

# 198:314 Principles of Programming Languages Fall 2007

<http://remus.rutgers.edu/cs314/f2007/ryder>

Professor Barbara G. Ryder  
CoRE 311, 732-445-6430 x3699  
ryder@cs.rutgers.edu  
<http://www.cs.rutgers.edu/~ryder>

## Introduction

- **Administrivia**
- **Why study PLs?**
- **Translation of programs for execution, Scott Ch 1**
- **History of Programming Languages**

## 198:314 Fall 2007

- **Class webpage**

<http://remus.rutgers.edu/cs314/f2007/ryder/index.html>

- Look at **Course Information** for course rules, grading, exam policy, etc.
- Read about **Academic Integrity**
- Review posted **Syllabus** (class Syllabus shows related readings in Scott text) and look at important dates
- Check **Projects** and **Homeworks** for assignments
- Visit **Useful Websites** for additional info on PLs we will study

## 198:314 Fall 2007

- **Three programming projects will be posted and submitted for grading electronically**
- **Midterm Exam on October 23, 2007 and Final Exam on December 19, 2007**
- **Lecture notes webpage**
  - <http://remus.rutgers.edu/cs314/f2007/ryder/lectures/>
  - Lecture notes available online in PDF by noon of day of class (and hopefully before)
  - Slides posted in 2-up format; to save paper print double-sided
  - ***Recitation attendance and participation counts in your final grade!***

## Syllabus

- <http://remus.rutgers.edu/cs314/f2007/ryder/syll.html>
- Introduction
- Formal languages - RE's, BNF, context-free grammars, parsing
- Logic programming (Prolog)
- Basic semantics
- Functional programming
- Parameter passing
- Scripting languages
- Types
- Parallel programming, threads and monitors

## Course Goals

- To gain understanding of basic structures of programming languages
  - Types, control structures, naming conventions and binding, memory management, etc.
- To study different language paradigms
  - To ensure an appropriate language is selected for a task
  - Functional, logic, object-oriented, scripting
- To make learning new programming languages easier by identifying shared features

## What is a programming language?

“a language intended for use by a **person** to express a **process** by which a **computer** can solve a problem”

--Hope and Jipping

“a set of conventions for communicating an algorithm”

--E. Horowitz

“ the art of programming is the art of organizing complexity”

--E. Dijkstra, 1972

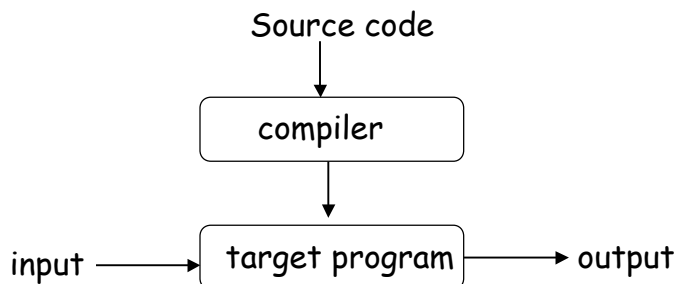
## Why learn more than one PL?

- Each language paradigm encourages thinking about a problem in a particular manner
  - Finding a natural match between problem and PL
- Somewhat different functionality supplied by different paradigms
- Computer professionals must be multi-lingual
  - PLs change over time as computer architecture and computing model changes
  - Specific applications sometimes result in specialized PLs
  - Need to understand each PL's functionality and limitations
  - Also need to understand some things about PL implementation, in order to use the PL effectively

## Translation

Scott Ch 1.4

- **Compilation:** translation of a program written in a high-level PL into a form that is executable on the machine



Introduction 1, CS314 Fall 2007 ©, B6 Ryder

9

## Translation

- **Interpretation:** a program is translated and executed one statement at a time



