Project 4: Constructing the Abstract Syntax Tree

Modify Parser.java

Basic Idea: Each parsing method will return an Abstract Syntax Tree (AST) for whatever was parsed.

PrintAst.java

`void printAst (Ast.Node t)`

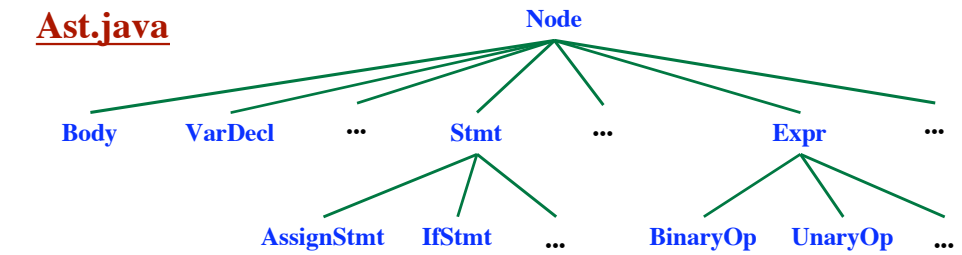
Print the AST in full detail

Ast.Java

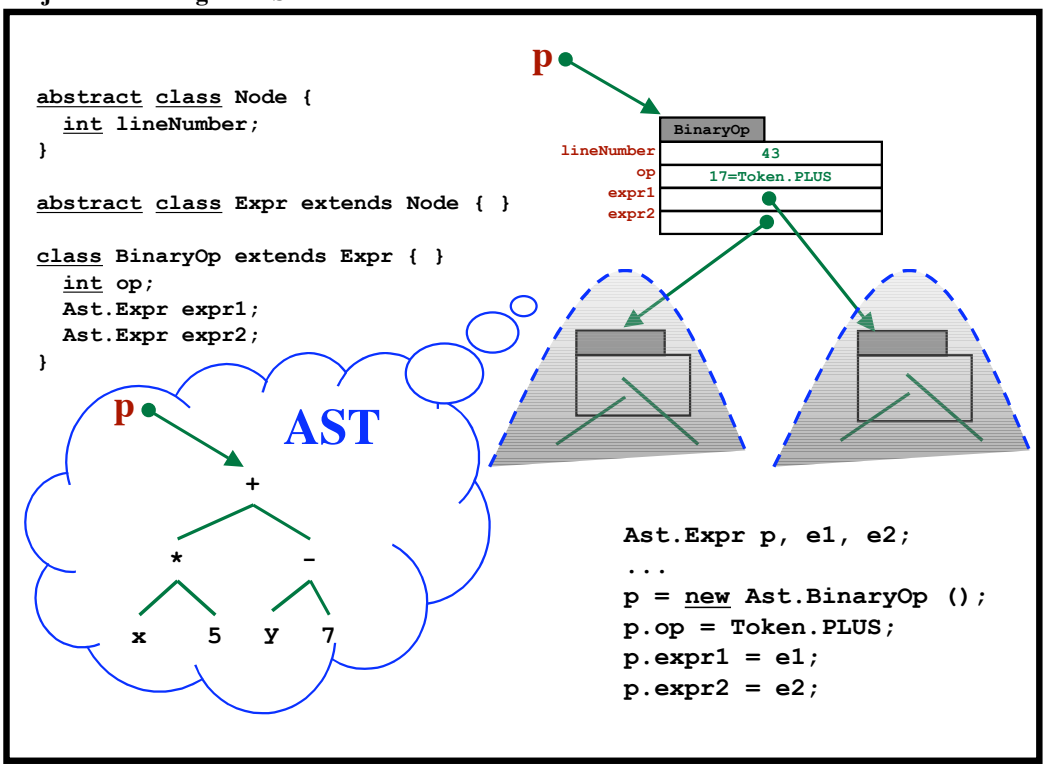
Contains classes related to the AST

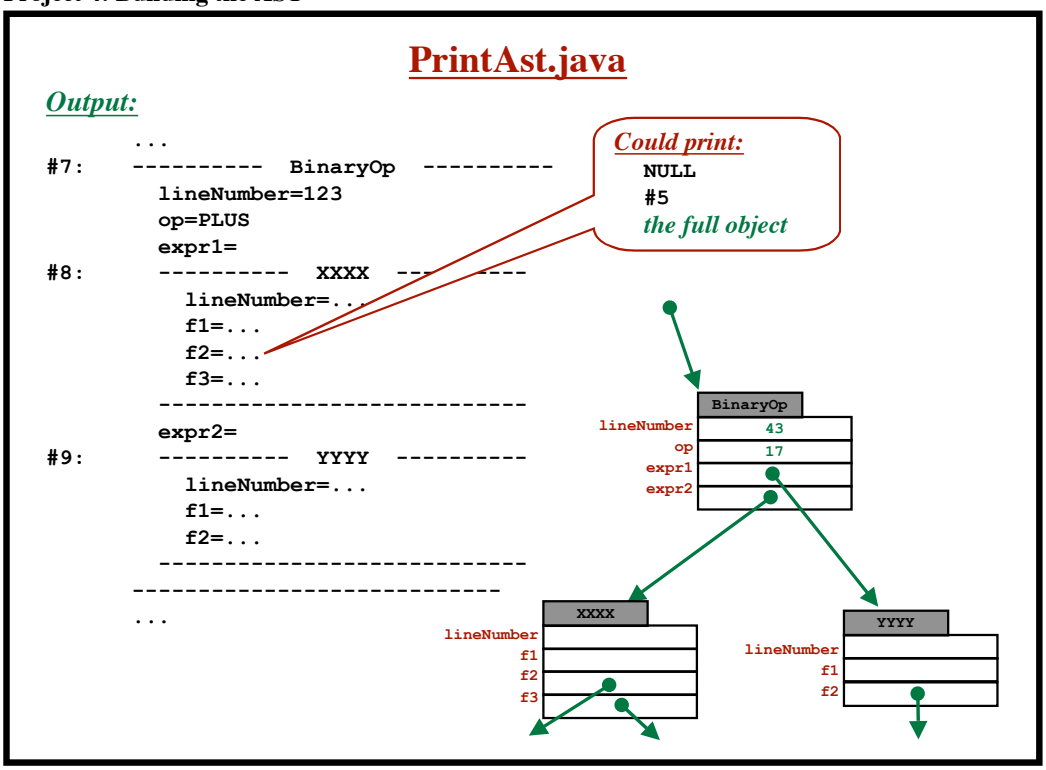
- Lots of classes
- Small (0-5 fields)
- No methods (of interest)
- "Data Structure" classes

Ast.java




```
class Ast {
  abstract class Node { ... }
  class Body extends Node { ... }
  class VarDecl extends Node { ... }
  ...
  abstract class Stmt extends Node { ... }
  class AssignStmt extends Stmt { ... }
  class IfStmt extends Stmt { ... }
  ...
  abstract class Expr extends Node { ... }
  class BinaryOp extends Expr { ... }
  class UnaryOp extends Expr { ... }
  ...
}
```





Displaying / Printout

PrintAst

Very detailed; prints all
Designed for verifying correct programs
Expressions:

1+2*3+4*5+6*7+8*9

84 lines of output!

Hard to understand incorrect output.

Special case: "Return Statement"

