CS & IT ENGINEERING

Compiler Design

Lexical Analysis

One Shot

One Shot

By- Venkat sir

Topics to be Covered

GATE

- Topic — Lexical Alalysis
- Topic — Parsing (Syntax Analysis)
- Topic — Syntax Directed Translation (Semantic Analysis)
- Topic — Intemediate Code Genaration
- Topic — Code Optimisation(Dataflow Analysis)
- Topic — Runtime Environment

Topics to be Covered

Topic ?????

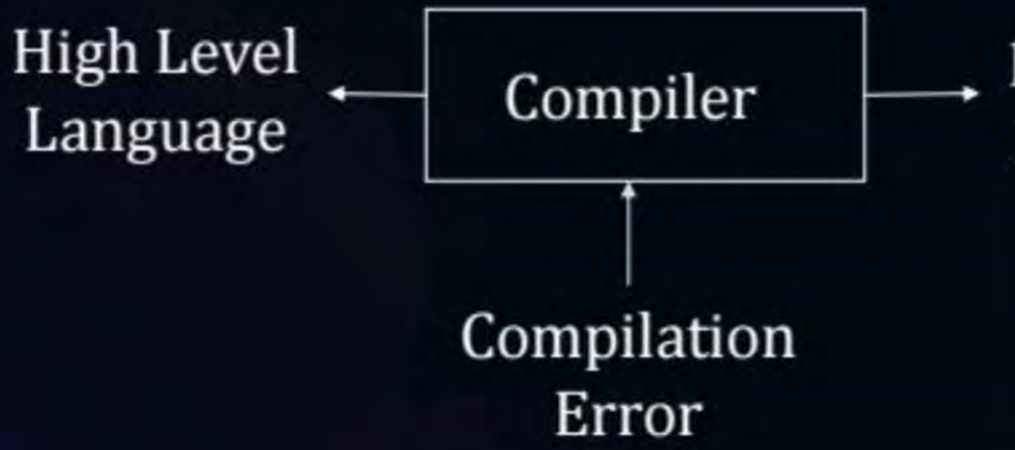
Lexical Analysis

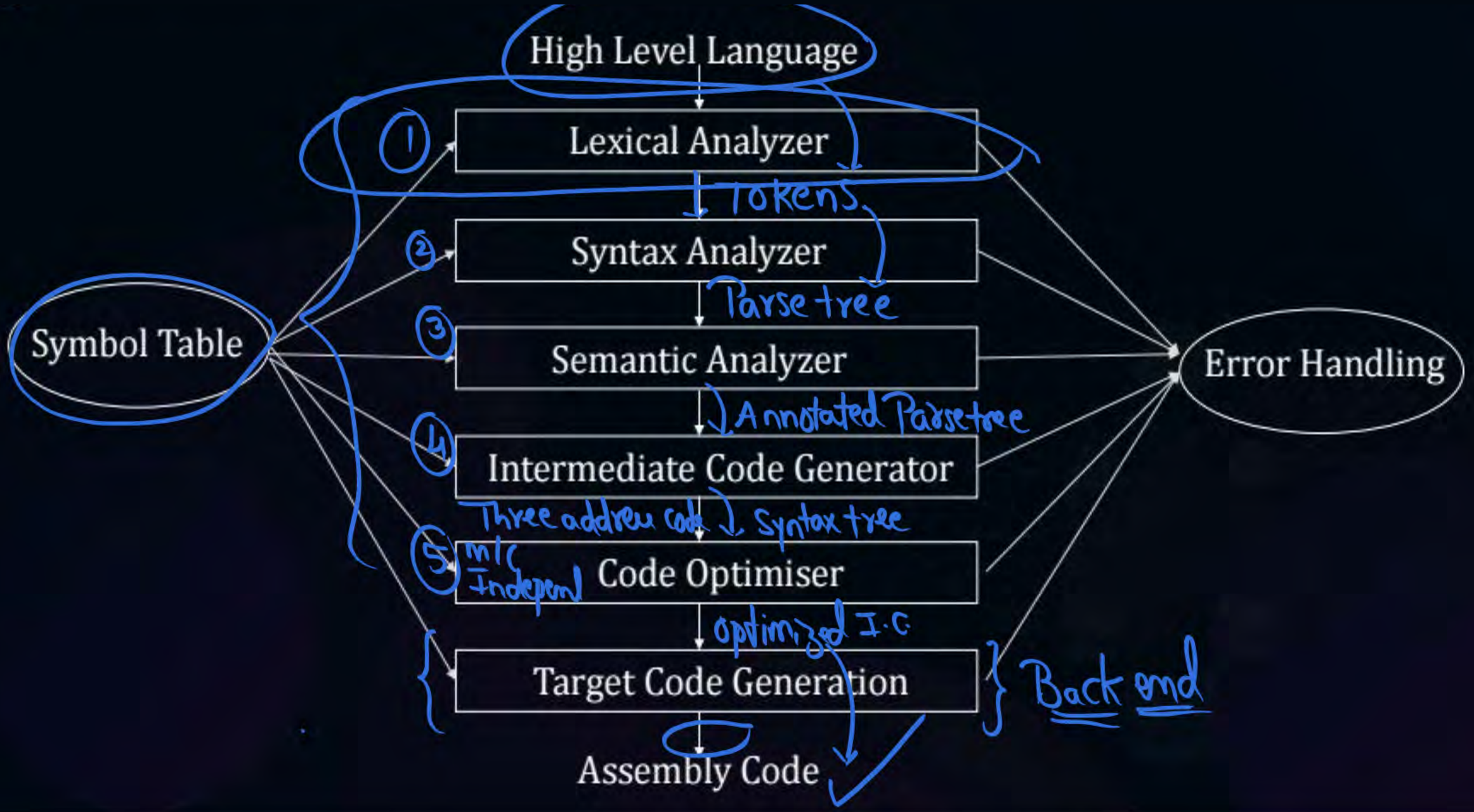
Topic : Compiler Design

- Lexical Analysis
- Parsing
- Syntax Directed Translation
- Intermediate Code Generation
- Runtime Environment

Topic : Introduction of Compiler Design

- Compiler is a translator which converts a program written in one language (Source Language) to other language (Object/Target Language).




