ABSTRACT SYNTAX AND ITS REPRESENTATION

Simple Grammar

- Consider the simple grammar we have seen before:

```
<expression>::= <identifier>
             ::= (lambda (<identifier>) <expression>)
             ::= (<expression> <expression>)
```

- Develop a data type for it.

Data Type for Simple Grammar

```
(define-datatype expression expression?
  [var-exp (id symbol?)]
  [lambda-exp (id symbol?)
              (body expression?)]
  [app-exp (operator expression?)
           (operand expression?)])
```

Relationship between Grammar and Data Type

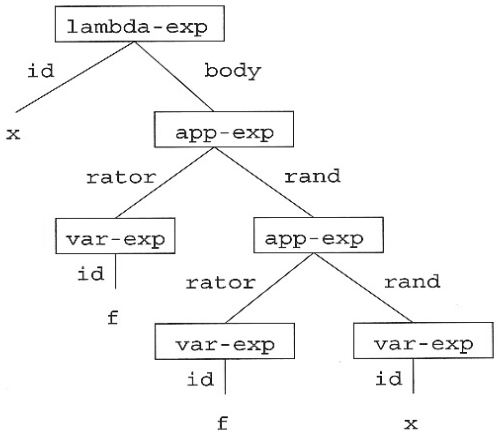
```
<expression>::= <identifier>
                var-exp (id)

            ::= (lambda (<identifier>) <expression>)
                lambda-exp (id body)

            ::= (<expression> <expression>)
                app-exp (operator operand)
```

Abstract Syntax Tree

- Consider: **`(lambda (x) (f (f x)))`**



Show and Tell

- A parser and interpreter for the simple language we have studied.

Extending the Language

- Class demo: Extend the
  - Grammar
  - Data type and
  - Parser/Unparser
- By:
  1. Numbers
  2. If-exp
  3. Lambda/App expressions with multiple parameters