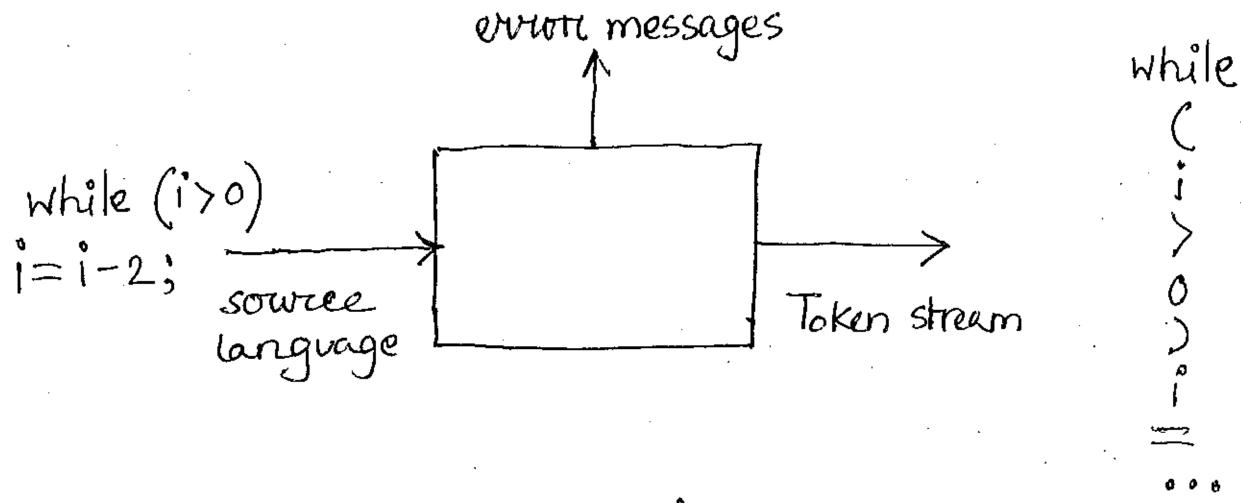
LEXICAL AND SYNTAX ANALYSIS

Lexical Analysis:

- Essentially a pattern matcher
  - attempts to find a substring from a given string of characters that matches a given character pattern.

- Serves as the front-end of a syntax analyzer.
  - Lexical analysis is a part of syntax analysis
  - Lexical analyzer performs syntax analysis at the lowest level of program structure.



- Stream of characters from source code is read left-to-right
- Grouped into tokens → sequence of characters with collective meaning.

Constants	operators	punctuation	Reserved words
integer, double	arithmetic, relational logical	, ; :	while, if, else

[Ref. Stanford lecture 7

卬 Lexical analysis

$$\begin{tikzcd}
\text{Source code} \arrow[d] \\
\boxed{\text{Lexical analysis}} \arrow[d] \\
\boxed{\text{Parsing}} \arrow[d, "\text{Abstract Syntax Tree (AST)}"] \\
\boxed{\text{Intermediate code Generation}} \arrow[d, "\text{Intermediate code}"] \\
\boxed{\text{Code generation}} \arrow[d] \\
\text{Assembly code}
\end{tikzcd}$$

W We already know –

There are three different approaches to implement programming language –

① Compilation ② Pure interpretation
③ Hybrid

→ All these three use both lexical and syntax analyzer

卬 Most-common Syntax-description Formalism

BNF
↳ Backus-Naur Form

* Clear and concise
* Can be used as the direct basis for syntax analyzer
* Easy to maintain because of modularity

⊞ Lexical analysis is separated from syntax analysis

There are three main reasons:

Simplicity: Techniques for lexical analysis are simpler than the other tasks required for syntax analysis

So, removing the easier and less-complex tasks from syntax analyzer make the syntax analyzer smaller and less complex.

Efficiency: Generally, lexical analysis requires longer time; infact, significant portion of the total compilation time.

↳ removing/separting it from syntax analyzer ~~p~~ ensures selective optimization.

Portability: Lexical analyzer is ~~machine~~ platform - dependent, whereas the syntax analyzer is ~~machine~~ platform - independent.

So, it is good to isolate the platform dependent part of any software system.

Also, lexical analyzer often buffer the inputs.

4 Lexical analyzer takes source program as input, and produces stream of tokens as output.

↳ Lexical analysis is also known as scanning.

✓ Lexical Analysis :

- Sentence consists of string of tokens
  - → number
  - → identifier
  - → keyword
  - → string

- Construct constants :

  25 becomes ⟨num, 25⟩

  For example, convert a number to token "num" and pass the value as the attribute of the token.

- Recognize keywords and identifiers

  Keywords : reserved words — do have special meaning that are already defined.

  For instance, C has 32 reserved words or keywords
  int, struct, long, switch, if, else etc.

  Identifiers : Generally, they are the given names to variables, constants, functions etc.

- Example:

  Recognize keyword and identifiers

  counter = counter + increment

  After lexical analysis, this becomes

  id = id + id

- Lexical analysis discards things do not contribute to parsing
  - ✓ White spaces (tabs, newlines, blanks)
  - ✓ Comments

- Implementation requires buffer
  - inputs are kept in buffer
  - Move pointers over the input

- Scanner/lexical analyzer could be implemented in alternative ways:
  - ✓ By using assembly language
    - efficient but complex; difficult to implement
  - ✓ By using high-level language such as C
    - efficient, but difficult to implement.
  - ✓ Use tools like lex, flex
    - easy to implement, but not efficient.