Attempts to print out the expression in infix
Example

```
#17:     ----------  ReturnStmt  ----------
           lineNumber=65
           summary=((((1+(2*3))+(4*5))+(6*7))+(8*9))
           expr=
             ...
         -------------------------------
```


Suggested Plan of Attack

Step 1: Read the assignment (16 pages).

```
compilers/p4
```

Step 2: Modify all method headers:

```
parseExpr3 → Ast.Expr
parseIfStmt → Ast.Stmt
parseLValue → Ast.LValue
```

Insert dummy return statements:

```
return null;
```

Step 3: Remove all print statements from Project 3.

Step 4: Get a clean compile.

Step 5: Get a "Body" returned and printed.

```
program is begin end;
```

Step 6:

```
parseStmts
parseReturnStmt
parseExpr0,1,2,3,4,5
```

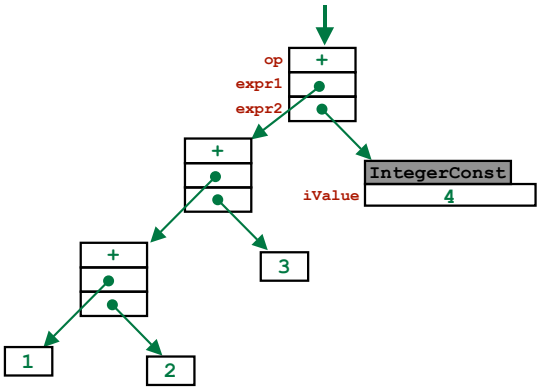
Skip ID, skip parseIDMods; Should now be able to do:

```
program is
  begin
    return 1+2+3;
  end;
```

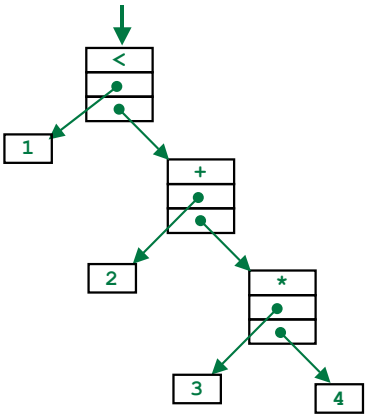
Expressions

AST must reflect the correct precedence and associativity.
Correct parsing → correct AST

1+2+3+4
= ((1+2)+3)+4

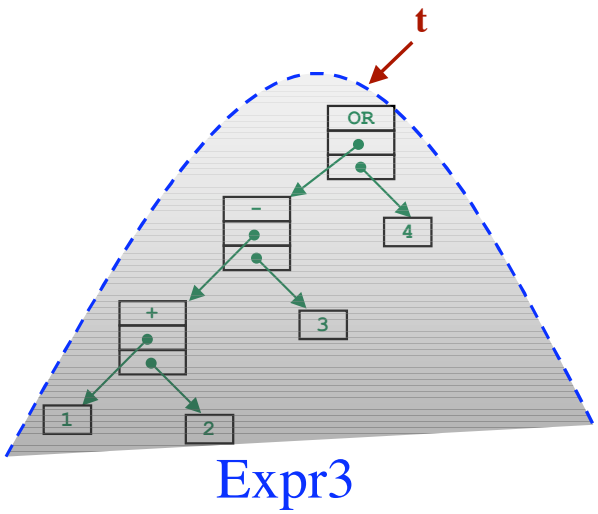


1<2+3*4
= 1<(2+(3*4))