High Level Language
Lexical Analyzer
Tokens
Syntax Analyzer
Parse tree
Semantic Analyzer
Annotated Parse tree
Intermediate Code Generator
Three address code
Syntax tree
m/c Independ
Code Optimiser
optimized I.C
Target Code Generation
Assembly Code
Symbol Table
Error Handling
Back end
1
2
3
4
5

Topic : Semantic Analysis Phase

- It is a program that takes parse tree as input and produces annotated parse tree as output.
- It also detects sematic errors present in the program.(type checking)

Topic : Intermediate Code Generation Phase

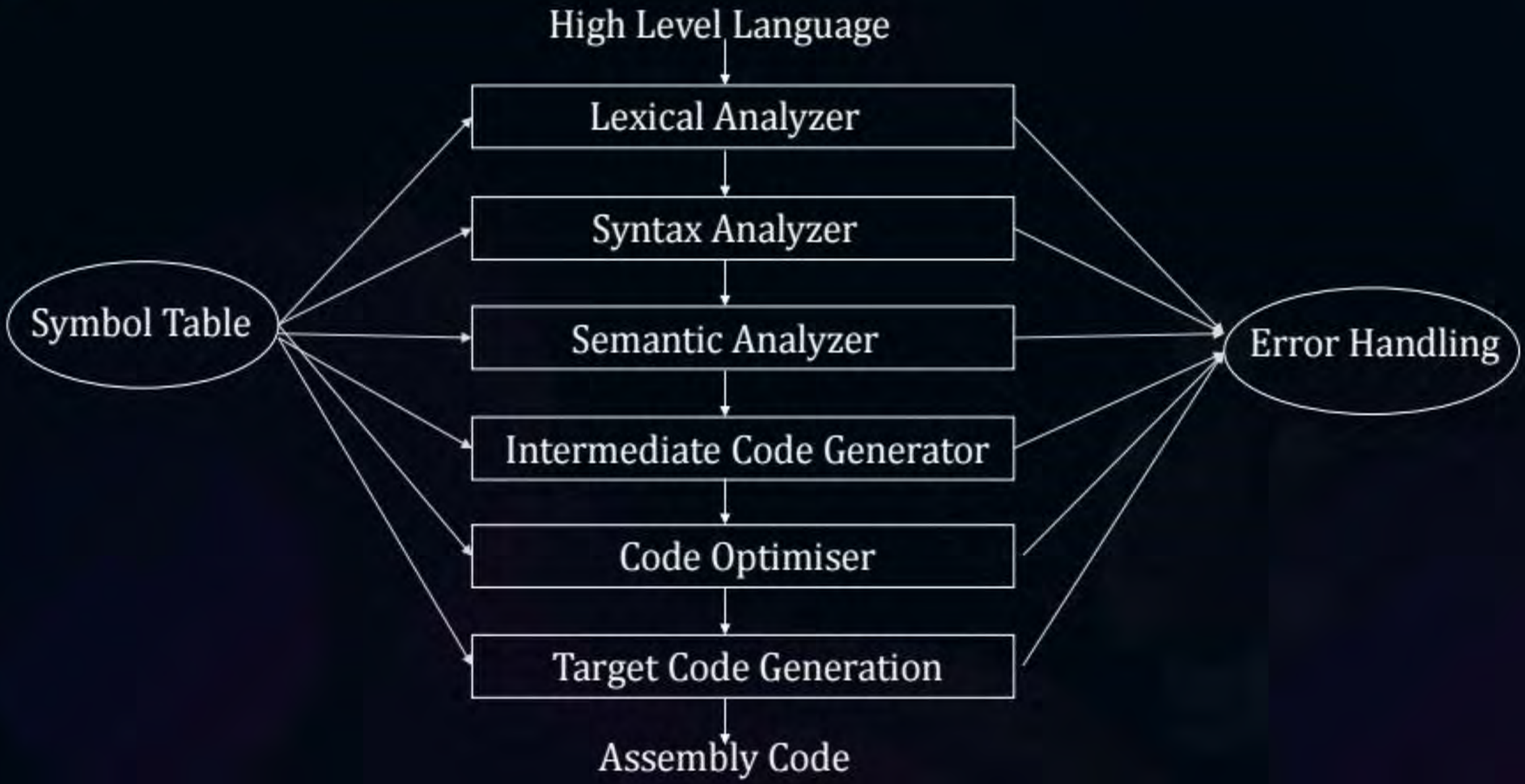
- It is a program that translates high level code into intermediate code.
- Advantage of generating intermediate code is to perform optimization in simple way.

