Grammars (cont.)

literal strings are specified with quotes

can group things together with parens

asterisk means "zero or more"

```
expression ::= term (("+" | "−") term)∗

term       ::= factor (("∗" | "/") factor)∗

factor     ::= "−" factor | "(" expression ")" | number

number     ::= digit (digit)∗

digit      ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

operator precedence is encoded in parsing order

pipe is like "or" between options

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
```