

COMS W4115

Programming Languages and Translators

Lecture 1: January 23, 2013

Introduction to PLT

1. Teaching Staff

Instructor

Professor Alfred V. Aho

<http://www.cs.columbia.edu/~aho>

aho@cs.columbia.edu

513 Computer Science Building

Office hours: Mondays and Wednesdays 1:00-2:00pm

Course webpage: <http://www.cs.columbia.edu/~aho/cs4115>

Courseworks website: <https://courseworks.columbia.edu>

Piazza bulletin board: <https://piazza.com/class#spring2013/comsw4115>

Lectures on Mondays and Wednesdays, 2:40-3:55pm, 535 Mudd

TAs

Karan Bathla

kb2658@columbia.edu

Office hours: Mondays & Tuesdays 4:00-5:00

TA Room: 122 Mudd

Melanie Kambadur

melanie@cs.columbia.edu

Office hours: Thursdays 10:00-12:00

TA Room: 122 Mudd

Jared Pochtar

jrp2181@columbia.edu

Office hours: Fridays 3:00-5:00

TA Room: 122 Mudd

Maria Taku

mat2185@columbia.edu

Office hours: Tuesdays and Thursdays 12:30-2:30

TA Room: 122 Mudd

2. Course Objectives

- You will learn about the syntactic and semantic elements of modern programming languages.
- You will learn the important algorithms used by compilers to translate high-level source languages into machine and other target languages.
- You will learn about imperative, object-oriented, functional, logic, scripting languages, and parallel languages.
- A highlight of this course is a semester-long programming project in which you will work in a small team to create and implement an innovative little language of your own design.
- You will learn computational thinking and good software engineering practices.
- The concepts, techniques, and tools that you will learn in this course have broad application to many areas of computer science and software development outside of programming languages and compilers.

3. Course Syllabus

- Computational thinking
- Kinds of programming languages
- Principles of compilers
- Lexical analysis

- Syntax analysis
- Tools for constructing compilers
- Syntax-directed translation
- Semantic analysis
- Run-time organization
- Intermediate code generation
- Code generation
- Code optimization
- Parallel and concurrent programming languages

4. Textbooks and References

- The course text is
 - Alfred V. Aho, Monica Lam, Ravi Sethi, and Jeffrey D. Ullman
Compilers: Principles, Techniques, and Tools, Second Edition
Pearson Addison-Wesley, 2007
- Other good references are
 - Andrew W. Appel
Modern Compiler Implementation in Java, second edition
Cambridge University Press, 2002
 - Keith D. Cooper and Linda Torczon
Engineering a Compiler, Second Edition
Morgan Kaufmann, 2012
 - Steven S. Muchnick
Advanced Compiler Design and Implementation
Morgan Kaufmann, 1997
 - Michael L. Scott
Programming Language Pragmatics, Third Edition
Morgan Kaufman, 2009
 - Robert W. Sebesta
Concepts of Programming Languages, Tenth Edition
Pearson/Addison-Wesley, 2012
- [Also see Professor Stephen Edwards' PLT website. Well worth a look!](#)

5. Course Requirements, Grading, and Late Policy

- Homework (10% of final grade)
- Midterm (20% of final grade): Wednesday, March 13, 2013
- Final (30% of final grade): Monday, May 6, 2013
- Course project (40% of final grade): project has team and individual components
- All assignments can be handed in one week after they are due for 50% credit.

6. Project Requirements

- Form a team of five. Teams should be formed by Monday, February 4, 2013.
- Design a new innovative little language
- Build a compiler or interpreter for it.