Topic : Code Optimization Phase

- It is a program that reduces time and space required by the target machine by removing some unnecessary code .
- There are two types of optimizations performed by compilers known as machine independent optimization and machine dependent optimization.

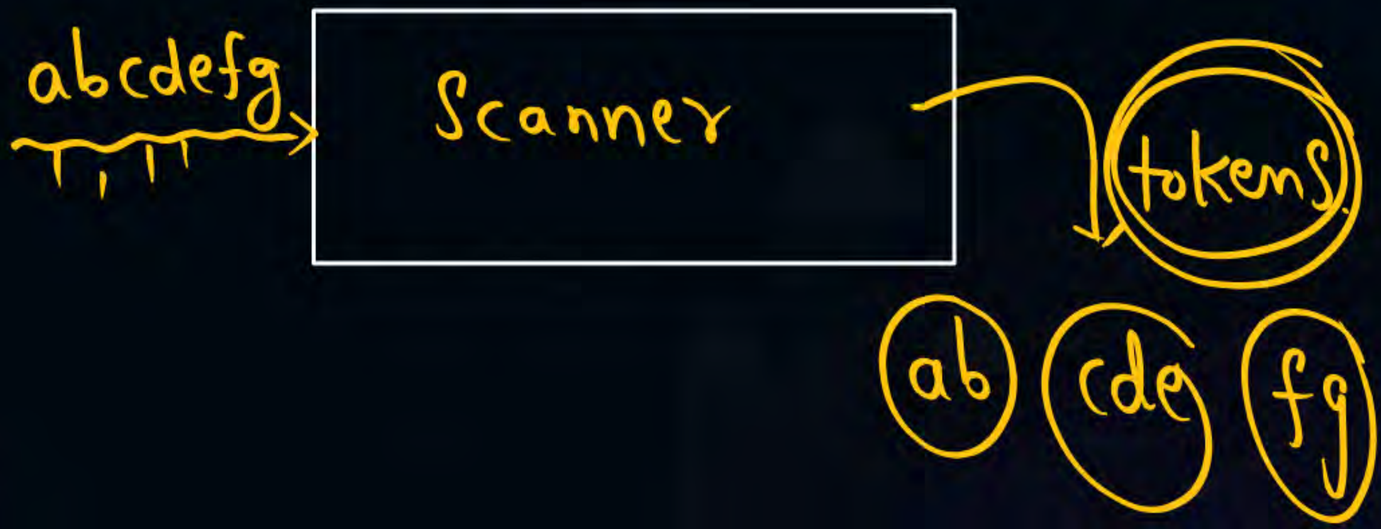
Topic : Code Generation Phase

- It is a program that translates optimized intermediate code into assembly language or target code.

