Introduction to Programming Languages Concepts

Synopsis

A. Course objectives (brief)

B. Housekeeping

C. Course objectives (detail)

D. Syllabus

E. Study tips

F. The PL universe

Course Objectives

1. Appreciate the relationship between most mainstream programming languages
2. Further your programming skills
3. Make learning of new programming languages easier in the future
4. Understand the workings of compilers and interpreters
5. Understand aspects of large-scale software development, and how they can be tackled at the programming language level
6. Enhance communication skills

Housekeeping

1. Lecturer's details
2. Teaching modes
3. IVLE
4. Textbooks
5. Assessment
6. Software

Lecturer's Details

- Name: Razvan Voicu
- email: `razvan@comp.nus.edu.sg`
- office phone: 6516 2732
- office: COM2-3-10
- Office hours: TBA
  - appointment by email, will always make time
  - usually in during afternoons
  - call office phone before dropping by my office

Etiquette

- Feel free to call me "Razvan"
  - Alternative: "Dr. Voicu"
  - Incorrect: "~~Dr. Razvan~~", "~~Voicu~~", "~~Prof~~", "~~Sir~~"
- Email openings
  - Correct: "Dear/Hi Razvan", "Dear/Hi Dr. Voicu"
  - Incorrect: "~~Dear/Hi Voicu~~", "~~Dear/Hi Dr. Razvan~~", "~~Dear Prof~~"
  - Always use correct name in opening (copy and paste if necessary)

Teaching Modes: Lectures

- Lecture notes will be made available — will contain supplementary readings
  - blog format
  - allow readers to give feedback and discuss
  - allows prompt corrections if necessary
- No simultaneous webcast
- Screen will be captured and recording will be uploaded to IVLE
- No particular textbook will be followed
  - references to the optional textbooks and various internet resources will be made whenever possible

Teaching Modes: Tutorials/Labs

- 2 hour classes, general set-up of a tutorial
  - Exercise set given out in advance
  - You are expected to prepare solutions and discuss them during class
- Venue: PC labs in I-cube
- Use machines in lab to run experiments and demos
  - Have your solutions ready (thumbdrive/internet repo)
  - Demo your solutions, tweak them, create interesting discussion points
- Screen-capture each tutorial for further revision

Teaching Modes: Homework

- 8 Problem Sets
- Multiple languages
- Many optional components
- Reinforcement of concepts taught in class and also exercises that require creative solutions
- Programming is a **skill** – it requires practice
  - homework exercises will provide plenty of opportunity for that
- Solving homework exercises: best preparation for exams

IVLE

- All course information
- Lesson plan
- Announcements and reminders
- Discussion forum
- General repository for materials
  - Lecture notes on separate site – only on intranet

    `http://razvan-course-z.comp.nus.edu.sg/cs2104`

    aliased as

    `http://cs2104notes.comp.nus.edu.sg/cs2104`
  - Discussion will be possible there as well
- Homework submission
- Gradebook

Optional Textbooks

Lecture notes are self-contained

- [Sebesta]
  "Concepts of Programming Languages", ISBN: 0-13-607347
- [Friedman]
  "Essentials of Programming Languages", ISBN: 0-262-06279-8
- [Tucker]
  "Programming Languages: Principles and Paradigms", ISBN: 007-125439-0
- [Watt]
  "Programming Language Design Concepts", ISBN: 0-470-85320-4
- [Sterling]
  "The Art of Prolog", ISBN: 0-262-19250-0
- [Krishnamurthi]
  "Programming Languages: Application and Interpretation" (available online, just Google the title)
- [van Roy]
  "Concepts, Techniques, and Models of Computer Programming" (available online, Google the title; also available in NUS's electronic library)

Optional Textbooks

- [SICP]
  "Structure and Interpretation of Computer Programs"
  (`http://mitpress.mit.edu/sicp/`)
- [O'Sullivan]
  "Real World Haskell"
  (`http://book.realworldhaskell.org/read/`)
- [Blum]
  "Professional Assembly Language" (ISBN: 0-7645-7901-0)
- [Kernighan]
  "The C Programming Language" (ISBN 0-13-110362-8, also in PDF, google the title)
- Documentation of various open-source languages, as found on their home pages
- Wikipedia

Assessment/Feedback

- Feedback on your performance: 40%
  - The homework and tutorials are meant to help you learn
  - Overall marks reward effort rather than knowledge

    no penalty for slow learners
  - Individual marks represent feedback on your performance in each exercise
  - Implemented as a **cap system**
  - Anybody who puts in a decent level of effort should attain the maximum
- Assessment: 60%
  - Mid-term: 10% (open book)
  - Final exam: 50% (open book)
  - Emphasize knowledge, rather than effort
  - Expect these to be rather tough

