

# Design Pattern

The made design trade off that was made was to use the recursive descent algorithm instead of parser generators. The recursive descent algorithm is in some ways more simple and easy to understand. It also meshes well with the EBNF definition of the grammar. The EBNF grammar is defined recursively so it fits naturally and intuitively with recursive descent. The primary reason for our choice in the recursive descent algorithm was the ease of understandability of the method.

One drawback of recursive descent is that we have to write more lines of code by hand. This increases the technical complexity of the project as more lines of code have to be written. We thought that this is balanced by the greatly improved comprehensibility of recursive descent. Another drawback of recursive descent is that parser generators are more standard. Parser generators are taught more in universities and are often used in industry (although recursive descent is prevalent in industry). This is another major drawback and is something worth considering, but we still thought it was worth the tradeoffs.

In order to better understand the tradeoffs, we first should understand what parser generators are and how they work. Essentially, a parser generator is something that you can program to generate all the key elements of a compiler (lexer, parser) automatically. The lexer essentially works with defining regular expressions (regex) that match a given lexical element. For example, comments:

```
1 COMMENT
2     :   '//' ~('\n' | '\r')* '\r'? '\n' {skip();}
3     |   '/*' ( options {greedy=false;} : . )* '*/' {skip();}
4     ;
```

Make sense? That essentially selects everything up to the new line character for the '//' and everything until the next '/\*' as the comment. The next is the parser. This makes a parse tree and has some specific headers, but after that there are some components that resemble the EBNF that the grammar is defined by. For example with addition,

```
1 addExpr
2     : ID
3     | INT
4     | STRING
5     ;
```

In general, lots of the syntax is very obscure and it is not an intuitive implementation of the EBNF grammar. For this reason, we chose to use recursive descent algorithm instead.