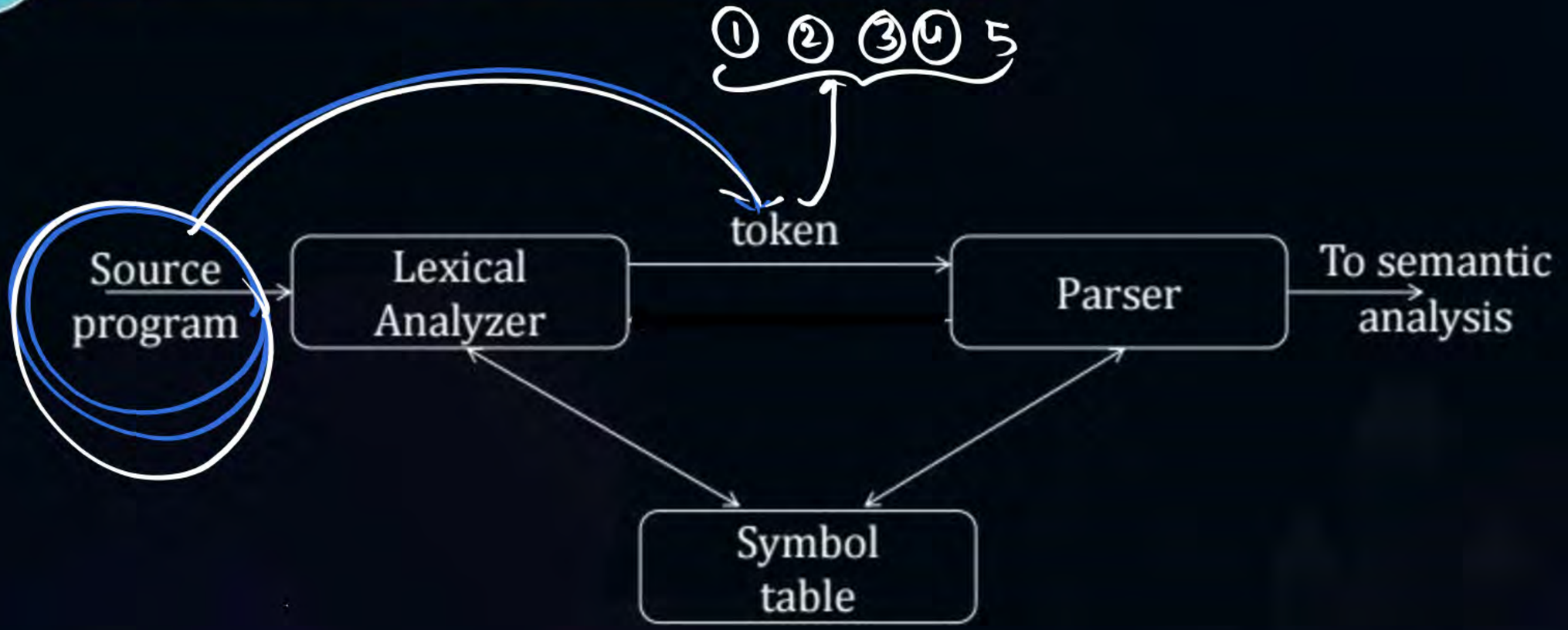

High Level Language
Lexical Analyzer
Syntax Analyzer
Symbol Table
Semantic Analyzer
Error Handling
Intermediate Code Generator
Code Optimiser
Target Code Generation
Assembly Code

Topic : Functionality of Lexical Analysis

1. It reads source program as stream of characters (Scanner) (abc)
2. It produces tokens as output
3. It detects lexical errors
4. It eliminates comment lines
5. It eliminates white space characters
- It constructs symbol table



