

Grammars (cont.)

literal strings are specified with quotes

can group things together with parens

asterisk means "zero or more"

operator precedence is encoded in parsing order

pipe is like "or" between options

```
expression ::= term (("+" | "-") term)*
term        ::= factor ("*" | "/" factor)*
factor      ::= "-" factor | "(" expression ")" | number
number      ::= digit (digit)*
digit       ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

The diagram illustrates the components of a grammar. It lists five rules: expression, term, factor, number, and digit. Annotations with arrows explain specific symbols: 'literal strings are specified with quotes' points to the quoted operators '+' and '-' in the expression rule; 'can group things together with parens' points to the parentheses in the expression rule; 'asterisk means "zero or more"' points to the '*' in the expression rule; 'operator precedence is encoded in parsing order' points to the '*' in the term rule; and 'pipe is like "or" between options' points to the pipe character in the digit rule.

Parser implementation

- Method of parsing that we will use is known as "recursive descent" parsing
- Determine which type of expression we must be at based on current and next tokens (can see next token by "peeking" at lexeme with next index)
- Our parser differs a little from the grammar given on the previous slide as some work is done by the lexer (digit grouping):

```
expression ::= term (("+" | "-") term)*  
term       ::= factor (("*" | "/" ) factor)*  
factor     ::= "-" factor | "(" expression ")" | number
```