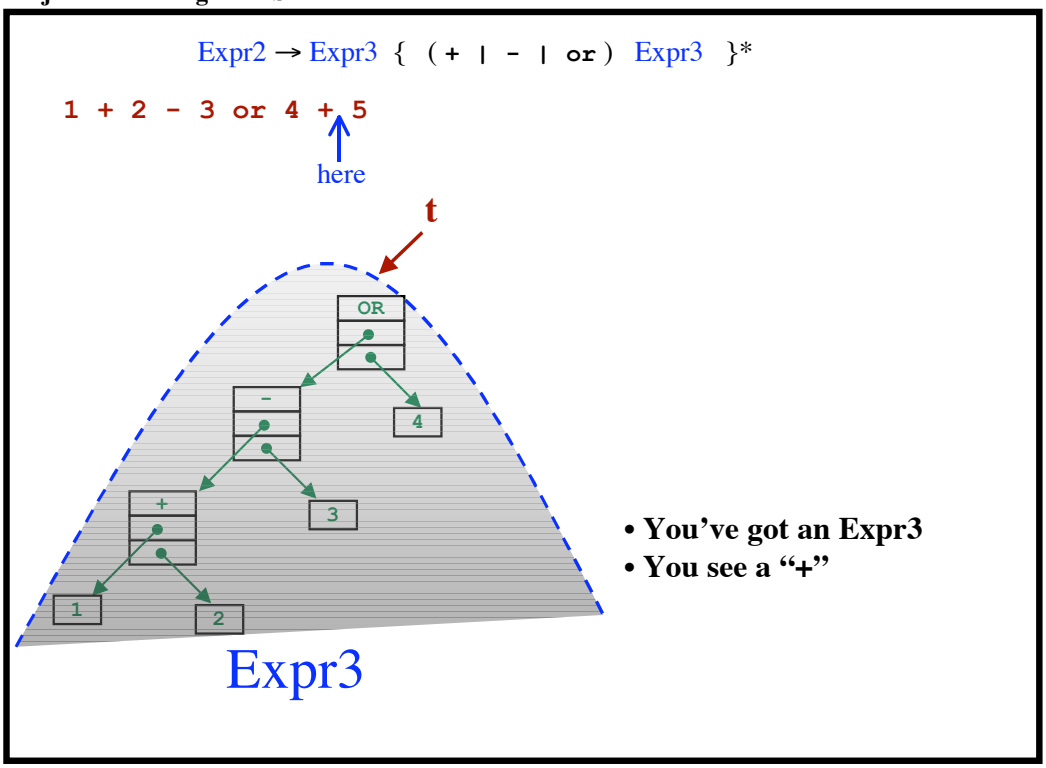
Expr2 → Expr3 { (+ | - | or) Expr3 }*

1 + 2 - 3 or 4 + 5
here

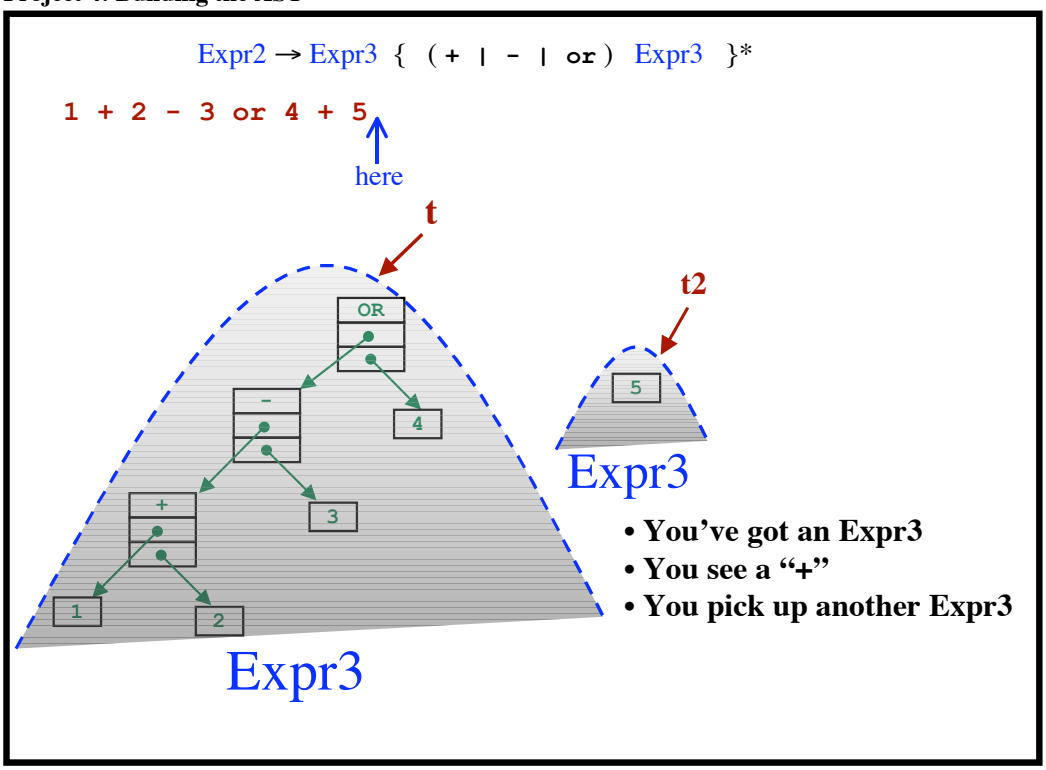


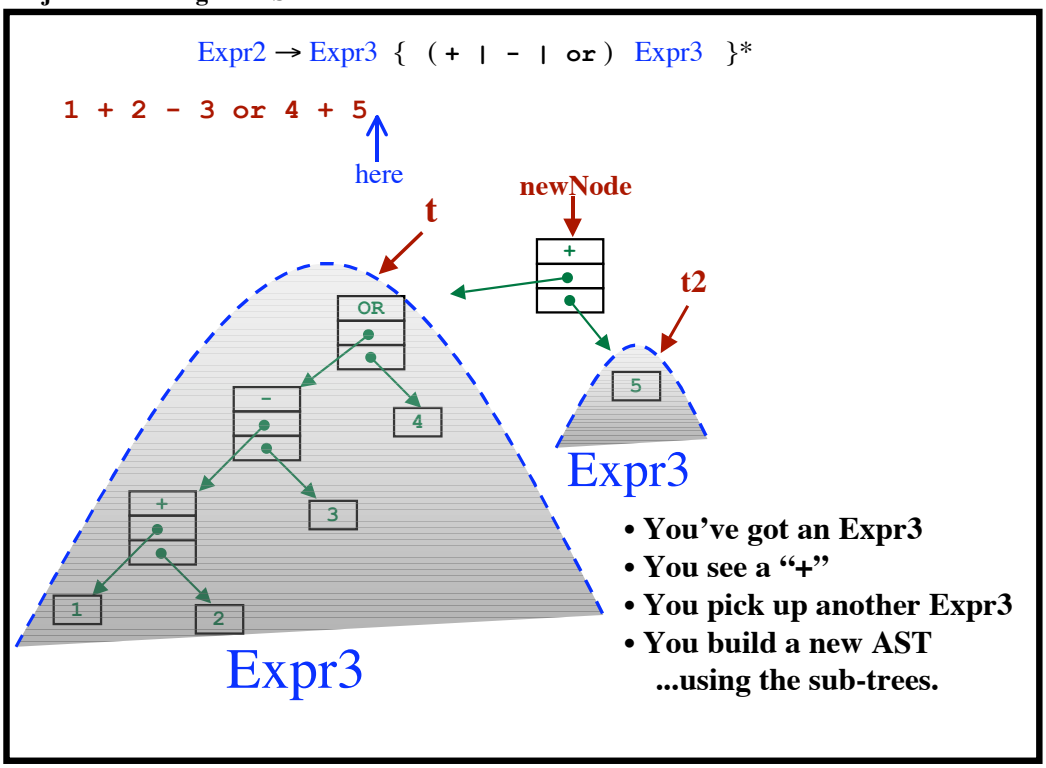
- **You've got an Expr3**

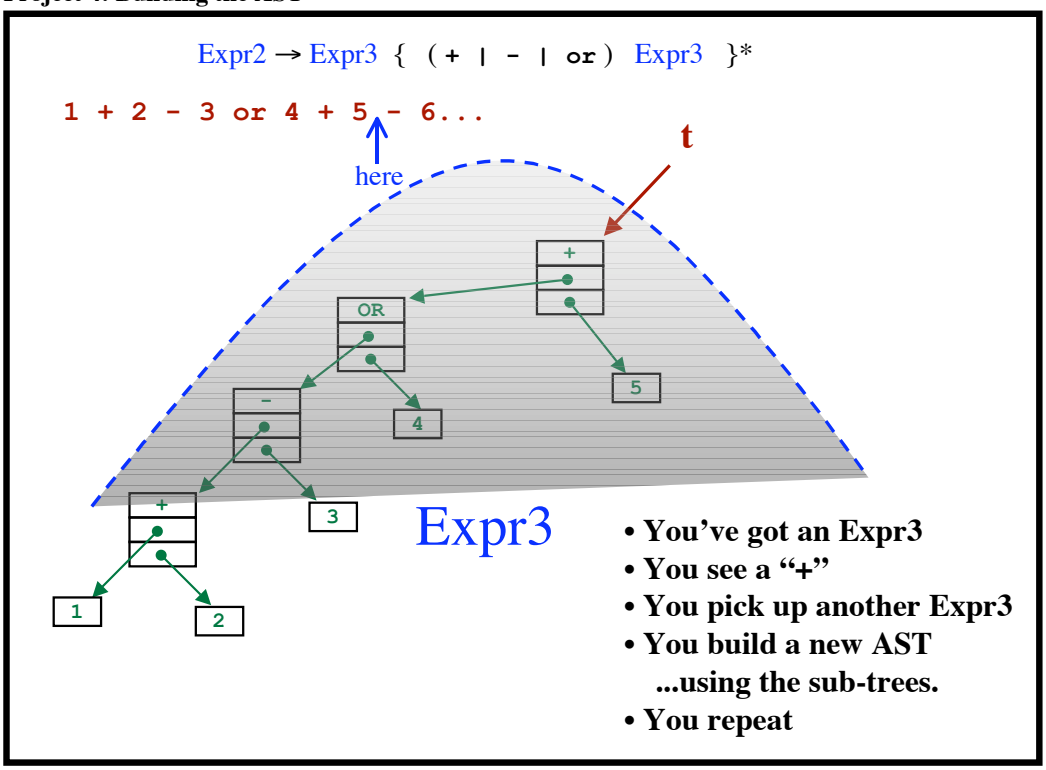
Project 4: Building the AST



Project 4: Building the AST







Outline of ParseExpr2

```
void parseExpr2 () {
  parseExpr3()
  while (nextToken == '+' or
         nextToken == '-' or
         nextToken == 'OR') do

    scan()
    parseExpr3()




  endWhile
  return
}
```

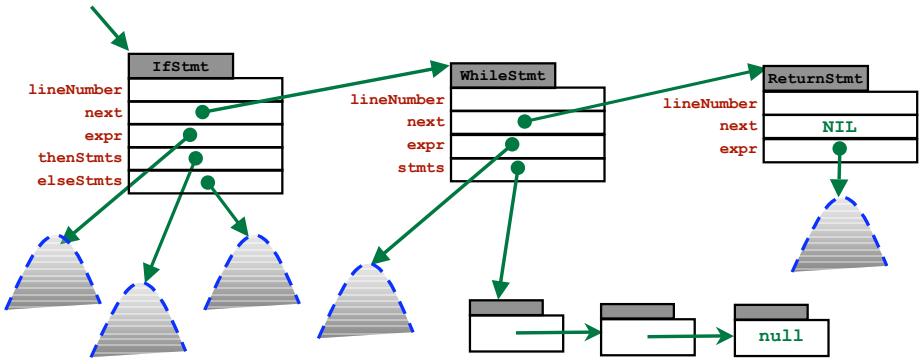

Outline of ParseExpr2

```
