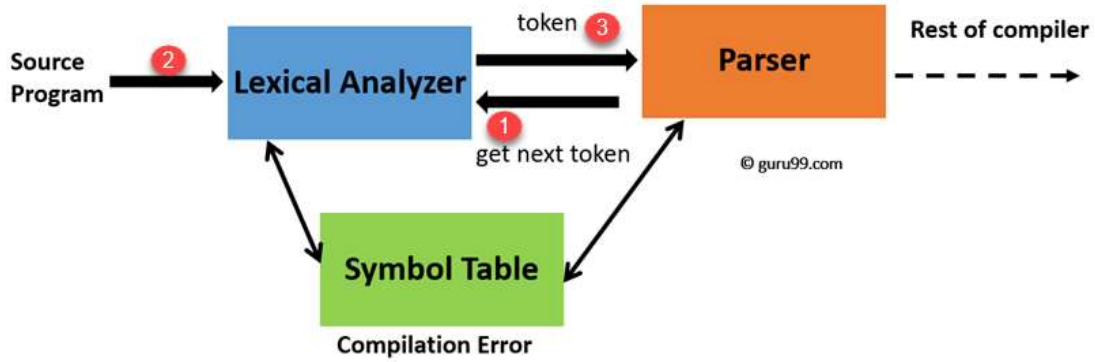
Role of Lexical Analyzer (Scanner)

- 1st phase in compilation → seq. of character to tokens
- Extra space, comments ← remove
- Preprocessor →Modified source program → lexemes → tokens



- Token → lexeme (sequence of character)-meaningful sentence
- Example → int a=10;
- Specification of Token → Alphabet, String, language

Lexical Analyzer – Basic Terminologies

- *Lexeme*
  - Sequence of characters in source program that matches a pattern for a token
- *Token*
  - Valid sequence of characters given by lexeme
  - Keyword, constant, identifier, operators, symbols
- *Pattern*
  - Rule that must be matched by the sequence of character (lexeme) to form the token
- *Example*
  - int a=10 →lexeme → int , token→keyword

Lexical Analyzer- Example

```
#include <stdio.h>
    int maximum(int x, int y) {
        // This will compare 2 numbers
        if (x > y)
            return x;
        else {
            return y;
        }
    }
```

Lexeme	Token
int	Keyword
maximum	Identifier
(	Operator
int	Keyword
x	Identifier
,	Operator

Lexical Analyzer- Example (Non Tokens)

Type	Examples
Comment	// This will compare 2 numbers
Pre-processor directive	#include <stdio.h>
Pre-processor directive	#define NUMS 8,9
Macro	NUMS
Whitespace	/n /b /t

Lexical Analyzer- Roles

- Helps to identify *token* into the symbol table
- Removes *white spaces and comments*
- Error message
  - Misspelling of identifiers, operators, keyword are considered as *lexical errors*
  - Error recovery
    - Replace a character with another character
    - By inserting the missing character into the remaining input
- *Expands the macros*

Lexical Analyzer