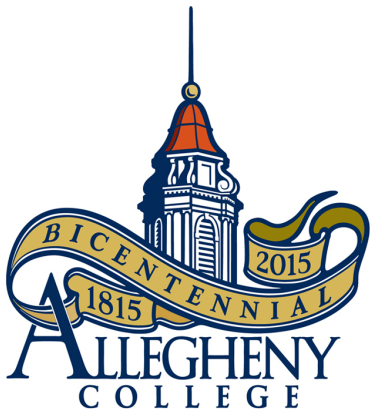
CS201 - Programming Languages

Background on PL's

Aravind Mohan

Allegheny College

September 1, 2021



For Starters

- **Reminder:** Lab 01 this afternoon at 3:00 PM.
- Last Week: Discussed the what and why's of CS-201
- **Next:** Some background on PL's, maybe introduce Compilers

There are different “levels” of PL

- Microcode languages
- Machine languages
- Assembly languages
- Intermediate/Internal PLs (e.g., Java byte code)
- High-level PLs (e.g., Java)

PL Paradigms

- Imperative
  - von Neumann (e.g., Fortran, Pascal, Basic, and C)
  - object-oriented (e.g., Smalltalk, Eiffel, C++, C#, and Java)
  - scripting languages (e.g., Bash, AWK, Perl, Python, JavaScript, and PHP)
- Declarative
  - functional (e.g., Scheme, ML, Lisp, and FP)
  - logic, constraint-based (e.g., Prolog, VisiCalc, and RPG)

Why are there so many PLs?

- Evolution: we've learned better ways of doing things over time.
- Socio-economic factors: proprietary interests and commercial advantage.
- Some PLs are oriented toward special purposes and hardware.
- Personal Interest, and some PLs are easier to learn and use.

What makes a PL successful?

- Some are easy to learn and implement (Python, Pascal, BASIC, LOGO, and Scheme).
- Some are more “expressive,” easier to use once fluent, and “powerful” (C, Common Lisp, APL, Algol-68, and Perl).

- Some can compile to very good (fast/small) code (Fortran).
- Some have the backing of a powerful sponsor (COBOL, and Visual Basic)
- Some enjoyed wide dissemination at minimal cost (Java, and Pascal).

Why do we have PLs? What is a PL for?

- A PL (or any language) helps determine the way you think, express algorithms, and solve problems.
- Some PLs try to match the user's point of view and others match the developer's point of view.
- They are an abstraction of a machine or virtual machine.

Why study PLs? (1 of 6)

- Knowing multiple PLs makes it easier to choose an appropriate PL:
  - C versus C++ versus Modula-3 for systems programming
  - Fortran versus APL versus Ada for numerical computations
  - C versus Ada for embedded systems
  - Common Lisp versus Scheme versus ML for symbolic data manipulation
  - Java versus C/CORBA versus C/COM for networked programs

Why study PLs? (2 of 6)

- Knowing multiple PLs makes it easier to learn new PLs, because many are similar. This is true for natural languages, too.
- PL concepts have even more similarity. For example, most PLs have iteration, recursion, and abstraction. They might only differ in syntax and semantic details.

Please take a look at the **triangle** directory in the activity folder.

Why study PLs? (3 of 6)

- It helps you use your PL more effectively, and understand its obscure features. For example:
  - In C/C++, you can understand unions, arrays, pointers, separate compilation, varargs, and exception handling.
  - In Lisp/Scheme, you can understand first-class functions/closures, streams, exception handling, and symbol internals.

Why study PLs? (4 of 6)

- It helps you understand implementation costs and choose between alternative ways of doing things, based on knowledge of what will be done underneath. For example:
  - Use `x<<1` instead of `x*2`, or `x*x` instead of `x**2`.
  - Use C pointers or Pascal's `with` statement, to factor address calculations.
  - Avoid call by value with large data items in Pascal.
  - Avoid call by name in Algol 60.
  - Choose between computation and table lookup.

Why study PLs? (5 of 6)

- It helps you figure out how to do things in PLs that don't support them explicitly. For example:
  - Old Fortran lacks suitable control structures, but you can use gotos, comments, and programmer discipline.
  - Some PLs lack recursion (e.g. CSP), but you can write a recursive algorithm and use mechanical recursion elimination.

Why study programming languages? (RECAP)

- Knowing multiple PLs is good.
  - Every PL does not have every feature.
  - There are different paradigms.
  - Different PLs are good for different tasks.
- Knowing the theory behind their design is good.

Reading Assignment

- PLP Chapter 01 - Section 1.1 to 1.3