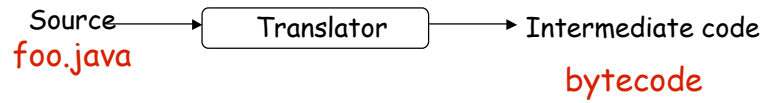
- **Most PL systems are a mixture of compilation and interpretation**
  - Interpreted: Java, Scheme, Prolog
  - Compiled: Fortran, C, C++

Introduction 1, CS314 Fall 2007 ©, B6 Ryder

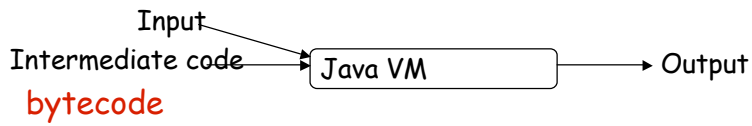
10

# Interpretation (Java)

> javac foo.java



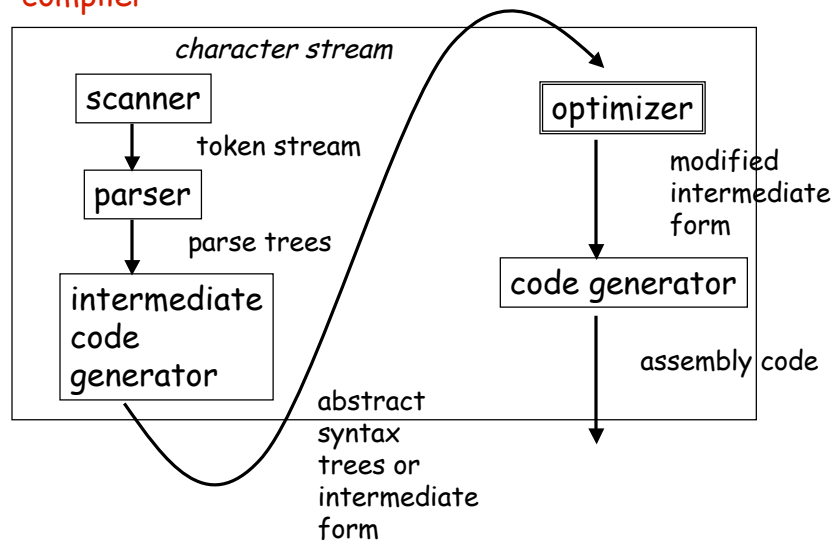
> java foo



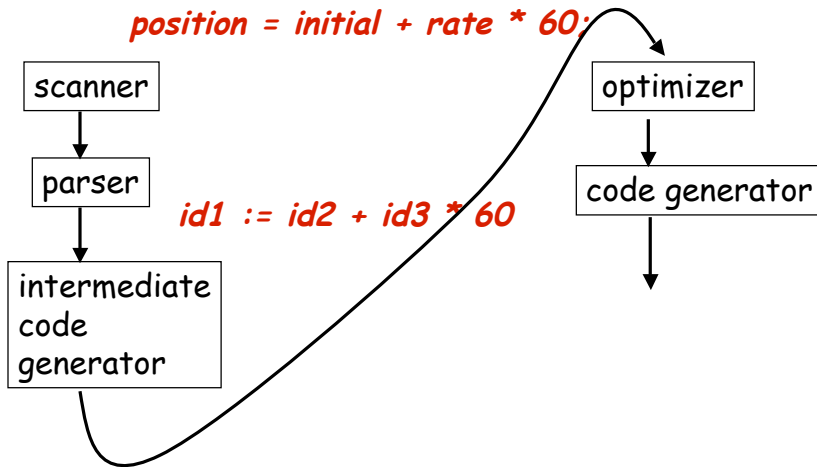
# Compilation

Scott Ch 1.6

compiler



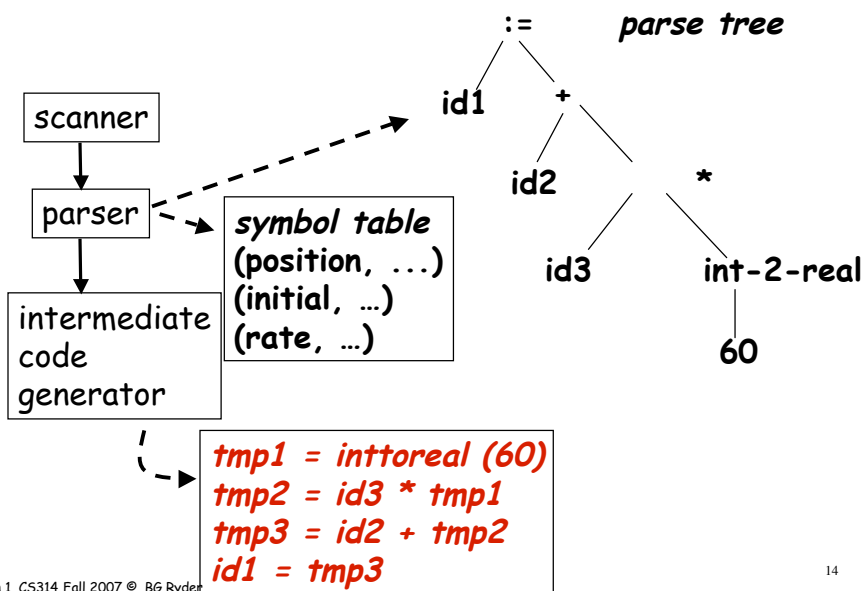
# Compilation



Introduction 1, CS314 Fall 2007 ©, B6 Ryder

13

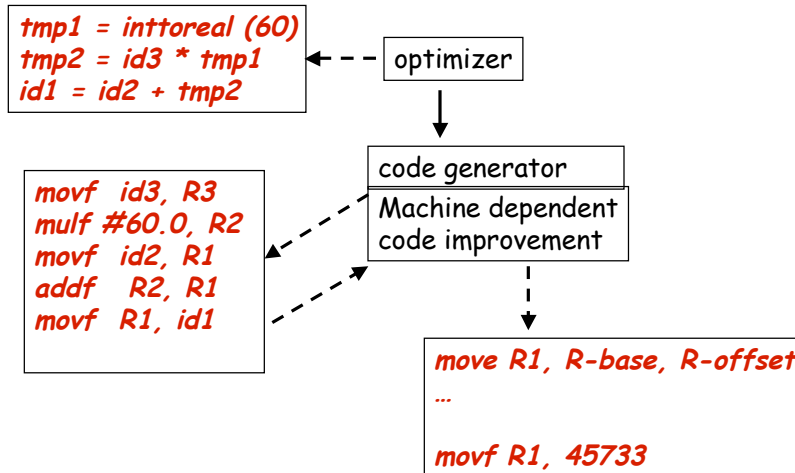
# Compilation



Introduction 1, CS314 Fall 2007 ©, B6 Ryder

14

# Compilation



Introduction 1, CS314 Fall 2007 ©, B6 Ryder

15

# History of PLs

ACM SIGPLAN HOPL conferences

- **Prehistory**
  - 300 B.C. Greece, Euclid invented the greatest common divisor algorithm - *oldest known algorithm*
  - ~1820-1850 England, Charles Babbage invented 2 mechanical computational devices
    - difference engine
    - analytical engine
    - Countess Ada Augusta of Lovelace, *first computer programmer*
  - Precursors to modern machines
    - 1940's United States, ENIAC developed to calculate trajectories

Introduction 1, CS314 Fall 2007 ©, B6 Ryder

16

## History of PLs - 2

- 1950's United States, first high-level PLs invented
  - **Fortran** 1954-57, John Backus (IBM on 704) designed for numerical scientific computation
    - fixed format for punched cards
    - implicit typing
    - only counting loops, if test versus zero
    - only numerical data
    - 1957 optimizing Fortran compiler translates into code as efficient as hand-coded

