Reading an s-expression

- A tree/list/s-expression, sexpr is
  - Empty, symbolized as ()
  - Not-empty, symbolized as ( number sexpr sexpr )

- We can write this as a grammar
  
  \[
  \text{sexpr} :: () \\
  \text{sexpr} :: ( \text{number} \text{sexpr} \text{sexpr} ) \\
  \text{number} :: \text{digit} \\
  \text{digit} :: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  \]

Parsing a grammar

- What character starts an s-expression?
- What character ends an s-expression?
- How do we tell if the s-expression is empty?

- Reading a number
  - Can read char-at-a-time and build the number
  - Can throw the char back and read as an int

```c
Node * readTree(istream& input) {
    char ch;
    input >> ch;        // use >> instead of get?
    if (ch == '(') {
        input >> ch;
        if (ch == ')') { // what to return?
            input >> ch;
            if (ch == ')') // what do we do here?
                // return value
        } else
            // do something
    }
    else
        // handle error
}
```

Counting Search Trees

- How many 0-node trees?
- How many 1-node trees?
- How many 2-node trees?
- How many 3-node trees?
- How many 4-node trees?

Generalizing # trees

- How many trees with N-nodes are there?
  - What knowledge/subproblems help us solve problem
  - How can we use recursive calls to help?

- Efficiency issues? What's the recurrence relation?
  - How can we memoize to reduce repeated work?
    - Avoid computing the same value multiple times
    - Store value and look it up rather than recomputing
  - What structures can we use to store our cached data?
    - Issues given size of universe for trees?
    - What happens in general?