Ast.Expr parseExpr2 () {
  t = parseExpr3()
  while (nextToken == '+' or
         nextToken == '-' or
         nextToken == 'OR') do
    op = nextToken
    scan()
    t2 = parseExpr3()
    newNode = new Ast.BinaryOp()
    newNode.op = op
    newNode.expr1 = t
    newNode.expr2 = t2
    t = newNode
  endWhile
  return t
}
```

Statement Lists

```
abstract class Stmt extends Node {
  Ast.Stmt next;
}
class IfStmt extends Stmt {
  Ast.Expr expr;
  Ast.Stmt thenStmts;
  Ast.Stmt elseStmts;
}
...
```

`parseStmts` *returns a pointer to a linked list of* `Stmt` *nodes (possibly null)*



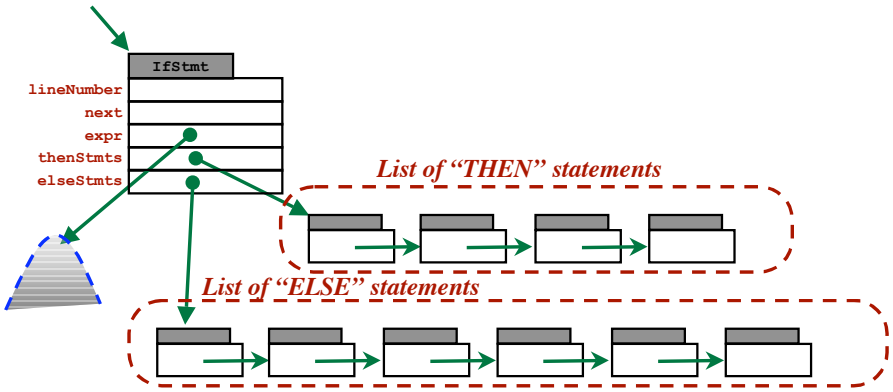
Lists

- List of Stmt Nodes
  `x:=4; y:=5; return x*y;`
- List of Argument Nodes
  `foo (4, x+y, 6, b-5)`
- List of VarDecl Nodes
  ```
  var x,y,z: Integer := 43;
      a,b,c: Real := 4.5;
  ```
- List of TypeDecl Nodes
  ```
  type T1 is ...;
       MyArray is ...;
  ```
- List of ProcDecl Nodes
- List of Formal Nodes
  `procedure foo (x,y:Integer, a,b,c:Real) is ...`
- List of FieldDecl Nodes
  `record f1:Integer, f2:Real, f3:MyArray, ... end`
- List of FieldInit Nodes
  `MyRec { f1=4; f2=3.14, f3=arr, ... }`
- List of ArrayValue Nodes
  `MyArray {{ 5, 7, 9, 11, 13 }}`

