CS 536 Announcements for Wednesday, February 21, 2024

Programming Assignment 2
• due Tuesday, February 20 – accepted until 11:59 pm Thursday
• see late policy on course website

Midterm 1
• Thursday, February 29, 7:30 – 9 pm
• S429 Chemistry
• bring your student ID

Last Time
• implementing ASTs

Today
• Java CUP

Next Week
• review for Midterm 1
• parsing

Parser generators

Tools that take an SDT spec and build an AST

• YACC

• Java CUP

Conceptually similar to JLex:
• Input: language rules + actions
• Output: Java code

parser \rightarrow\quad \text{Java CUP} \quad \rightarrow\quad \text{parser source}

\text{specification} \quad \rightarrow\quad \text{symbols}
Java CUP

parser.java

- constructor takes argument of type Yylex
- parse method
  - if input correct, returns Symbol whose value field contains translation of root nonterm
  - if input incorrect, quits on first syntax error
- uses output of JLex
  - depends on scanner and TokenVal classes
  - sym.java defines the communication language
- uses definitions of AST classes (in ast.java)

Parts of Java CUP specification

Grammar rules with actions:

```plaintext
eexpr ::= INTLITERAL
    | ID
    | eexpr PLUS eexpr
    | eexpr TIMES eexpr
    | LPAREN eexpr RPAREN
```

Terminal and nonterminal declarations:

```plaintext
terminal INTLITERAL;
terminal ID;
terminal PLUS;
terminal TIMES;
terminal LPAREN;
terminal RPAREN;
non terminal eexpr;
```

Precedence and associativity declarations:

```plaintext
precedence left PLUS;
precedence left TIMES;
```
Java CUP Example

Assume:

- **Java class** `ExpNode` with subclasses `IntLitNode`, `IdNode`, `PlusNode`, `TimesNode`
- `PlusNode` and `TimesNode` each have two children
- `IdNode` has a **String** field (for the identifier)
- `IntLitNode` has an **int** field (for the integer value)
- **INTLITERAL** token is represented by `IntLitTokenVal` class and has field `intVal`
- **ID** token is represented by `IdTokenVal` class and has field `idVal`

**Step 1: add types to terminals and nonterminals**

```
/*
 * Terminal declarations
 */
terminal INTLITERAL;
terminal ID;
terminal PLUS;
terminal TIMES;
terminal LPAREN;
terminal RPAREN;

/*
 * Nonterminal declarations
 */
non terminal expr;
```

**Step 2: add precedences and associativities**

```
/*
 * Precedence and associativity declarations
 */
precedence left PLUS;
precedence left TIMES;
```
Java CUP Example (cont.)

Step 3: add actions to CFG rules

    /*
     * Grammar rules with actions
     */
    expr ::= INTLITERAL
         {:
         :
         }
       | ID
         {:
         :
         }
       | expr PLUS expr
         {:
         :
         }
       | expr TIMES expr
         {:
         :
         }
       | LPAREN expr RPAREN
         {:
         :
         }
     ;
Java CUP Example (cont.)

Input: 2 + 3
Translating lists

Example

\[ \text{idList} \rightarrow \text{idList COMMA ID | ID} \]

Left-recursion or right-recursion?

- for top-down parsers

- for Java CUP

Example

CFG: \[ \text{idList} \rightarrow \text{idList COMMA ID | ID} \]

Goal: the translation of an idList is a LinkedList of Strings

Example

Input: \( x, y, z \)

Output:
Example (cont.)

Java CUP specification for this syntax-directed translation
Terminal and nonterminal declarations:

Grammar rules and actions:

```
idList ::= idList COMMA ID
   {:
   :
   :
   |
   |
   |
   |
   |
   |

;}
```
Handling unary minus

/*
 * precedences and associativities of operators
 */
precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE;

/*
 * grammar rules
 */
exp ::= . . .
    | MINUS exp:e
      {: RESULT = new UnaryMinusNode(e);
      :}
    | exp:e1 PLUS exp:e2
      {: RESULT = new PlusNode(e1, e2);
      :}
    | exp:e1 MINUS exp:e2
      {: RESULT = new MinusNode(e1, e2);
      :}
    . . .
;