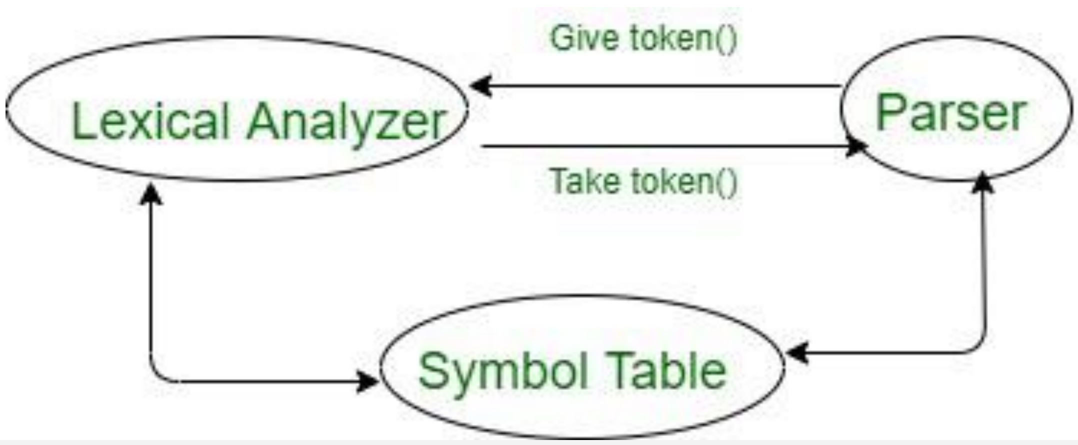
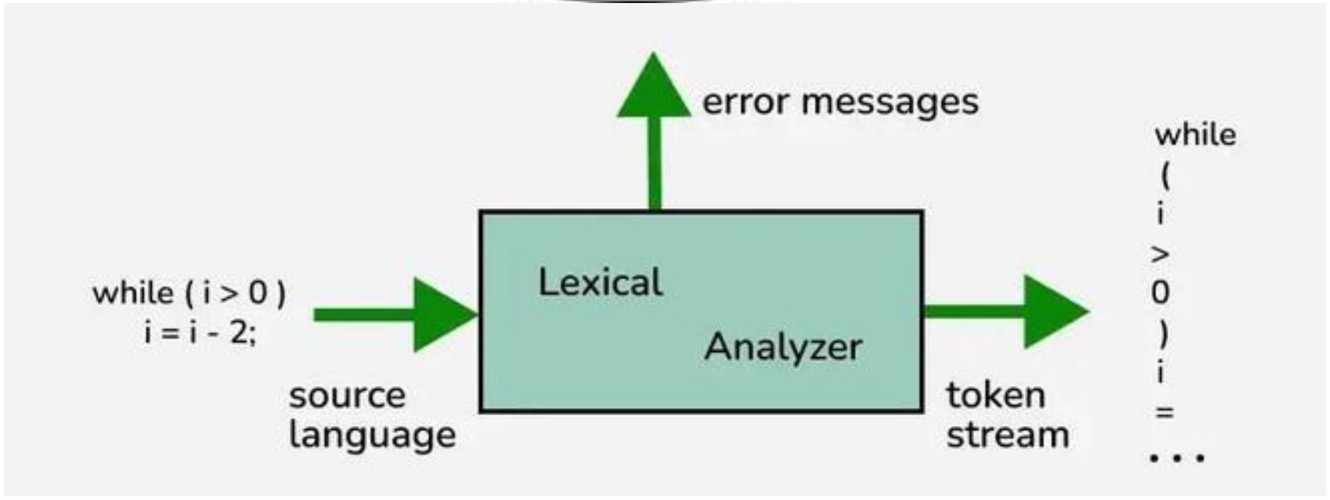
❖Lexical analysis: / Scanning

1. First phases of compilation process. It takes pure HLL as input and covert it into stream of tokens of meaningful lexemes.
2. It is basically scanning or tokenizing process.





sources code --> lexeme --> tokens --> syntax parser

3. Remove all the white spaces
4. Remove all the comments
5. Identify errors with the help of automata machine
6. Give error message when the lexeme doesn't match any pattern. Types of error are:-
    a. ***Exceeding length***
    b. ***Unmatched string***
    c. ***Illegal character***
7. Types of Tokens:
    Keyword = if, for, while, do
    Identifier = variable or function name
    Operator = +, 0, /, %, ?, ++, --, =, <,> etc
    Separator = ',' , ';' (comma, semicolom), **{}, [], ()**
    Constant = 20,30, value
    Special character = $, # etc

