CS 3304 Comparative Languages Fall 2000

Lenwood S. Heath
McB 638: MW 1:15–3:00
heath@cs.vt.edu
http://courses.cs.vt.edu/~cs3304/Fall00/

What Will You Learn?

- Survey of programming paradigms, including representative languages
- Language definition and description methods
- Overview of features across all languages
- Implementation strategies

Semester Outline

Part I: Paradigms and Description

- Introduction and Language Evaluation
- Programming Language Paradigms
- History and Evolution
- Imperative, Functional, Logic, Object-Oriented
- Syntax and Semantics

Semester Outline (cont.)

Part II: Features and Implementation

- Names and Typing
- Data Types
- Expressions and Assignment
- Control Structures
- Subprograms
- Abstract Data Types (ADTs)

Reasons to Study Concepts of PLs

Why bother with this stuff anyway?

- Increased capacity to express programming concepts
- Improved background for choosing appropriate languages
- Increased ability to learn new languages
- Understanding the significance of implementation
- Increased ability to design new languages
- Overall advancement of computing

Chapter 1: Introduction

- Language evaluation criteria
- Main implementation methods
- Trade–offs
- Influences on language design
- Programming paradigms

Evaluating A Language

- 4 main criteria:
  - Readability
  - Writability
  - Reliability
  - Cost
- Are there others?

Evaluation: Readability

- The most important criterion
- Overall simplicity
  - Too many features is bad
  - Multiplicity of features is bad
- Orthogonality
  - Makes the language easy to learn and read
  - Meaning is context independent
- Control statements
- Data type and structures
- Syntax considerations

Evaluation: Writability

- Factors:
  - Simplicity and orthogonality
  - Support for abstraction
  - Expressivity

Evaluation: Reliability

- Factors:
  - Type checking
  - Exception handling
  - Aliasing
  - Readability and writability

Evaluation: Cost

- Categories
  - Programmer training
  - Software creation
  - Compilation
  - Execution
  - Compiler cost
  - Poor reliability
  - Maintenance
- Other criteria: portability, generality, well–definedness

Implementation Methods

- Compilation
  - Translate high–level program to machine code
  - Slow translation
  - Fast execution
- Pure interpretation
  - No translation
  - Slow execution
  - Becoming rare
- Hybrid implementation systems
  - Small translation cost
  - Medium execution speed

Language Design Trade-offs

- Reliability versus cost of execution
- Writability versus readability
- Flexibility versus safety

Primary influences on language design

1. Computer architecture
   - We use imperative languages, at least in part, because we use von Neumann machines
2. Programming methodologies
   - 1950s and early 1960s: Simple applications; worry about machine efficiency
   - Late 1960s: People efficiency became important; readability, better control structures
   - Late 1970s: Data abstraction
   - Middle 1980s: Object-oriented programming

Language Categories

Programming paradigms:

- Procedural/Imperative
- Functional/Applicative
- Logic
- Object-oriented (closely related to imperative)
- Problem-oriented/application-specific