CS 331, Principles of Programming Languages

Introduction

Objectives

- To introduce several different paradigms of programming
  - *But isn't one language pretty much like another? No!*
- To gain experience with these paradigms by using example programming languages
- To understand concepts of syntax, translation, abstraction, and implementation

Paradigms of Programming?

- There are several ways to think about computation:
  - a set of instructions to be executed
  - a set of expressions to be evaluated
  - a set of rules to be applied
  - a set of objects to be arranged
  - a set of messages to be sent and received

Some Programming Paradigms

- Procedural
  - examples: C, Pascal, Basic, Fortran
- Functional
  - examples: Lisp, ML
- Object-oriented
  - examples: C++, Java, Smalltalk
- Rule-based (or Logic)
  - example: Prolog

Why so many?

- Most important: the choice of paradigm (and therefore language) depends on how humans best think about the problem
- Other considerations:
  - efficiency
  - compatibility with existing code
  - availability of translators

Models of Computation

- RAM machine
  - procedural
- directed acyclic graphs
  - Smalltalk model of O-O
- partial recursive functions
  - Lisp and ML
- Markov algorithms
  - Prolog is loosely based on these

Lots of Languages

- There are many programming languages out there
- Lots of other PL-like objects
  - document languages, e.g. LaTeX, Postscript
  - command languages, e.g. bash, MATLAB
  - markup languages, e.g. HTML and XML
  - specification languages, e.g. UML

Issues for all Languages

- Can it be understood by people and processed by machines?
  - although translation may be required
- Sufficient expressive power?
  - can we say what needs to be said, at an appropriate level of abstraction?

Translation

- Compilation
  - Translate into instructions suitable for some other (lower level) machine
  - During execution, that machine maintains program state information
- Interpretation
  - May involve some translation
  - Interpreter maintains program state

Trade-offs

- Compilation
  - lower level machine may be faster, so programs run faster
  - compilation can be expensive
  - examples: C (and Java?)
- Interpretation
  - more ability to perform diagnostics (or changes) at run-time
  - examples: Basic, UNIX shells, Lisp