Topic : The Role of Lexical Analyzer

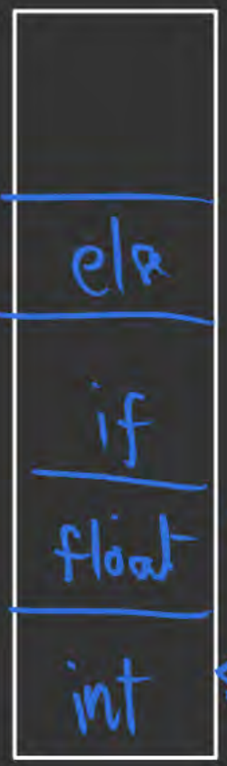


① Regular Expressions

② Finite Automata

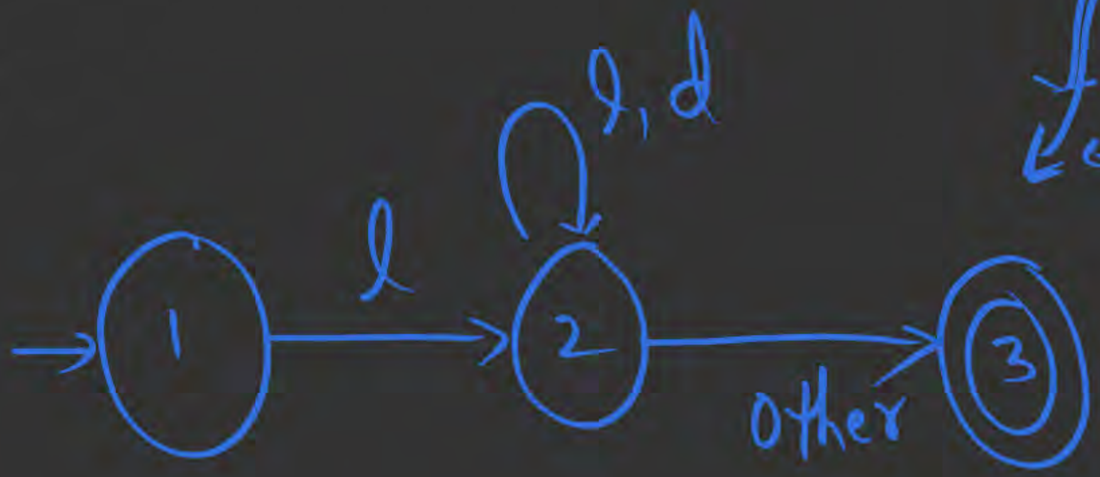
Identifier

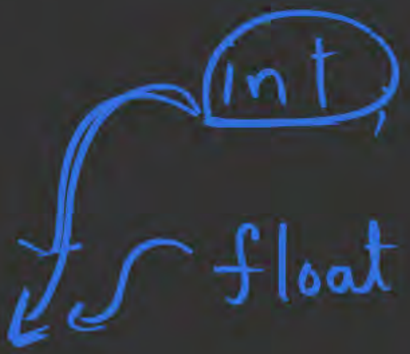
ID

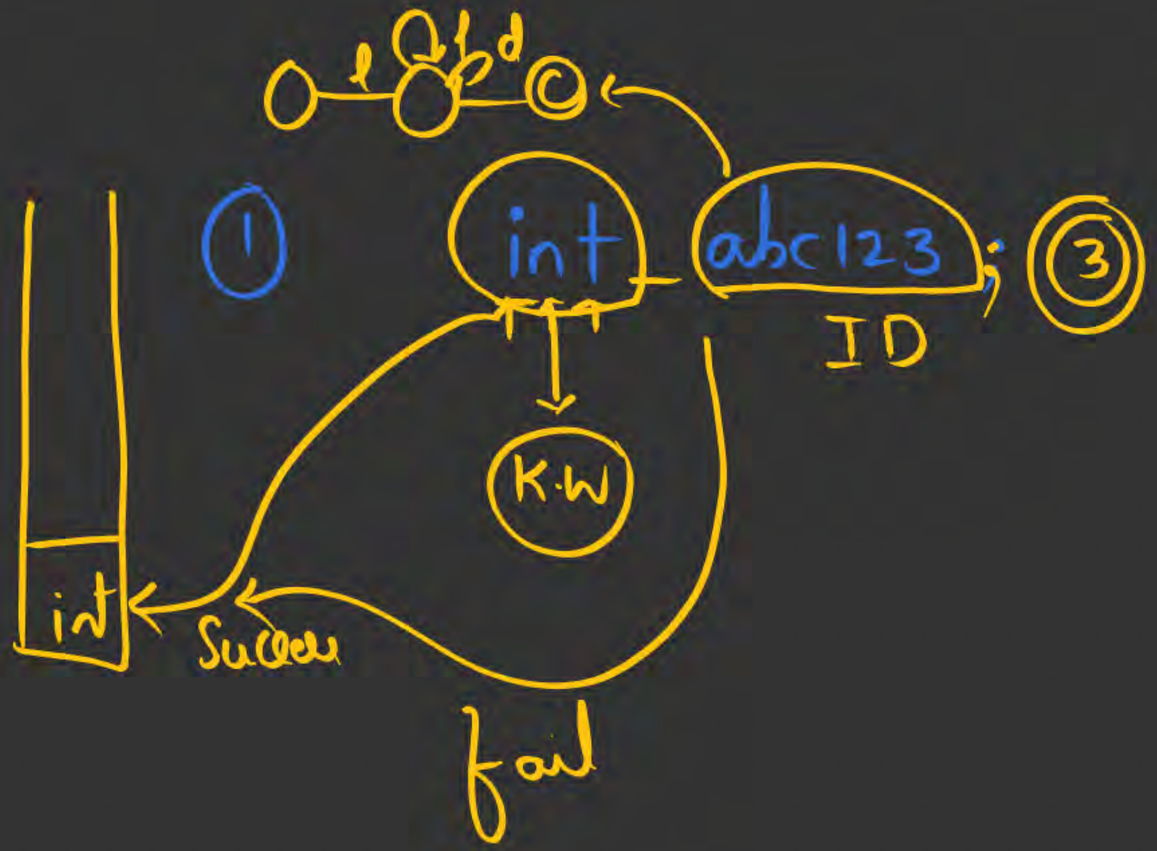


