ASTs for Expressions
AST design

• One way to build ASTs in OO languages like Java:
  • `ASTNode` abstract class
  • Sub-class `ASTNode` for specific constructs (statements, expressions, etc.)
• Expression `ASTNodes`:
  • Binary expressions (`x + y`)
  • Unary expressions (`z++`, `-x`, `*p`)
  • Array expressions (`a[i]`)
• Node stores type of expression result
building ASTs for expressions

• Each expression non-terminal generates an ExpressionNode

• Expression non-terminals reference other expression non-terminals:

```plaintext
expr : term
    | expr ?? term ;

term : primary
    | term ?? primary ;
```

• Key: assume each expression non-terminal returns a correctly-built ExpressionNode

• Only challenge then: “hook up” the existing expression nodes to create new one, propagate types

```plaintext
a + b * c
```
base cases

• Identifier
  • Check if variable is in symbol table
  • Create Variable AST node with pointer to symbol table
• Literals
  • Create AST node for constants
  • Often store constant value as string (why?)
other node types

• Assignment nodes
  • Store left-hand-side expression (may not be a variable!)
  • Store right-hand-side expression
• Statement lists
  • Build up list of statements recursively
next: generating code