# Semantics

COMP360

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

Albert Einstein

# Snowflake Parser

- A recursive descent parser for the Snowflake language is due by noon on Friday, February 17, 2017
- This parser should work with your Snowflake scanner
- The parser must determine if the input Snowflake program has proper syntax
- An appropriate error message (showing line and column) must be displayed if there is a syntax error

# 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();      // consume + or -
      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;
}
```

# Handling Errors

- Most compilers attempt to tell the user all syntax errors in their program
- If the program has incorrect syntax in the beginning, it may be difficult for the parser to know how to restart
- Some compilers will move forward to a token that starts the next obvious statement
- The compiler moves up the parse tree to try again

# Only the First Error

- Our compiler can stop when it detects an error
- You must report where you think the error is
- An easy way for recursive descent parsers to respond to an error is to throw an exception
- All methods should throw the exception except the main which should catch it and print and error message

# java.text.ParseException

- java.text.ParseException is a convenient exception to throw
- The constructor allows a message string and an int ParseException(String s, int curToken)
- The integer can be an index into the token list indicating where the error was detected
- Alternately, global variables can be set to the error message and location

# Errors in Different Phases

- When an error is detected in one phase of a compiler, it will usually stop and not perform any following phases
- If you get a simple error detected by the scanner, it will not run the parser
- Sometimes you will compile a program and get only one error.  After you fix that error, you get many more

# What made the Scanner program difficult?

A. I managed to write the program correctly
B. I don't understand the concept of scanning
C. I understand scanning, but could not make a program do it
D. Other factors (i.e. time) prevented me from doing it

# Semantic Analysis

- Semantic Analysis checks for non-syntax errors
- Some books define Semantic Analysis as checking all the errors you can't define in BNF

# What errors exist in this program?