- Project deliverables and due dates:
 1. Feb 27: Language white paper. See Section 1.2 of <http://www.oracle.com/technetwork/java/index-136113.html> for a sample white paper on Java.
 2. Mar 27: Language tutorial and reference manual.
 - See Chapter 1 of K&R for a sample language tutorial.
 - See Appendix A of K&R for a sample language reference manual.
- May 14-16: Final project report and demo to teaching staff.
- Start to form project teams of five right away. Elect a
 - Project manager
 - Language guru
 - System architect
 - System integrator
 - Verification and validation person

7. Programming Languages

- A programming language is a notation for specifying computational tasks that a person can understand and a computer can execute.
- Every programming language has a syntax and a semantics.
 - The syntax specifies how a concept is expressed.
 - The syntax is often defined using a (context-free) grammar.


```
statement -> while ( expression ) statement
```
 - The semantics specifies what the concept means or does.
 - Semantics can be specified operationally, axiomatically or denotationally.
- Ambiguity
 - "Time flies like an arrow."
- Domains of application
 - Scientific
 - Fortran
 - Business
 - COBOL
 - Artificial intelligence
 - LISP
 - Systems
 - C
 - Web
 - Java
 - General purpose
 - C++

8. Kinds of Languages

- Imperative
 - Specifies how a computation is to be done.
 - Examples: C, C++, C#, Fortran, Java
- Declarative
 - Specifies what computation is to be done.
 - Examples: Haskell, ML, Prolog
- von Neumann
 - One whose computational model is based on the von Neumann architecture.
 - Basic means of computation is through the modification of variables (computing via side effects).
 - Statements influence subsequent computations by changing the value of memory.
 - Examples: C, C++, C#, Fortran, Java
- Object-oriented
 - Program consists of interacting objects.
 - Each object has its own internal state and executable functions (methods) to manage that state.
 - Object-oriented programming is based on encapsulation, modularity, polymorphism, and inheritance.
 - Examples: C++, C#, Java, OCaml, Simula 67, Smalltalk
- Scripting
 - An interpreted language with high-level operators for "gluing together" computations.
 - Examples: AWK, Perl, PHP, Python, Ruby
- Functional

- One whose computational model is based on the recursive definition of functions (lambda calculus).
- Examples: Haskell, Lisp, ML.
- Parallel
 - One that allows a computation to run concurrently on multiple processors.
 - Examples
 - Libraries: POSIX threads, MPI
 - Languages: Ada, Cilk, OpenCL, Chapel, X10
 - Architecture: CUDA (parallel programming architecture for GPUs)
- Domain specific
 - Many areas have special-purpose languages to facilitate the creation of applications.
 - Examples
 - YACC for creating parsers
 - LEX for creating lexical analyzers
 - MATLAB for numerical computations
 - SQL for database applications
- Markup
 - Not programming languages in the sense of being Turing complete, but widely used for document preparation.
 - Examples: HTML, XHTML, XML

9. Influential Languages

- 1950s: assembler, Cobol, Fortran, Lisp
- 1960s: Algol60, Basic, Simula67
- 1970s: C, ML, scripting languages, application-specific languages
- See [TIOBE Index](#) for their list of this month's 100 most popular programming languages.
 - TIOBE top ten for January 2013: C, Java, Objective-C, C++, C#, PHP, Visual Basic, Python, Perl, Ruby

10. Language Design Issues

- Application domain
 - Exploit domain restrictions for expressiveness and performance.
- Computational model
 - Choose a model that encourages simplicity and ease of expression.
 - Incorporate a few primitives that can be elegantly combined to solve large classes of problems.
- Abstraction mechanisms
 - Abstractions should foster reuse and be suggestive of solutions.
- Type system
 - Type systems can help reliability and security of programs.
- Usability
 - Language design should promote readability, writability, and efficiency.

11. To Do

1. Start forming your project team immediately. Teams should be in place by Feb 4, 2013.
2. Use Courseworks discussion board (<https://courseworks.columbia.edu>) to publicize your interests.
3. Contact Maria Taku (mat2185@columbia.edu) for help forming or finding a team.
4. Give your language a name.

12. Reading Assignment

- ALSU: Chapters 1 and 2