# Compiler Construction (List 6)

Hans de Nivelle

13 November 2013

This is an exercise about top-down parsing using DFA-based grammars.

1. Download **top_down.tar.gz** from the course homepage, and make sure that it compiles. There is a tokenizer class and a token class. Tokens have attributes, but you don't need to worry about them because we will not be concerned with attributes in this exercise.

   In file **parser.cpp** is a main file that tests the tokenizer.

2. Consider the language $(\Sigma, R, S)$, defined by $\Sigma = \{\ '(', ')'\ \}$, $R = \{S \to SS,\quad S \to (S),\quad S \to \epsilon$. In the previous task list 5, we have constructed a DFA-based grammar for this language. Write a parser, based on this grammar.

   You can use some of the exising non-terminal symbols, for example **tkn_Start** instead of $S$. There is a class **parsestack**, that corresponds to the state, defined on slide 36. Methods Read/Return/Descend are already present. Complete the parser. Use **if**s or **switch**es.

3. Also implement a DFA-based grammar for Lisp. Test it on some expressions, e.g.

```
( a b c )
(( a ) ( b ) c )
( set a ( quote b ))
( define ( abs x ) ( if ( < x 0 ) ( - x ) x ))
```

**Note**: This is an exercise about top down parsing, not about hacking. I am aware that the languages in this exercise are so simple that they can be written without using any systematic method.