More on Writing a Parser

COMP360

“Peace cannot be kept by force; it can only be achieved by understanding.”

Albert Einstein

Parsing Methods

- Top Down Parsing
  - Traces a leftmost derivation
  - Builds a parse tree in preorder
  - LL and Recursive Descent are top down
- Bottom Up Parsing
  - Builds a parse tree starting at the leaves
  - Produces the reverse of a rightmost derivation
  - LR and LR-K algorithms are bottom up

Recursive Descent

- Relatively easy algorithm to implement
- Functions or methods are written to represent each BNF production

- The goal of the recursive descent parser is to create a parse tree whose leaves (left to right) are the tokens, in order, created by the lexical scanner

Input function

- There is an input function that gets the next token from the lexical scan

```
Token getNextToken() {
        if (curTok >= tokenList.size()) {
            return special EOF Token;        // no more tokens
        }
        return tokenList.get(curTok++);
    }
```

- I also create a method, peek(), that returns the next token, but does not remove it from the input queue

Example Grammar

expr → term | term **+** expr

| term **-** expr

term → factor | factor * factor

| factor **/** factor

factor → **variable** | **(** expr **)**

exp function

```
boolean exp() {
   int saveLocation = curTok;
   if (!term()) {
      curTok = saveLocation;
      return false;
   }
   while (peekIs('+' || peekIs('-')) {
      getNextToken();
      if (!term()) {
         curTok = saveLocation;
         return false;
      }
   }
   return true;
}
```

expr → term | term + expr
| term - expr

term function

```
boolean term() {
   int saveLocation = curTok;
   if (!factor()) {
      curTok = saveLocation;
      return false;
   }
   while (peekIs('*' || peekIs('/')) {
      getNextToken();
      if (!factor()) {
         curTok = saveLocation;
         return false;
      }
   }
   return true;
}
```

term → factor | factor * factor
| factor / factor

Write the factor method

factor → **variable** | **(** expr **)**

- Assume there is an `isVariable()` method for objects of the Token class

Possible Solution

factor → variable | (expr)

```
boolean term() {
    int saveLocation = curTok;
    Token tok = getNextToken();
    if (tok.isVariable()) return true;
    if(tok.value == '(') {
      if (!expr()) {
          curTok = saveLocation;
          return false;
       }
    }
    tok = getNextToken();
    if(tok.value != ')') throw exception;
    return true;
}
```