

# Grammars

- Once we have the lexemes generated, we need to figure out how they fit together
- The ways lexemes fit together to create things like an "addition expression", or "number", etc. are typically specified by a **grammar**
- Grammars consist of a set of rules for creating expressions from lexemes
- They are typically written in Backus-Naur form ("BNF"), in which a type of expression is on one side and the way(s) it can be created are on the other

# 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 quotes around '+' and '-' in the expression rule; 'can group things together with parens' points to the parentheses around the expression rule's recursive part; '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.