Programming Languages

Lecture 1: Introduction

Benjamin J. Keller

Department of Computer Science, Virginia Tech

Lecture Outline

- Preview
- History of Programming Languages
- Course topics

Programming Languages

- Representation that allows humans to communicate with a computer
- Abstraction of a machine language
- Different languages provide different abstractions

Kinds of Abstractions

- Data abstractions
  - primitive data types: numeric, characters, booleans, pointers
  - structured types: arrays, records
  - unit: support for ADTs, modules, packages, classes
- Control abstractions
  - basic: assignment, goto, sequencing
  - structured: conditionals, loops, procedures and functions
  - unit: separately compiled units, modules, packages, concurrent tasks

Defining Languages

- *Program* — sequence of symbols that specify a computation
- *Programming Language* — a set of rules that specify which sequences of symbols constitute a program, and what computation the program describes.
- *Syntax* — how program is formed
- *Semantics* — what program does
- *Pragmatics* — how language is used

Language Goals

- Languages designed for different purposes
  - Fortran — numerical computation
  - Pascal — teaching programming
  - C — low-level system programming
- Evaluate language with respect to goals of its design

Languages and Software Development

- Language design goals may conflict with large-scale system development
- Modern languages support specific design methodology/philosophy
  - functional design
  - object-based design
  - encapsulation
  - information-hiding
- Language influences how people think about programming

Language Paradigms

- Computer architecture led to *imperative*
- Others developed as different ways to solve problems
  - *Functional* — computation as expression evaluation, popular in AI and theoretical programming language communities.
  - *Logical* — computation as proving queries. AI and database query languages.
  - *Object-Oriented* — computation as manipulation of/or interaction of objects.

Language Requirements

- Universal — if a problem can be solved on a computer, then can solve it in the language
- Natural (expressive) — easy to express solutions
- Implementable
- Efficient — writing, compilation, or execution
- Reliable — ability to deal with exceptional behavior
- Maintainable — easy to make changes to programs

History

- Trend from machine language toward high-level languages
- Trend from single programmer to teams of programmers
- Early Languages
  - FORTRAN — John Backus, IBM, 1957. Goals: efficient compilation of solutions to numerical problems. Multiple revisions (latest in 90's).
  - Algol 60 — committee, 1960. Goals: numeric, block structure, recursion. Influential on most modern languages.
  - COBOL — committee, 1960. Goals: business data processing. Multiple revisions.

More Early History

- LISP — John McCarthy, MIT, 1962. Functional core. Goals: List processing, symbolic manipulation (AI applications).
- APL — Iverson, Harvard, 1960. Calculator language for array computations.
- SNOBOL 4 — Griswold, Bell Labs, 1966. Goals: string processing with pattern matching.
- PL/I — committee at IBM, 1967. Attempt to combine FORTRAN, COBOL, Algol 60. Large language difficult to learn and use.

More History

- Algol 68 — Algol with "orthogonal" elements. Elegant, but hard to understand.
- Simula 67 — simulation language based on Algol 60. Basis for most object-oriented languages.
- Pascal — Wirth, ETH-Zurich, 1971. Teaching language designed to support structured programming. Simple and elegant.

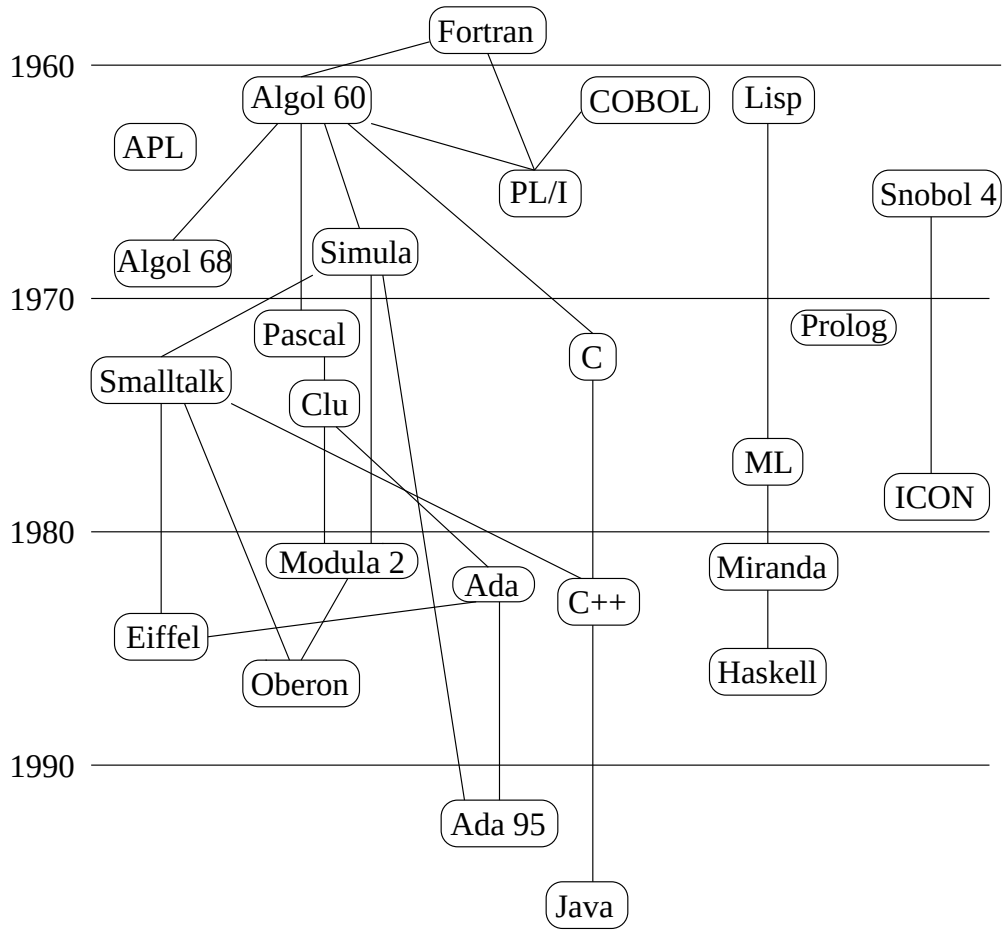
Further Developments

- Module support for abstract data types: Clu, Mesa, Modula-2, Ada
- Object-oriented: Smalltalk (70's), Eiffel, C++, Object Pascal, Java
- Functional: Scheme (80's), ML (70's), Miranda, Haskell
- Logical: Prolog, newer constraint languages

Fourth Generation Languages

- Languages in business applications
- Specialized packages of powerful commands with simplified syntax.

Family Tree of Important Languages



Course Topics

- Language features: modules, classes, exception handling, generic types, etc.
- Functional, object-oriented, logical and imperative paradigms.
- Support for reliable programming: abstraction, encapsulation, information hiding, polymorphism, higher-order operators.
- Support for concurrency.
- Formal definition of programming languages.
- Compilers and interpreters.
- Run-time behavior of programming languages