# Lexical analysis/scanning

Software that performs *lexical analysis* is called a *scanner*, *lexer*, or *tokenizer*.

The scanner's job is to break the input into a sequence of tokens for the parser to process.

The two most important calls to scanner software are

`next()`, which returns the next token without removing it from the token sequence, and

`match(s)`, which removes the next token from the token stream if it matches `s` and throws an exception otherwise.

A parser will call `next` whenever it wants to know the next token, and `match` whenever it wants to remove the next token.

---

Some scanners are written by hand as essentially a bunch of 'if' statements, especially if the tokens are easy to identify and return.

But, there are also tools called *scanner generators* that allow you to specify what your tokens look like using regular expressions and then the tool produces scanner code that, when run, return tokens according to your regular expressions.

For example a (fictional) *lexical specification* like

```
KEYWORD: (aaa+bbb)
ID: (a+b)*
```

specifies that "aaa" and "bbb" are tokens of type KEYWORD because they match the first regular expression, and every sequence of a's and b's is a token of type ID because it matches the second regular expression.

If a string matches multiple regular expressions, the token type it gets assigned to is the first one listed. So, even though "aaa" matches both regular expressions, it would be considered to be a token of type KEYWORD.

The process for tokenizing an input string based on a set of regular expressions goes like this. To find the next token string (also called a *lexeme*) and token type:

1. Remove any whitespace from the beginning of the input (most languages do not care about whitespace).
2. Find and remove from the input character sequence, the longest prefix of the remaining input that matches any of the regular expressions.
3. Return the found prefix as the next token and the token type of the regular expression that it matched. If more than one regular expression matches, return the type of the regular expression listed first in the specification.

**Example:** For input "aa aaaa aaa" the above specification would first match the initial "aa" with (a+b)* and return ("aa", ID). Note that the space keeps longer strings from matching. For the next token, the space would be skipped and "aaaa" would match (a+b)* and return ("aaaa", ID). For the final token, the space would be skipped and "aaa" would match both (aaa+bbb) and (a+b)*, but because (aaa+bbb) is listed first in the lexical specification the token and type returned would be ("aaa", KEYWORD).

---

So, a scanner generator takes a lexical specification and produces some code that can then be used as a scanner. How do these scanner generators work? They...

1. Turn each regular expression into an NFA.
2. Create a new start state with a lambda transition to each NFA's start state.
3. Convert the resulting NFA into a DFA.
4. Any final state in the DFA should indicate which token type it came from (and if the final state comes from more than one DFA, list only the first from the lexical specification).

Here is the resulting DFA from the lexical specification in the example. On top are the two NFAs for KEYWORD and ID with a new start state. Below is the result of the NFA-to-DFA conversion process. In this example, every NFA state is an accept state, and every accept state is for token type ID except for the states labeled "5,9" and "8,9". (Errata: The DFA states labelled 9 should be merged together.)