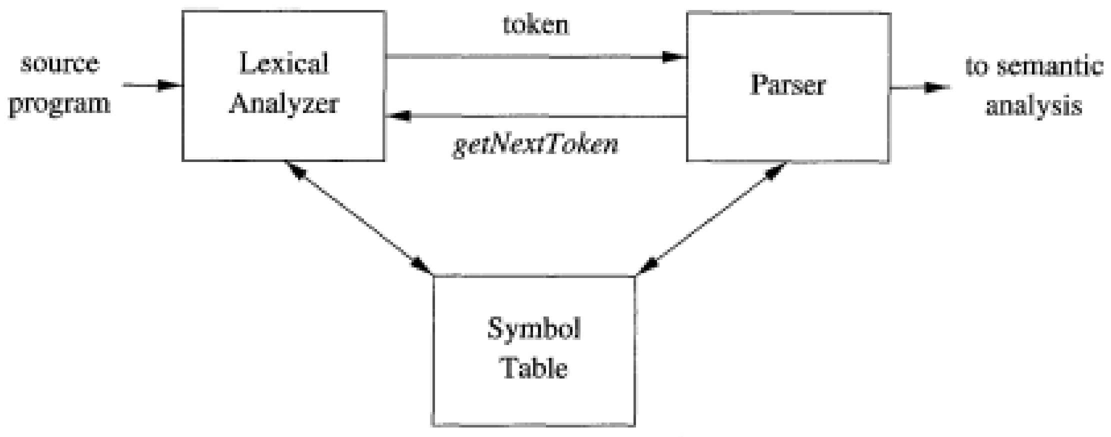
Lexical Analysis



As the first phase of a compiler, the main task of the lexical analyser:

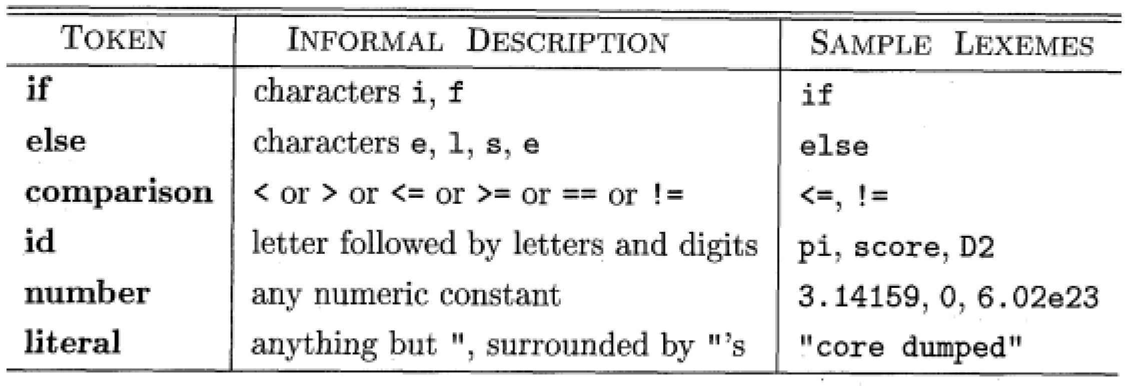
a. read the input characters of the source program, for eg. 'i' , 'n' , 't', ' ' , 'm' , 'a', 'i' , 'n', '(' , ')' , ...
b. group them into lexemes. For eg. int main() …
c. produce as output a sequence of tokens for each lexeme in the source program. For eg. <int>, <id,1> , …
d. The stream of tokens is sent to the parser for syntax analysis.
e. the lexical analyzer may keep track of the number of newline characters seen, so it can associate a line number with each error message.
f. If the source program uses a macro-preprocessor, the expansion of macros may also be performed by the lexical analyser. For eg.

```
#include<stdio.h> ===> File Inclusion
#define x = 6;      ===> macro expansion
```

Tokens, Patterns, and Lexemes

1. *A token* is a pair consisting of a token name and an optional attribute value, i.e. <token-name, attribute-value>. The token name is an abstract symbol representing a kind of lexical unit, e.g., a particular keyword, or a sequence of input characters denoting an identifier. The token names are the input symbols that the parser processes. Eg. int x = a - -b + c
2. *A lexeme* is a sequence of characters in the source program that matches the **pattern** for a token and is identified by the lexical analyzer as an instance of that token.
3. *A **pattern*** is a description of the form that the lexemes of a token may take.
   a. keywords: sequence of characters
   b. identifiers: starts with any alphabet or an underscore followed by any number of alphanumeric characters. For eg. abc, _abc123
   c. numbers: any sequence of digits where each digit can be from 0 to 9.

d. Operators: [+,-,*,/….]



TOKEN	INFORMAL DESCRIPTION	SAMPLE LEXEMES
if	characters `i`, `f`	`if`
else	characters `e`, `l`, `s`, `e`	`else`
comparison	`<` or `>` or `<=` or `>=` or `==` or `!=`	`<=`, `!=`
id	letter followed by letters and digits	`pi`, `score`, `D2`
number	any numeric constant	`3.14159`, `0`, `6.02e23`
literal	anything but `"`, surrounded by `"`'s	`"core dumped"`

Q1. Find the number of tokens in the following:

a.
```
main(){
  printf("cd");
  // prints the message
}
```

main, (,), { , printf, (, "cd" ,),;, } → 10 tokens

b.
```
while(i>0){
    printf(i);
    i++;
 }
```
while, (, i,>,0,),{,printf,(,i,), ;, i, ++,;, } → 16 tokens