35	1x	3

Imperative VS Functional

Imperative programming:

- We have a *Von Neumann machine* to manage
- it has a *status* (memory and registry values)
- we perform computations by changing the status of the machine
- we give instructions (or *commands*) to be taken in sequence according to a control flow, and we manage this flow with statements for iteration, selection etc.
- to modify memory cell values (and therefore the status of the machine), we modify values of variables

Imperative VS Functional

Functional programming:

- We define *functions* that produce the desired output, starting from their input
  - computation means to apply functions to their arguments
- The functions are defined as mathematical ones, using more *conditions* and *recursion*, than sequences of instructions
- Moreover, functions have no *side effects*, nor assignments
  - E.g.: given the function $f(x) = x + 1$, compute $f(x)$ is not changing a "status":
    - f is not changing an internal status, $f(3)$ is always 4
    - f in not changing an external status, not opening sockets nor saving to disk

Imperative VS Functional

No status ...

- no loops: only *recursion* is used in place of *any other iteration statement*
- no assignments and no variables, no memory management
- only function definitions, and applications of functions to data (no side effects)

Recursive thinking ...

Examples:

- factorial
- Fibonacci's sequence
- sum and power

Functional programming and functional languages

Why to design dedicated languages to functional programming?

- syntax: to have code that resembles as much as possible mathematical functions
- semantics: to allow coding *any type of function*
- ... without restrictions on type or number of parameters (e.g. having functions taking functions as arguments, or returning functions, or returning tuples of values)

Clojure

Clojure: a functional programming language targeting the JVM
official website: http://clojure.org/
tutorial: http://java.ociweb.com/mark/clojure/article.html

‘Hands on’ session: Clojure

From basics to XML files semantic equivalence using Clojure