Cap System

- Homework: 35% cap
  - Each exercise will have an individual weightage
  - 1 exercise mark = 1 final mark out of 100 marks
  - Total marks accross all exercises: 60
  - Marks in excess of 35 will be forfeited and entered in contest
- Tutorials: 10% cap
  - Each contribution to the tutorial class will be credited with 1, 2, or 3 marks
  - 1 tutorial mark = 1 final mark out of 100 marks
  - Marks in excess of 10 will be forfeited and entered in contest
- Forum contributions: 5% cap
  - Good contributions will receive some credit
  - Marks in excess of 5 will be forfeited and entered in contest
- Overall: 40% cap
  - Forfeited marks entered in contest
  - Should be attainable by anybody who puts in decent level of effort

Course Objectives (detail)

1. Appreciate the relationship between most mainstream programming languages
2. Further your programming skills
3. Make learning of new programming languages easier in the future
4. Understand the workings of compilers and interpreters
5. Understand aspects of large-scale software development, and how they can be tackled at the programming language level
6. Enhance communication skills

Relationship between Languages

- Over 1000 languages in existence, with some 25 in relative widespread use
- Each has a different perspective on performing computation and on software deveolpment
- Sometimes it's easier to implement a solution in one language instead of another
- However, external constraints may impose a specific language, other than the one we choose, for our implementation
- Important question: how do we convert our solution from one language to another?

Further Your Programming Skills

- In writing a program, we have two parts
  - Problem solving — find a solution pattern or paradigm that applies
  - Expressing the solution in a programming language
- PLs are invented to provide simple way to express complicated solution patterns
- In studying PLs, we shall explore a multitude of solution patters
- This will help you build a repertoire of solution patterns, and ability to express them in various PLs

New Programming Languages

- In the universe of PL we have
  - a number of paradigms
  - a number of features
- Each of these "components" renders
  - certain expressive power w.r.t. a specific solution pattern
  - certain software development productivity features
- Most languages are made up of a combination of such "components"
- Studying the "components", and knowing the "components" a programming language is made of will make it clear how to apply a specific language to a given solution pattern

Workings of Compilers and Interpreters

- Semantics
  - Helps reason about program correctness
  - Indicates what kind of low-level instruction a program executes
- Execution time
  - Certain programming constructs take longer to execute than others
  - Some languages are inherently slower than others
- Memory usage
  - Some primitive operations do not have constant memory usage
  - Garbage collection sometimes adds extra overhead
- Programming platform
  - Understand the difference between language primitive and standard library procedure/method
  - Understand the role of compiler, linker, and loader

Large Scale Software Development

- Cost-effective large-scale software development requires modular development, as well as compliance with a complicated set of rules
- Cooperation requires communication, but communication is costly, and desirable to be minimized
- Module systems implement efficient communication and cooperation between team members
- As for the rules, without an enforcement system, programmers often "stray", leading to increased costs later in the process
- Some languages try to enforce such rules, leading to lower development costs

Communication Skills

- Every discipline has its formative values, developing skills that are outside the profession.
- Study of programming languages makes one pay more attention at what information needs to be communicated, and the format in which this information must be expressed in order to be well understood.

Syllabus

1. Introduction
   - Concepts, classifications, bird's eye view of PL universe
2. Assembly languages, and relationship to C
3. Languages, grammars, regular expressions
4. Data types and expressions
5. Sequential programming; semantics
6. Stateful and non-stateful programming
7. Procedural abstraction; higher order programming
8. Lazy evaluation
9. Types
10. Object oriented programming

Syllabus

11. Exception handling and event handling
12. Modules, components, re-usability, robustness
13. Embedded languages
14. Concurrency
15. Rule-based programming
16. Constraint programming
17. Meta-circularity
18. Parser and lexer generators
19. Domain specific languages

Languages Covered

- Pentium assembly language
  - `gas` (from linux/cygwin binutils)
  - Linux: use "add software" to add the "binutils" package
  - Windows: install cygwin (from cygwin.com), and then the "binutils" package
  - Windows (alternative): install MinGW
- C/C++
  - Install GCC in Linux/Cygwin
  - Windows alternative: Install Visual Studio Express Edition
- Java
  - All resources at:
    `http://www.oracle.com/technetwork/java/index.html`
- C#
  - Install Visual Studio Express Edition
- Scheme
  - `racket-lang.org`

Languages Covered

- Prolog/CLP
  - `http://www.swi-prolog.org/`
  - `http://eclipseclp.org/`
- Ocaml
  - `http://caml.inria.fr/`
- Haskell
  - `haskell.org`
- Python
  - `python.org`
- Javascript
  - Available in any browser
- PHP
  - `http://php.net`

Languages Covered

- SQL
  - `http://mysql.com/` (community edition)
  - `http://postgresql.org/`
- Oz
  - `http://www.mozart-oz.org/`
- Go (The Google language)
  - `golang.org`
- Ruby
  - `http://www.ruby-lang.org/en/`
- Bash
  - Available in Linux/Cygwin
- AutoHotKey
  - `http://www.autohotkey.com/`

Study Tips

- Why is it difficult to learn programming?
- Beginner's misconception
- Practice, practice, practice !
- Dare bug the teacher!
- Learn by association

Why is it Difficult to Learn Programming?

