

Lexer

Tokenizing stream of characters using regular expressions

Lex

Lex is a program that generates lexical analyzer or scanner.

Structure of lex file (*.lex)

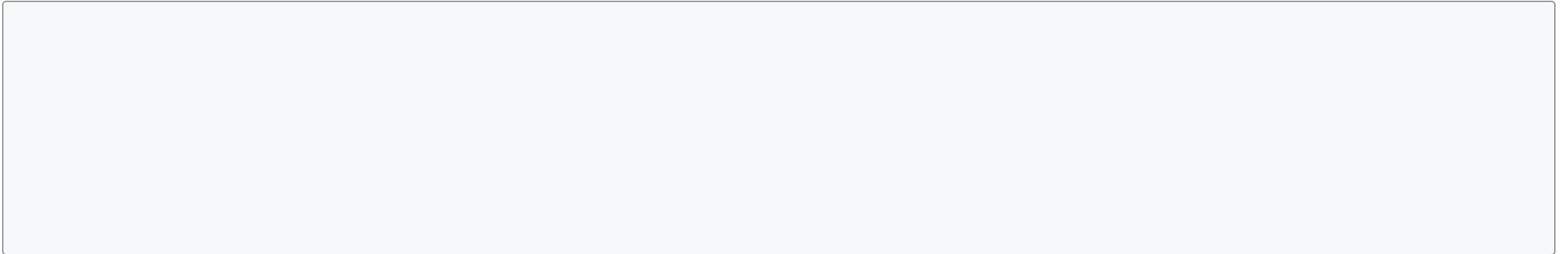
- Definition
 - define macros, import headers and etc.
- Rules
 - define regular expressions and the associated C / java / scala code block
- C code (lex or flex) / java code (jflex)
 - define utility functions which are accessible by rule code blocks

Concerning lex

- `YYINITIAL` : initial lexical state of the scanner
- `yylex()` : special function that returns the matched token
- `yybegin(<state>)` : "goto" <state>
- `%foo` : this is directive ([complete list of available directives](#))
 - `%line` : turns on line number counter so we could use a special function `yyline()`
 - `%class` : name of generated lexer class
 - `%type` : type of returned tokens in each code block
 - `%implements` : generated class implements
 - `%state` : defines new lexical state
- `[^]` : matches all characters not listed in the class. This is used to catch errors.

Lab assignment

- In this lab we will tokenize math expressions (parsing is next week!).
- Think about the tokens we may need.
- Source file:



Where to start

Checkout the first two lines of `Ca1.y`

- To view all possible tokens to return to we should look at `Ca1.y`
 - we can see all possible tokens, such as `ADD`, `SUB` and etc
 - we also see a `OPERAND` token which takes a `Double` as an argument
 - we can capture the "lexeme" by using `yytext()` method

Complete lex file