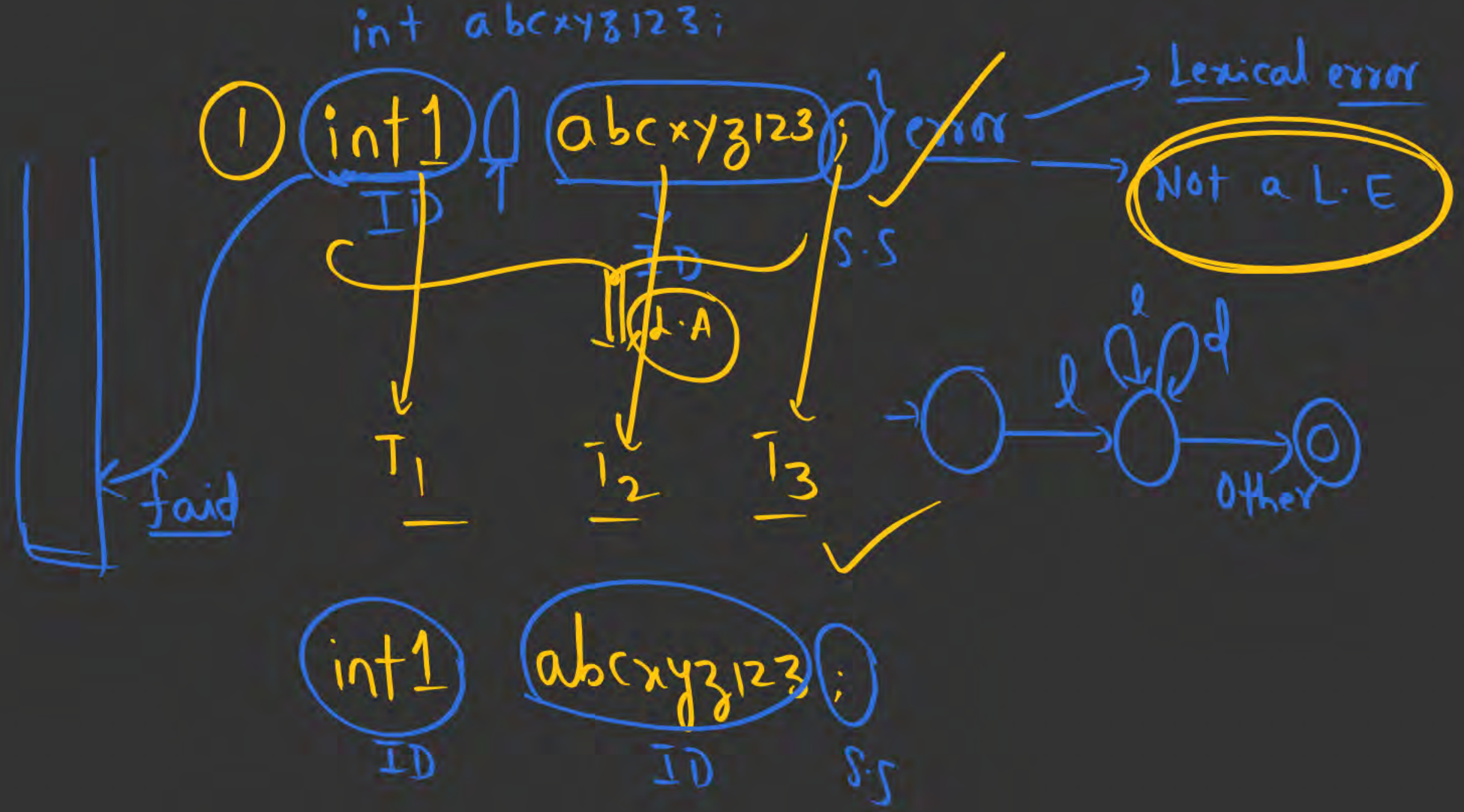
Succes K.W

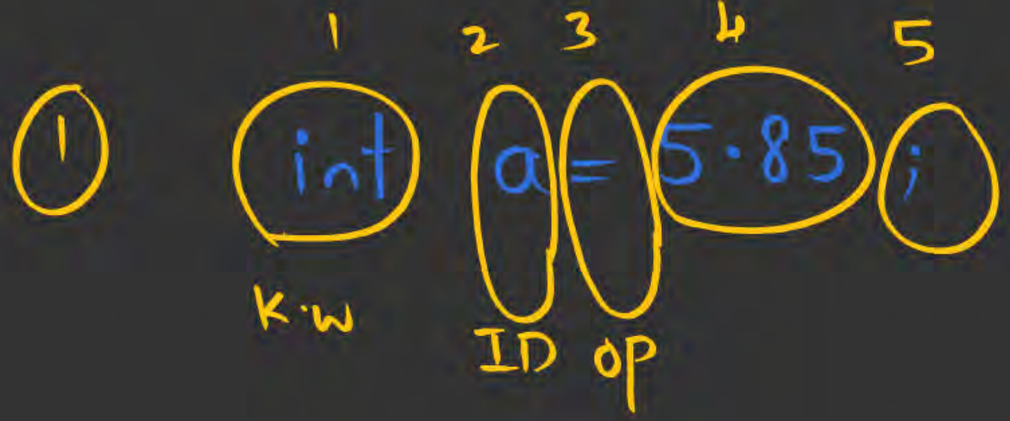
fail ID.






l
d
1
int
abc123
;
3
ID
K.W
int
Succes
fail


int abcxyz123;
1
int1
abcxyz123
;
error
Lexical error
Not a L.E
ID
ID
S.S
L.A
T1
T2
T3
faild
l
l
d
Other
int1
abcxyz123
;
ID
ID
S.S


