

# CS 603: Programming Language Organization

---

Lecture 3

Spring 2005

Department of Computer Science

University of Alabama

Joel Jones

# Outline

---

- Questions
- Personal Information
- Using Impcore
- Impcore Examples
- Operational Semantics
- Reading for next time

# Personal Information

Name:

Email:

Programming Languages/Environments:

Group Work Experience:

What you expect of the class:

Laptop?

# My Personal Information

**Name:** Joel Jones

**Email:** jones@cs.ua.edu

**Programming Languages/Environments:** C, C++, Java,  
Smalltalk on Macintosh, Unix, VMS

**Group Work Experience:** Industry experience for DP,  
scheduling, OS, and compilers

**What you expect of the class:** Preparation, Participation,  
Perception

# LISP-like Syntax

```
toplevel ::= exp
          | (use file-name)
          | (val variable-name exp)
          | (define function-name
             (formals) exp)
exp       ::= value
          | variable-name
          | (set variable-name exp)
          | (if exp exp exp)
          | (while exp exp)
          | (begin exp exp)
          | (function {exp})
formals  ::= {variable-name}
value    ::= integer
function ::= function-name
          | primitive
primitive ::= + | - | * | / | = | < | > |
           | print
```

```
integer ::= sequence of digits, possibly
          prefixed with a plus or minus
          sign
*-name  ::= sequence of characters not
          an integer and not
          containing (, ), ;, or
          whitespace
```

# Environments

- Set of mappings from names to values (or meanings)
- Operations on environments
  - Lookup—given name, return value:  $\rho(x)$ 
    - $\rho$  - environment
    - $x$  - name
  - Extend—given name and value, add to mappings:  $\rho\{x \mapsto v\}$ 
    - $v$  - value

# Specifying Meaning

- Operational Semantics
  - Mapping from AST (or other abstraction representation) to meaning, in terms of primitives
  - Mechanics involve state and transitions from state to state, involving inference, environments, sets, etc. which define a virtual machine
- Interpreters
  - Given operational semantics, interpreters can be derived, almost (but not quite!) mechanically

# Judgements and Rules of Inference

- Transition rules of virtual machine are written in the form of *judgments*
- A judgment is a relation, not a function, which implies that non-deterministic evaluations are possible.
- A judgment consists of premises and a conclusion
- A judgment holds only if all of the premises are true



# Operational Semantics of Impcore (State)

- Four parts to state
  - *Toplevel*  $t$  or *expression*  $e$  being evaluated (matches on AST tags)
  - Value environment holding values of global variables,  $\xi$
  - Function definition environment,  $\phi$
  - Value environment holding formal parameters,  $\rho$

# Operational Semantics of Impcore (Judgements)

- State when evaluating judgments for *oplevel*:  $\langle t, \xi, \phi \rangle$
- State when evaluating judgments for expression:  $\langle e, \xi, \phi, \rho \rangle$
- State between evaluations of *oplevel*:

Pair Up:

- How many elements to state tuple?
  - What contents? Why?
- 
- $\langle \xi, \phi \rangle$

# See Supplement

