CMSC 330: Organization of Programming Languages

http://www.cs.umd.edu/~atif/Teaching/Fall2007

Introduction

Instructor: Atif M Memon

TAs: Guilherme Fonseca, Michael Lam, Xun Yuan

Calendar / Course Overview

- Final Exam (25%)
- 2 midterms (30%)
- One project every few weeks (5 total) (40%)
  - Project 1 - Write a web server log analysis tool.
    - Will be posted next week.
  - Project 2 - Write a unit testing framework in Ruby.
  - Project 3 - Write some OCaml code.
  - Project 4 - NFAs and reg exps in OCaml.
  - Project 5 - Write a threaded bank simulation.
- Two homework assignments (one before each exam) (5%)

- Ruby
- OCaml
- Java

Academic Integrity

- All written work (including projects) must be done on your own
- Work together on practice questions for the exams
- Work together on high-level project questions
  - Never see another student's code
  - If unsure, ask instructor!

What if…?

Rules and Reminders

- Quiet cell phones
- Be on time
- Come to class and discussion section
- Laptops in class only if really needed
- Use lecture notes as the book
- Stay organized and ahead of your work
- Get help as soon as you need it (but not when you don't)
  - Office hours
    - http://www.cs.umd.edu/~atif/Teaching/Fall2007/office-hours.shtml
- Use internet resources

Syllabus

- Scripting Languages (Ruby)
- Regular expressions and finite automata
- Context-free grammars
- Functional programming (OCaml)
- Concurrency
- Object-oriented programming (Java)
- Environments, scoping, and binding
- Advanced Topics

Course Goal

Learn how programming languages "work"

- Broaden your language horizons
  - Different programming languages
  - Different language features and tradeoffs
- Study how languages are implemented
  - What *really* happens when I write x.foo(…)?
- Study how languages are described
  - Mathematical formalisms

All Languages Are Equivalent

- A language is *Turing complete* if it can compute any function computable by a Turing Machine
- Essentially all general-purpose programming languages are Turing complete
- Therefore this course is useless!

Why Study Programming Languages?

Introduce yourself to your neighbor(s) and together write down three of your own reasons…

Why Study Programming Languages?

- Using the right language for a problem may be easier, faster, and less error-prone
  - Programming is a human activity
  - Features of a language make it easier or harder to program for a specific application
- To make you better at learning new languages
  - You may need to add code to a legacy system
    - E.g., FORTRAN (1954), COBOL (1959), …
  - You may need to write code in a new language
    - Your boss says, "From now on, all software will be written in {Ada/C++/Java/…}"

- You may think Java is the ultimate language, but if you are still programming or managing programmers in 20 years, they probably won't be programming in Java!

Why Study Programming Languages?

- To make you better at using languages you think you already know
  - Many "design patterns" in Java are functional programming techniques
  - Understanding what a language is good for will help you know when it is appropriate to use

Changing Language Goals

- 1950s-60s: Compile programs to execute efficiently
  - Language features based on hardware concepts
    - Integers, reals, goto statements
  - Programmers cheap; machines expensive
    - Keep the machine busy
- Today:
  - Language features based on design concepts
    - Encapsulation, records, inheritance, functionality, assertions
  - Processing power and memory very cheap; programmers expensive
    - Ease the programming process

Language Attributes to Consider

- Syntax -- What a program looks like
- Semantics -- What a program means
- Implementation -- How a program executes

Imperative Languages

- Also called *procedural* or *von Neumann*
- Building blocks are functions and statements
  - Programs that write to memory are the norm

```
int x = 0;
while (x < y) x := x + 1;
```

  - FORTRAN (1954)
  - Pascal (1970)
  - C (1971)

Functional Languages

- Also called *applicative* languages
- No or few writes to memory
  - Functions are higher-order

```
let rec map f = function [] -> []
                       | x::l -> (f x)::(map f l)
```

  - LISP (1958)
  - ML (1973)
  - Scheme (1975)
  - Haskell (1987)
  - OCaml (1987)

Logical Languages

- Also called *rule-based* or *constraint-based*
- Program consists of a set of rules
  - "A :- B" – If B holds, then A holds

```
append([], L2, L2).
append([X|Xs],Ys,[X|Zs]) :- append(Xs,Ys,Zs).
```

  - PROLOG (1970)
  - Various expert systems

Object-Oriented Languages

- Programs are built from objects
  - Objects combine functions and data
  - Often have classes and inheritance
  - "Base" may be either imperative or functional

```
class C { int x; int getX() {return x;} … }
class D extends C { … }
```

  - Smalltalk (1969)
  - C++ (1986)
  - OCaml (1987)
  - Java (1995)

Scripting Languages

- Rapid prototyping languages for "little" tasks
  - Typically with rich text processing abilities
  - Generally very easy to use
  - "Base" may be imperative or functional; may be OO

```
#!/usr/bin/perl
for ($j = 0; $j < 2*$lc; $j++) {
     $a = int(rand($lc));
…
```

  - sh (1971)
  - perl (1987)
  - Python (1991)
  - Ruby (1993)

"Other" Languages

- There are lots of other languages around with various features
  - COBOL (1959) – Business applications
    - Imperative, rich file structure
  - BASIC (1964) – MS Visual Basic widely used
    - Originally an extremely simple language
    - Now a single word oxymoron
  - Logo (1968) – Introduction to programming
  - Forth (1969) – Mac Open Firmware
    - Extremely simple stack-based language for PDP-8
  - Ada (1979) – The DoD language
    - Realtime
  - Postscript (1982) – Printers- Based on Forth
  - …