Always linked on a field called "`next`*".*

IF-THEN-ELSE Statements

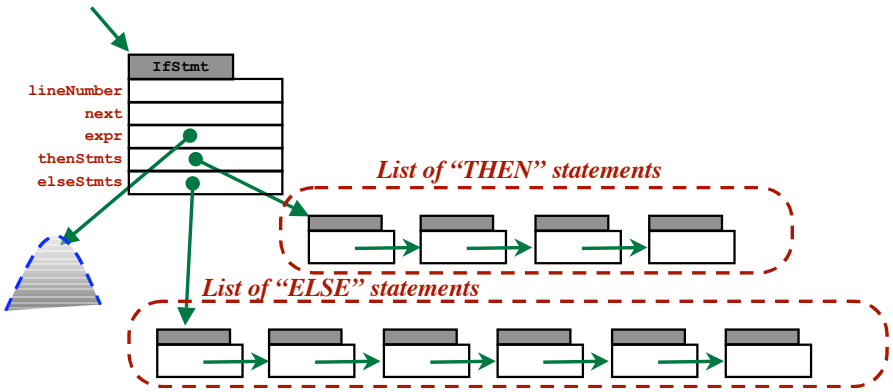
```
if expr then
  thenStmts
else
  elseStmts
end;
```



IF-THEN-ELSE Statements

```
if expr then
  thenStmts
else
  elseStmts
end;
```

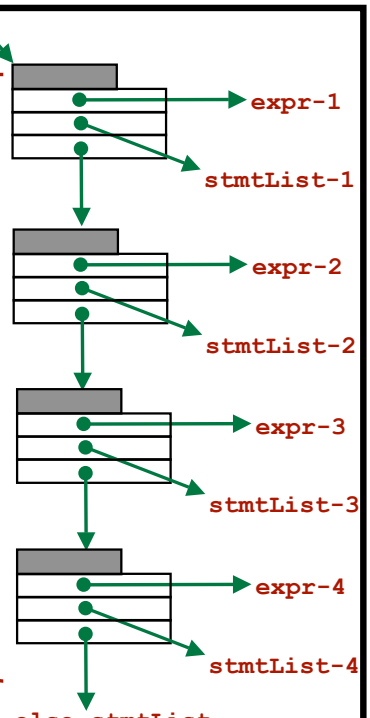
No provision for the "else-if" clause!!!



Project 4: Building the AST

```
if expr-1 then
   stmtList-1
elseif expr-2 then
   stmtList-2
elseif expr-3 then
   stmtList-3
elseif expr-4 then
   stmtList-4
else
   else-stmtList
end;
```

