Source:

1 | The Role of the Lexical Analyzer

1.1 | Lexical vs Syntactic analysis

1. Syntax and raw text are different and can be treated separately
2. it may be more efficient
3. better portability

1.2 | terms: tokens, patterns, lexemes

- #definition token: is a name and a value, where the name like a keyword or an identifier and the value is a section of the source text?
- #definition pattern: basically a regex of what string structures are allowed
- #definition lexeme: part of the source text that is matched by a pattern as an instance of a token

1.3 | common token breakdown

1. keywords (usually one per keyword)
2. operators (sometimes in operator classes)
3. identifiers
4. constants (sometimes one per type)
5. punctuation (usually one per each, including parens, comma, and semecolon)

1.4 | token attributes

- Token name only contains what type of token it is, not the value
  - if the token is "number", then what number actually was it?
- "token name influences parsing decisions, while the attribute value influences translation of tokens after the parse."
- the identifier token **id** needs to associate lots of data, such as it's lexeme, type, and location in memory, etc

1.5 | lexical errors

Sometimes we can modify the source to attempt to fix typos, etc. Such as removing some letters, edit distance, etc.

2 | Input Buffering

#todo-learn

3 | specification of tokens

3.1 | strings and languages (many definitions)

3.1.1 |#definition alphabet

a set of characters. examples include the *binary alphabet* $\{0, 1\}$, ASCII, and Unicode

3.1.2 |#definition string

a string over an alphabet is a ”finite sequence of symbols” from that alphabet. It’s length $|s|$ is the number of symbols in s. ϵ is the empty string.

3.1.3 |#definition language

countable set of strings over some fixed alphabet. Some languages are abstract, like or ϵ are boring languages.

3.2 | operations on languages