Ex-1:

```
int main()
{
  // 2 variables
  int a, b;
  a = 10;
 return 0;
}
```

All the valid tokens are:
‘int’ ‘main’ ‘(‘ ‘)’ ‘{‘ ‘int’ ‘a’ ‘,’ ‘b’ ‘;’
‘a’ ‘=’ ’10’ ‘;’ ‘return’ ‘0’ ‘;’ ‘}’

Ex-2:

```
main(){
int a=10;
char b="abc";
in t c = 30;
ch ar d= "xyz";
in/*comment*/t m=40.5;
}
```

tokens:33

Exercise 3: Count number of tokens:

```
int main()
{
int a = 10, b = 20;
printf(“sum is:%d”,a+b);
return 0;
}
```

Answer: Total number of token: 27.

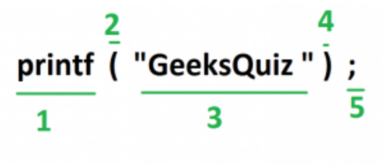
Exercise 4: Count number of tokens:
int max(int i);
Answer:
Total number of tokens 7

Ex-5:

```
int main)(
}
  x = y + z;
  int x, y, z;
  print("Goto GFG %d%d", a);
  {
```

ans: 26

Ex-6:



Example 7:

```
int main() {
  // printf() sends the string inside quotation to
  // the standard output (the display)
  printf("Welcome to GeeksforGeeks!");
  return 0;
}
```

Total number of tokens = 14

Ex-8:

```
int main()
{
 int x = 0, y = 5;
printf(“First number is %d and the second number %d”, x, y);
return 0;
}
```

Ans: 27

Ex-9:

```
int main)(
{
 x = y+z;
 int x,y,z;
 Printf("Sum %d %d", x);
{
```

tokens: 26

Ex-10:

```
main(){
a=b+++-- --+++==;
Printf("%d %d,a,b);
}
```

tokens:17

Ex-11:

```
main(){
a=b+++-- --+++==;
Printf("%d %d",a,b);
}
```

tokens:27

Ex-12:

```
int a = 10;
```

Answer – Total number of tokens = 5

Ex-13:

```
switch(input)
{
 case 1: b = c*d; break;
 default: b = b++; break;
}
```

tokens: 26

Ex-14:

```
printf("i = %d, j=%f, &i=%x]n", i,j, &i);
```

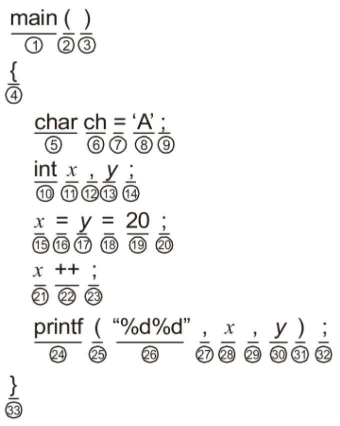
tokens: 12

Ex-15:

```
/*abc*/Printf("whats up %d", ++&&***a);
```

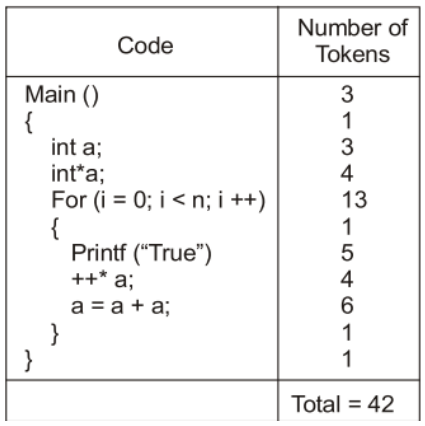
tokens:12

Ex-16:



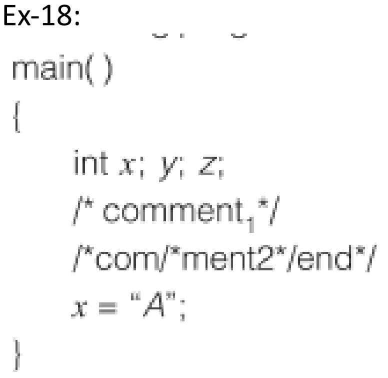
```
main ( )
 ①  ②③
{
④
    char ch = 'A' ;
     ⑤   ⑥⑦ ⑧⑨
    int x , y ;
    ⑩ ⑪⑫⑬⑭
    x = y = 20 ;
    ⑮⑯⑰ ⑱ ⑲ ⑳
    x ++ ;
    ㉑ ㉒ ㉓
    printf ( "%d%d" , x , y ) ;
      ㉔   ㉕   ㉖    ㉗㉘㉙㉚㉛㉜
}
㉝
```

Ex-17:



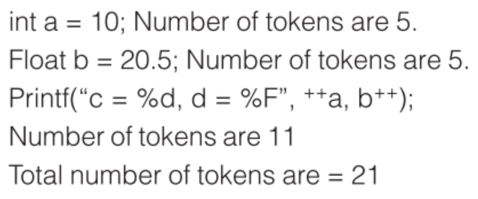
Code	Number of Tokens
Main ()	3
{	1
int a;	3
int*a;	4
For (i = 0; i < n; i ++)	13
{	1
Printf ("True")	5
++* a;	4
a = a + a;	6
}	1
}	1
	Total = 42

Ex-18:



```
main( )
{
    int x; y; z;
    /* comment1*/
    /*com/*ment2*/end*/
    x = "A";
}
```

Ex-19:



int a = 10; Number of tokens are 5.

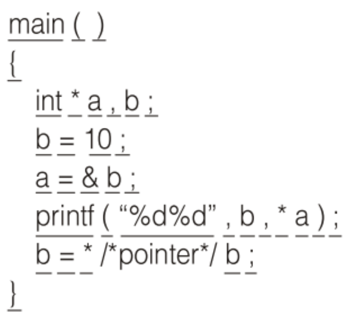
Float b = 20.5; Number of tokens are 5.

Printf("c = %d, d = %F", ++a, b++);

Number of tokens are 11

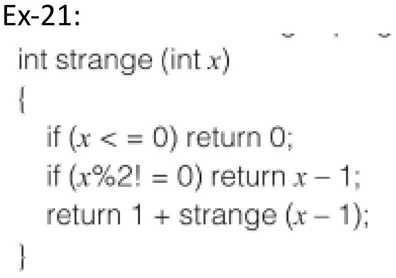
Total number of tokens are = 21

Ex-20:



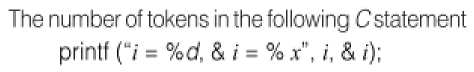
```
main ( )
{
    int * a , b ;
    b = 10 ;
    a = & b ;
    printf ( "%d%d" , b , * a ) ;
    b = * /*pointer*/ b ;
}
```

Ex-21:



```
int strange (int x)
{
    if (x < = 0) return 0;
    if (x%2! = 0) return x – 1;
    return 1 + strange (x – 1);
}
```

Ex-22:



The number of tokens in the following *C* statement

printf ("*i* = %*d*, & *i* = % *x*", *i*, & *i*);

Ans : 10

Ex-23:

```
int main(){
 int i;
 for(i=0; i<5; i++) {
int i=102;
printf(" %d the value of i:", i);
i++;
 }
 return 0;
}
```

tokens: 42

Ex-24:

```
int main(){
 int x, a=2, b=3, c=5;
 x = a+b*c;
 printf("the value of x is %d", x);
 return 0;
}
```

tokens: 39

Lexemes	**Tokens**	**Lexemes Continued…**	**Tokens Continued…**
while	WHILE	a	IDENTIEFIER
(	LAPREN	=	ASSIGNMENT
a	IDENTIFIER	a	IDENTIFIER
>=	COMPARISON	–	ARITHMETIC
b	IDENTIFIER	2	INTEGER
)	RPAREN	;	SEMICOLON

Tokens, Lexemes, Patterns:

Tokens: A lexical token is a sequence of characters that can be treated as a unit in the grammar of the programming language.
Eg: identifiers, keywords, operators, special symbol, constants
Eg of non-tokens: comments, tabs, spaces, blanks, newline

Lexemes: A lexeme is a sequence of characters the input that match a pattern.

Patterns: A set of strings in the input for which the same token is produced as output. This set of string is described by a rule called a pattern associated with the token.