Tokenizing
The tokenizer is relatively easy.
Aside: Characters vs. Tokens

• In the examples of CFGs, we dealt with languages over the alphabet of individual characters (e.g., Java’s `char` values)
  \[ \Sigma = \text{character} \]

• Now, we deal with languages over an alphabet of tokens, each of which is a unit that you want to consider as a single entity in the language
  – Choice of tokens is a design decision
Example: Expression CFG

\[
\begin{align*}
\text{expr} & \rightarrow \text{expr add-op term} \mid \text{term} \\
\text{term} & \rightarrow \text{term mult-op factor} \mid \text{factor} \\
\text{factor} & \rightarrow (\text{expr}) \mid \text{digit-seq} \\
\text{add-op} & \rightarrow + \mid - \\
\text{mult-op} & \rightarrow * \mid \text{DIV} \mid \text{REM} \\
\text{digit-seq} & \rightarrow \text{digit digit-seq} \mid \text{digit} \\
\text{digit} & \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{align*}
\]
Example: Expression

\[
\begin{align*}
\text{expr} & \rightarrow \text{expr add-op term} \\ 
\text{term} & \rightarrow \text{term mult-op factor} \\ 
\text{factor} & \rightarrow (\text{expr}) \\ 
\text{add-op} & \rightarrow + | - \\ 
\text{mult-op} & \rightarrow * | \text{DIV} | \text{REM} \\ 
\text{digit-seq} & \rightarrow \text{digit digit-seq} | \text{digit} \\ 
\text{digit} & \rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
\end{align*}
\]

Appropriate tokens for this CFG are “words” consisting of strings of consecutive terminal symbols (characters) that “belong together”, e.g., "+", "DIV", "5".
The job of the tokenizer is to transform a string of characters into a string of tokens.

Example:

- Input: "4 + (7 DIV 3) REM 5"
Tokenizer

• The job of the tokenizer is to transform a string of characters into a string of tokens

• Example:
  – Input: "4 + (7 DIV 3) REM 5"

characters used as terminal symbols of the language
Tokenizer

• The job of the tokenizer is to transform a string of characters into a string of tokens

• Example:
  – Input: "4 + (7 DIV 3) REM 5"

whitespace characters
Tokenizer

• The job of the tokenizer is to transform a string of characters into a string of tokens

• Example:

  – Input: "4 + (7 DIV 3) REM 5"

Mathematically, input is a string of character
Tokenizer

• The job of the tokenizer is to transform a string of characters into a string of tokens

• Example:
  – Input: "4 + (7 DIV 3) REM 5"
  – Output: <"4", "+", ",", "7", "DIV", "3", ",", "REM", "5">

Mathematically, output is a string of string of character
Another Example: BL

- In BL, tokens can be the “words” such as "IF", "next-is-empty", etc.
- A BL tokenizer is then easy: it can simply treat strings of consecutive whitespace characters as separators between tokens.
  - This makes it easy for the language to allow line separators, extra spaces and tabs used for indentation, etc., to have no impact on the legality of a program.
Resources

• Wikipedia: Lexical Analysis