We want some kind of a a linked list, with one element per `elseif` *clause...*

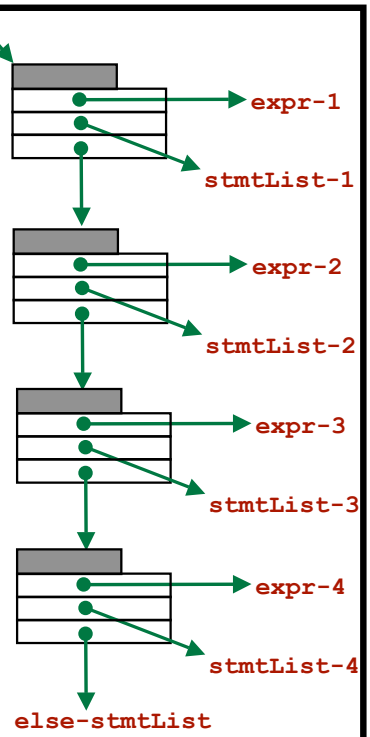


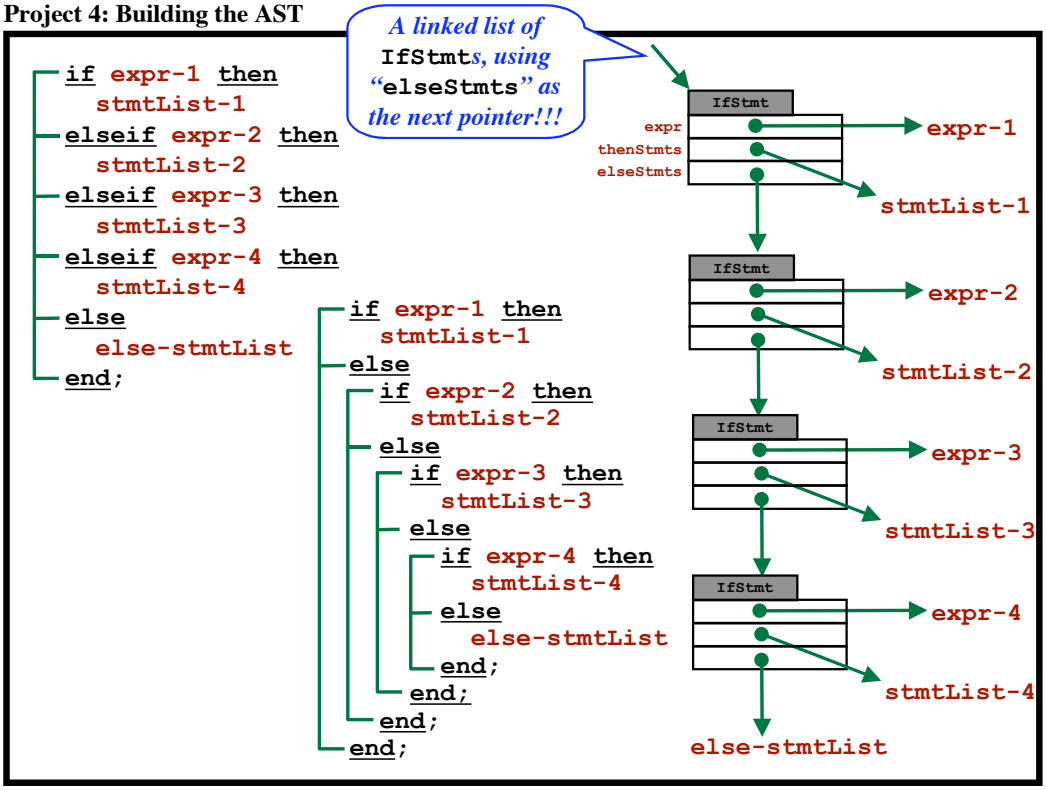
Project 4: Building the AST

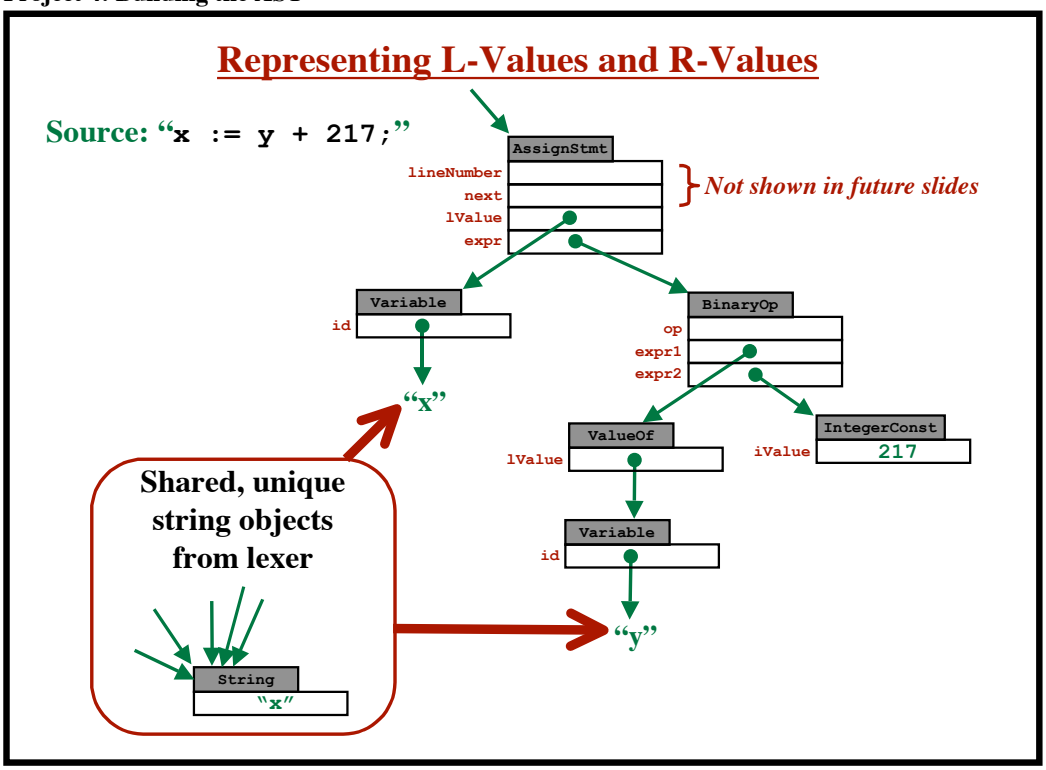
```
if expr-1 then
   stmtList-1
elseif expr-2 then
   stmtList-2
elseif expr-3 then
   stmtList-3
elseif expr-4 then
   stmtList-4
else
   else-stmtList
end;
```

```
if expr-1 then
   stmtList-1
else
   if expr-2 then
      stmtList-2
   else
      if expr-3 then
         stmtList-3
      else
         if expr-4 then
            stmtList-4
         else
            else-stmtList
         end;
      end;
   end;
end;
```

Notice that these are semantically equivalent







Representing L-Values and R-Values

Class Hierarchy:

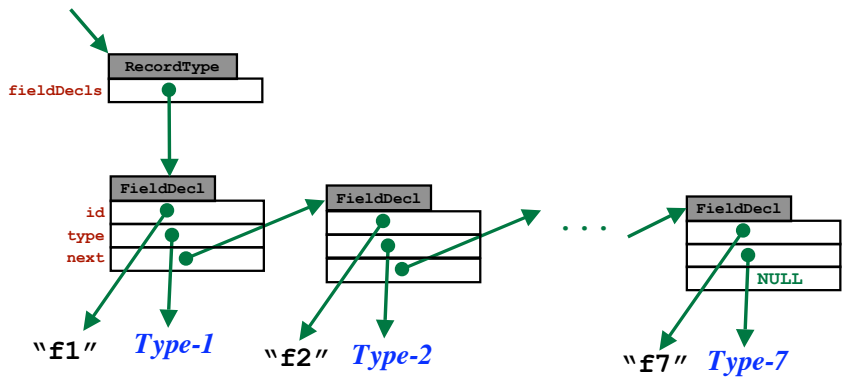
```
LValue
    Variable
    ArrayDeref
    RecordDeref
```

Whenever an L-Value is used as an R-Value...
There must be a `ValueOf` node inserted.

