Lecture 1: Overview

CSC 131
Fall, 2008

Kim Bruce
TAs: Sam Cunningham, Alejandro Lopez-Lago, & Marquis Wang

Do Languages Matter?

- Why choose C vs C++ vs Java vs Python ...
- What criteria to decide?
- Impact on programming practice
- SIGPLAN Education Board documents

Why Article

- Learn widely-applicable design & implementation techniques
- Creating new domain specific languages or virtual machines
- Learning new computational models and speeding language learning
- Choosing the right language

Provide Abstractions

- Data Abstractions:
  - Basic data types: ints, reals, bools, chars, pointers
  - Structured: arrays, structs (records), objects
  - Units: Support for ADT's, modules, packages
- Control Abstractions:
  - Basic: assignment, goto, sequencing
  - Structured: if...then...else, loops, functions
  - Parallel: concurrent tasks, threads, message-passing

PL's & Software Development

- Development process:
  - requirements
  - specification
  - implementation
  - certification or validation
  - maintenance
- Evaluate languages based on goals

Goals of Some older PL's

- Languages & their goals:
  - BASIC - quick development of interactive programs
  - Pascal - instruction
  - C - low-level systems programming
  - FORTRAN, Matlab - number-crunching scientific
- What about large-scale programs?
  - Ada, Modula-2, object-oriented languages

PL Choice

- Languages designed to support specific software methodologies.
- Language affect way people think about programming process.
- Hard for people to change languages if requires different way of thinking about process.

Minimum Requirements

- Natural
- Implementable
- Efficient
- Reliable
- Maintainable

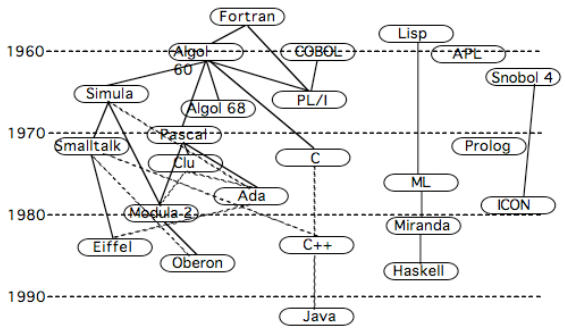
Paradigms
or whatever you want to call them

- Not crisp boundaries
  - Procedural
  - Functional
  - Logic or Constraint-programming
  - Object-oriented

History of PL's

- Machine language
  ⇒ Assembly language
  ⇒ High-level language
- Single highly-trained programmers
  ⇒ Teams of programmers

History of PLs



Course Goals

- Upon completion of course should be able to:
  - Quickly learn programming languages, & how to apply them to effectively solve programming problems.
  - Rigorously specify, analyze, & reason about the behavior of a software system using a formally defined model of the system's behavior.
  - Realize a precisely specified model by correctly implementing it as a program, set of program components, or a programming language.

Theory Matters!

Theory, Dynamic Execution, & Static Checking

- Program, when started w/ input can:
  - Terminate normally.
  - Terminate w/ error message
  - Run forever
- Effect of program:
  - partial function f: string → string ∪ {error}
  - Ex: g(x) = if odd(x) then 1 else g(x-1) + g(x-2)

Computable Functions

- f is computable iff exists program computing it.
- Church's thesis:
  Computability independent of language
  - Church-Kleene used lambda calculus
  - Turing used Turing machines
  - Godel-Kleene used partial recursive functions
  - ...

Halting Problem

- Is there is a program that will determine for any other program whether or not it will halt?
- *More precisely:*
  - Is there a program that when provided with another program and its input, will determine correctly 100% of the time whether or not the given program will halt on its input? (In particular, this program always halts telling me yes or no.)

Halting Problem

- Proved undecidable -- i.e., no algo to solve it.
- Other undecidable problems:
  - Will program eventually divide by 0?
  - Will program eventually dereference a null pointer?
  - Will program touch a particular piece of memory?
  - Will program ever print out 0?

Questions?