```
main( ) {
        short mouse = badProg( 47 );
}
int badProg( String cat ) {
        double dog = cat *  1.23;
        return dog;
}
```

# Some Errors

- The main passes an integer to a String parameter
- The string is multiplied by a double
- The int method returns a double
- The main stores an int result in a short

# Which errors would be caught by the parser?

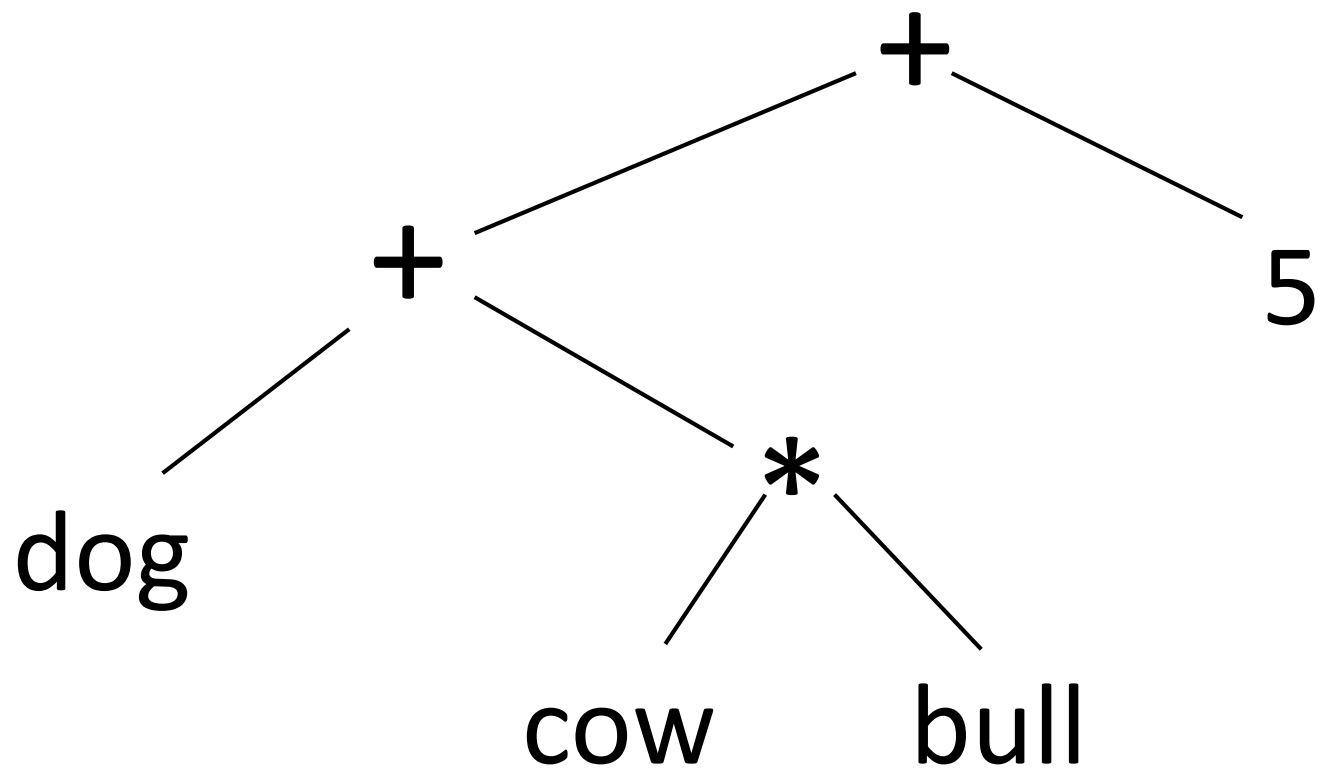
A. The main passes an integer to a String parameter
B. The string is multiplied by a double
C. The int method returns a double
D. The main stores an int result in a short
E None

# Creating a Parse Tree

- The compiler has to create a parse tree to direct the code generation
- Traversing the parse tree in a postfix order defines the execution order of the program
- Parse trees a built upon the parse derivation trees with additional information

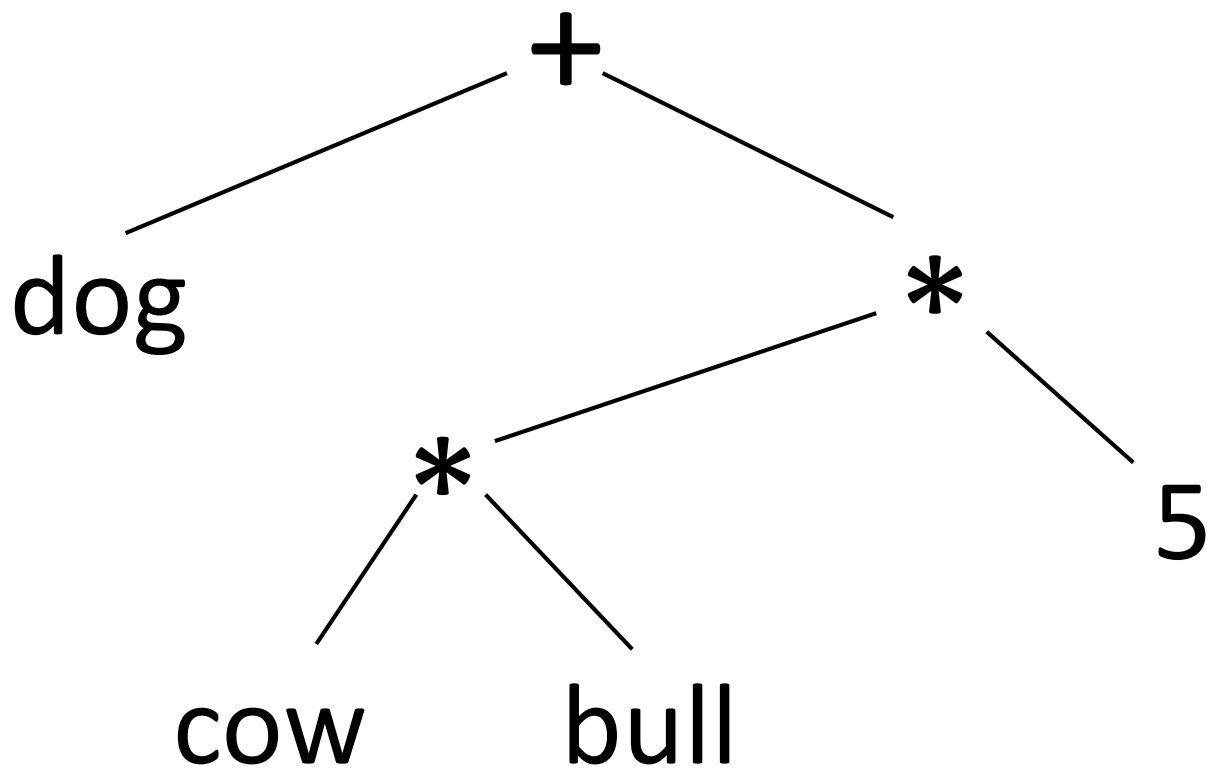
# Example Parse Tree

- Consider dog + cow * bull + 5



# Example Parse Tree

- Consider dog + cow * bull * 5



# Boolean Parsing

- The parser you are writing as an assignment only verifies that the input source has correct syntax
- The parser needs to build a parse tree
- Each method of the recursive descent parser needs to add to the parse tree
- Branches are added to the parse tree so the result will execute in the correct order when traversed

# Snowflake Parser

- A recursive descent parser for the Snowflake language is due by noon on Friday, February 17, 2017
- This parser should work with your Snowflake scanner
- The parser must determine if the input Snowflake program has proper syntax
- An appropriate error message (showing line and column) must be displayed if there is a syntax error