An LValue is not a kind of expression.
...but a "`ValueOf`" is!

Record Types

```
type MyRec is record
                f1: ...Type-1... ;
                f2: ...Type-2... ;
                ...
                f7: ...Type-7... ;
              end;
```



The Body Node

Example:

All of this grouping info will be Lost in the representation:

```
type T1 is ...;
     T2 is ...;
var x,y,z: ...;
    a,b,c: ...;
procedure
  foo() is ... end;
  bar() is ... end;
type T3 is ...;
     T4 is ...;
     T5 is ...;
var i,j: ...;
procedure
  goo() is ... end;
begin
  ...
end;
```


The Body Node

Example:

All of this grouping info will be Lost in the representation:

```
type T1 is ...;
     T2 is ...;
var x,y,z: ...;
    a,b,c: ...;
procedure
  foo() is ... end;
  bar() is ... end;
type T3 is ...;
     T4 is ...;
     T5 is ...;
var i,j: ...;
procedure
  goo() is ... end;
begin
  ...
end;
```

```
type T1 is ...;
     T2 is ...;
     T3 is ...;
     T4 is ...;
     T5 is ...;
procedure
  foo() is ... end;
  bar() is ... end;
  goo() is ... end;
var x,y,z: ...;
    a,b,c: ...;
    i,j: ...;
begin
  ...
end;
```

The Body Node

Example:

```
type T1 is ...;
     T2 is ...;
var x,y,z: ...;
    a,b,c: ...;
procedure
  foo() is ... end;
  bar() is ... end;
type T3 is ...;
     T4 is ...;
     T5 is ...;
var i,j: ...;
procedure
  goo() is ... end;
begin
  ...
end;
```

All of this grouping info will be Lost in the representation:

```
type T1 is ...;
     T2 is ...;
     T3 is ...;
     T4 is ...;
     T5 is ...;
procedure
  foo() is ... end;
  bar() is ... end;
  goo() is ... end;
var x,y,z: ...;
    a,b,c: ...;
    i,j: ...;
begin
  ...
end;
```

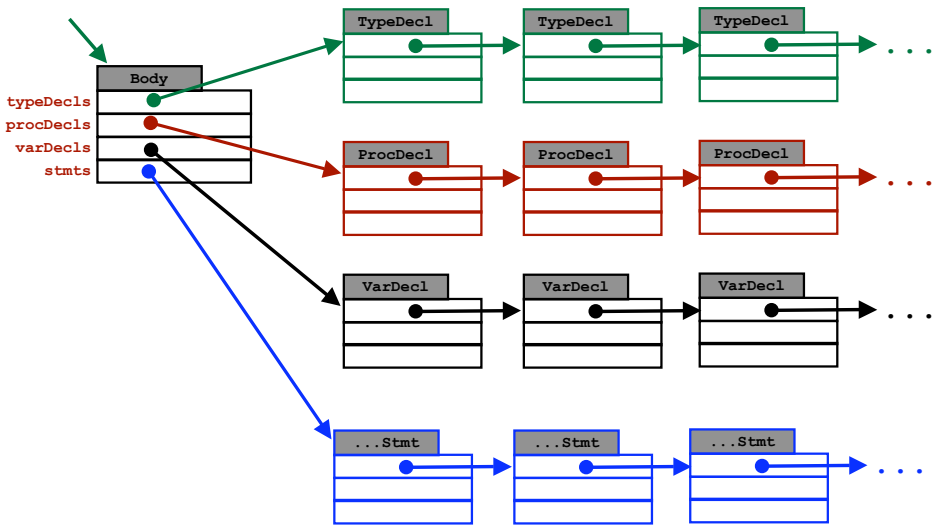
Type Decls

Proc Decls

Var Decls

Stmt List

The Body Node



```
Ast.Body parseBody () {
  b = new Ast.Body ();
  // All lists initialized to null
  parseDecls(b);
  b.smts = parseStmts();
  return b;
}
```

An "inherited" attribute

```
void parseDecls (Ast.Body b) {
  while (true) {
    if nextToken=="TYPE" then
      list = parseTypeDecl();
      ...Append list to end of b.typeDecls...
    elseif nextToken=="PROC" then
      list = parseProcDecl();
      ...Append list to end of b.procDecls...
    elseif nextToken=="VAR" then
      list = parseVarDecl();
      ...Append list to end of b.varDecls...
    elseif nextToken=="BEGIN" then
      return;
    else
      syntaxError();
    endIf
  }
}
```


TypeDecls

```
var listPtr: MyRec := nil;
type MyRec is record
               val: Integer;
               next: MyRec;
             end;
```