Automatic Parser Generation in MiniJava

- We use the CUP tool to automatically create a parser from a specification file, Parser/minijava.cup
- The build.xml file automatically rebuilds the parser whenever its specification file changes
- A CUP file has several sections:
  - introductory declarations included with the generated parser
  - declarations of the terminals and nonterminals with their types
  - The AST node or other value returned when finished parsing that nonterminal or terminal
  - precedence declarations
  - productions + actions

Terminal Declarations

- Terminal declarations are as before:

```
/* reserved words: */
terminal CLASS, PUBLIC, STATIC,
EXTENDS;
...
/* tokens with values: */
terminal String IDENTIFIER;
terminal Integer INT_LITERAL;
```

Nonterminal Declarations

- Nonterminals are similar. The flavor is:

```
nonterminal Program Program;
nonterminal MainClassDecl MainClassDecl;
...
nonterminal List<Stmt> Stmts;
nonterminal Stmt Stmt;
nonterminal List<Expr> Exprs;
nonterminal Expr Expr;
nonterminal String Identifier;
```

- See the actual AST classes for the real names, etc.
  (Taken from the Appel MiniJava project web site)

Precedence Declarations

- Can specify precedence and associativity of operators
  - equal precedence in a single declaration
  - lowest precedence textually first
  - specify left, right, or nonassoc with each declaration
- Examples:

```
precedence left AND_AND;
precedence nonassoc EQUALS_EQUALS, EXCLAIM_EQUALS;
precedence left LESSTHAN, LESSEQUAL, GREATEREQUAL,
GREATERTHAN;
precedence left PLUS, MINUS;
precedence left STAR, SLASH;
precedence left EXCLAIM;
precedence left PERIOD;
```

Productions

- All of the form:

```
LHS ::= RHS1 {: Java code 1 :}
| RHS2 {: Java code 2 :}
| ...
| RHSn {: Java code n :};
```

- Can label symbols in RHS with :var suffix to refer to its result value in Java code
  - varleft is set to line in input where var symbol was
- Java code is for semantic actions that build the AST (next part of the project)

Productions (cont.)

- Example (again, may not match your grammar symbols – adjust as needed)

```
Expr ::= Expr:arg1 PLUS Expr:arg2
| INT_LITERAL:value
| ...
;
```

Error Handling

- How to handle syntax error?
- Option 1: quit compilation
  - + easy
  - - inconvenient for programmer
- Option 2: error recovery
  - + try to catch as many errors as possible on one compile
  - - difficult to avoid streams of spurious errors
- Option 3: error correction
  - + fix syntax errors as part of compilation
  - - hard!!

Panic Mode Error Recovery

- When finding a syntax error, skip tokens until reaching a “landmark”
  - landmarks in MiniJava: ;,), }
  - once a landmark is found, hope to have gotten back on track

Panic Mode Error Recovery

- In bottom-up parser, can add special error nonterminals, followed by landmarks
  - o if syntax error, then will skip tokens till seeing landmark, then reduce and continue normally
- E.g.

  Stmt ::= ... | error ; | { error }

  Expr ::= ... | (error)