- The programming process proceeds in two phases:
  - Problem solving
  - Expressing the solution into a language
- The second phase is often overlooked in teaching programming
- In learning that, it is important to pay attention to **patterns**
- Patterns can only be learned by example
- The student has to be engaged in active learning
  - Extract patterns from examples
  - Something has to "click" in your mind, you must have an "aha" moment
  - The instructor can help by picking relevant examples, and answering questions
  - The student must keep asking questions till the "aha" moment

Beginner's Misconception

- "I need to read the full book on a PL before starting programming in that language
- Best way to learn programming
  - Decide on a project
  - Find an open source program that performs a similar task
  - Start hacking and modifying the open source program
  - Read up and learn on a need-to-know basis
  - Most importantly: **BE PERSISTENT!**

Practice, Practice, Practice!

- Programming is a skill
- Skills are built with practice
- You won't learn to swim or ride a bicycle by just reading about it
- Try solving the same problem in many ways
- Try solving the same problem in many languages
- Try to build an intuition about the relationship between languages in the process
- We will try to support this process by teaching translation schemes
  - Systematic ways of translating from one language into another

Dare Bug the Teacher!

- Remember, you should keep asking questions till you have the "aha" moment
- I'm always pleased to answer questions, it provides feedback on how I can improve my teaching
- It's always best if you do it in the forum, it benefits all of your colleagues
- It's always easiest if you can showcase an example of code whose behaviour you have doubts about; those kinds of doubts are the easiest to clear
  - It requires active learning, since you have to put an effort into finding that example...

Learn by association

A common reasoning pattern.

```
a = b + c*d ;
```

$\Longrightarrow$

```
mov eax,[_c]    ; REG_eax = c
imul [_d]       ; REG_eax *= d
add eax,[_b]    ; REG_eax += d
mov [_a],eax    ; a = REG_eax
```

```
a = (b+c) * d ;
```

$\Longrightarrow$ **?**

The PL Universe

1. What drives the development of PL
2. Short history
3. Programming paradigms

What Drives the Development of PLs?

- Commercial Languages: Software Productivity
  - Simple execution model, readable code
  - Fill a specific niche
  - Robustness: less bug prone
  - Modularity: easy collaboration
  - Reusability: libraries can be reused in new projects
  - Maintainability: adding new features do not require lots of changes
  - Security: mobile code can be trusted
  - Enduring paradigm: object orientation
  - New driver: customer loyalty (e.g. Java, C#)

What Drives the Development of PLs?

- Research Languages: Development of New Paradigms
  - Novel ways of exprssing computation
  - Complicated execution model
  - Complicated problems get simple solutions in new paradigms
  - Useful for "proof of concept" projects
  - Inherently slow
  - Elegant, "puristic" approach

Short History

- Evolution of commercial languages based on
  - Power of hardware
  - Software development needs
- Research languages
  - What is percieved that computer science should be able to solve

    Artificial Intelligence, Logic, Theorem Proving, Algebra, Constraint Solving, etc

Short History: Commercial Languages

- Assembly languages (ealry 1950s)
- Fortran (late 1950s)
- Algol (1960s) – precursor of PL/1, Modula, Pascal
- Cobol (1960s)
- C (1973) – Portable assembly language
- Ada (1970s) – introduces concurrency
- SQL (late 1970s)
- C++ (1985) – Borrowed OOP from Smalltalk
- Java (1995)
- Perl, Python, Javascript, PHP, VB (1990s)
- C# (2000)
- Go (2009) – Fine-grain concurrency in a compiled language

Short History: Research Languages

- Lisp (1958) – LISt Processing
  - Garbage collection, meta-circularity
  - Many variants, including Scheme
- Prolog (1972) – Programming in Logic
  - Rule based programming, dynamic syntax modification, meta-circularity
- Smalltalk (1972) – Object orientation
- ML (1970s) – typed functional programming
  - Precursor of Standard ML, Ocaml
- CLP (1980)– Constraint Logic Programming
- Haskell (1990)
  - Algebraic programming, lazy functional programming
- Oz (1991)
  - Multiparadigm, fine-grained concurrency

Programming Paradigms

- Imperative Programming
- Logic Programming
- Functional programming
- Object-oriented programming
- Constraint programming
- Event-driven programming
- Aspect-oriented programming (not covered)
- Orthogonal paradigmatic features
  - Typing
    - Statically/dynamically typed
    - Strongly/weakly typed
  - Strictness: strict/lazy
  - Concurrency: fine/coarse grain