## History of PLs - 3

- **Algol60** 1958-60, designed by international committee for numerical scientific computation [Fortran]
  - block structure with lexical scope
  - free format, reserved words
  - while loops, recursion
  - explicit types
  - BNF developed for formal syntax definition
- **Cobol** 1959-60, designed by committee in US, manufacturers and DoD for business data processing
  - records
  - focus on file handling
  - English-like syntax

## History of PLs - 4

- **APL** 1956-60 Ken Iverson, (IBM on 360, Harvard) designed for array processing
- **LISP** 1956-62, John McCarthy (MIT on IBM704, Stanford) designed for non-numerical computation
  - uniform notation for program and data
  - new conditional control structure (COND)
  - recursion as main control structure
- **Snobol** 1962-66, Farber, Griswold, Polansky (Bell Labs) designed for string processing

## History of PLs - 5

- **PL/I** 1963-66, IBM designed for general purpose computing [Fortran, Algol60, Cobol]
  - user controlled exceptions
  - multi-tasking
- **Simula-67** 1967, Dahl and Nygaard (Norway) designed as a simulation language [Algol60]
  - data abstraction
  - inheritance of properties
- **Algol68** 1963-68, designed for general purpose computing [Algol60]
  - orthogonal language design
  - interesting user defined types

## History of PLs - 6

- **Pascal** 1969, N. Wirth(ETH) designed for teaching programming [Algol60]
  - 1 pass compiler
  - call-by-value semantics
- **Prolog** 1972, Colmerauer and Kowalski designed for Artificial Intelligence applications
  - theorem proving with unification as basic operation
  - logic programming
- **Recent**
  - **C** 1974, D. Ritchie (Bell Labs) designed for systems programming
    - allows access to machine level within high-level PL
    - efficient code generated

## History of PLs - 7

- **Clu** 1974-77, B. Liskov (MIT) designed for simulation [Simula]
  - supports data abstraction and exceptions
  - precise algebraic language semantics
  - attempt to enable verification of programs
- **Smalltalk** mid-1970s, Alan Kay (Xerox Parc), considered first real object-oriented PL, [Simula]
  - encapsulation, inheritance
  - easy to prototype applications
  - hides details of underlying machine
- **Scheme** mid-1970s, Guy Steele, Gerald Sussman (MIT)[Lisp]
  - Static scoping and first-class functions

## History of PLs - 8

- **Modula** 1977 N. Wirth (ETH), designed language for large software development [Pascal]
  - to control interfaces between sets of procedures or *modules*
  - real-time programming
- **Ada** 1979, US DoD committee designed as general purpose PL [Algol]
  - explicit parallelism - *rendezvous*
  - exception handling and generics (*packages*)
- **C++** 1985, Bjarne Stroustrup (Bell Labs) general purpose [C]
  - goal of type-safe object-oriented PL
  - compile-time type checking
  - templates

Introduction 1, CS314 Fall 2007 ©, B6 Ryder

23

## History of PLs - 9

- **SML** ~1990, R. Milner
  - Strict, polymorphic, functional PL with compile-time type inference and checking and modules
  - SML/NJ compiler (Bell Labs, Princeton)
- **Lua** ~1993 at Pontifical Catholic Univ of Rio de Janeiro
  - Scripting PL used extensively for programming games
  - Dynamically typed, uses tables as main data structure
- **Java** ~1995, J. Gosling (SUN)
  - Aimed at portability across platform through use of JVM - abstract machine to implement the PL
  - Aimed to *fix* some problems with previous OOPLs
    - ◆ multiple inheritance
    - ◆ static vs dynamic objects
  - Ubiquitous exceptions
  - Thread objects

Introduction 1, CS314 Fall 2007 ©, B6 Ryder

24

## History of PLs - 10

- **Python** ~1995
  - Object-oriented + functional aspects, good for fast prototyping, framework programming (large # of runtime libraries)
- **Haskell98** ~1997
  - Polymorphic, typed, lazy, purely functional
- **Php** ~1997
  - Embedded scripting language for dynamic webpage construction
- ...