1
2
3
4
5
1
int
a
=
5.85
;
k.w
ID
op

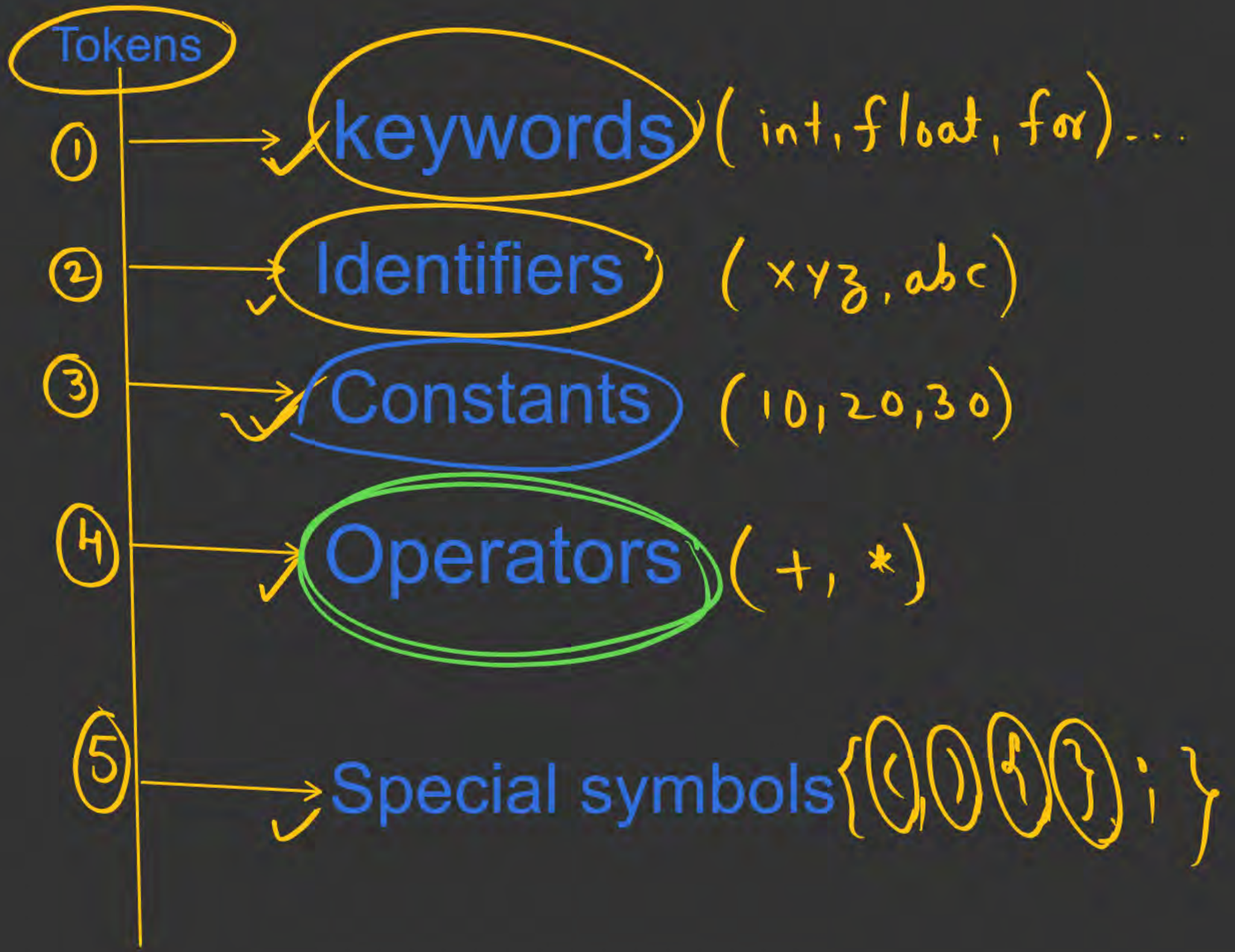
Topic : The Role of Lexical Analyzer

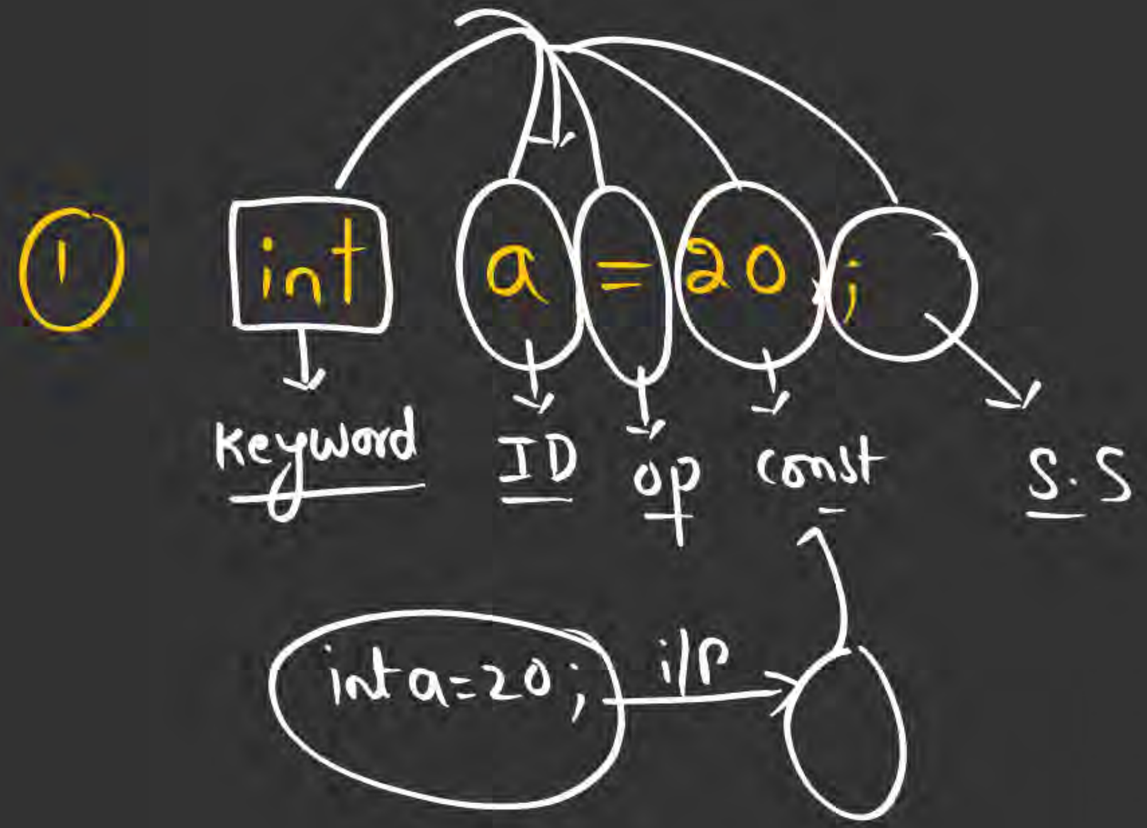
Why to separate Lexical analysis and parsing

- Simplicity of design
- Improving compiler efficiency
- Enhancing compiler portability

Topic : The Role of Lexical Analyzer

- **Token:** Token is a group of characters from source code that can be treated as a single logical entity.
- Typical tokens are,
- 1. Identifiers
- 2. keywords
- 3. operators
- 4. special symbols
- 5. constants


Tokens
1
keywords
(int, float, for) ...
2
Identifiers
(xyz, abc)
3
Constants
(10, 20, 30)
4
Operators
(+, *)
5
Special symbols
{ ( ) ( ) ; }



5 tokens

keywords (Reserve Words)

int	do
float	else
char	goto
for	while
if	short
goto	void.

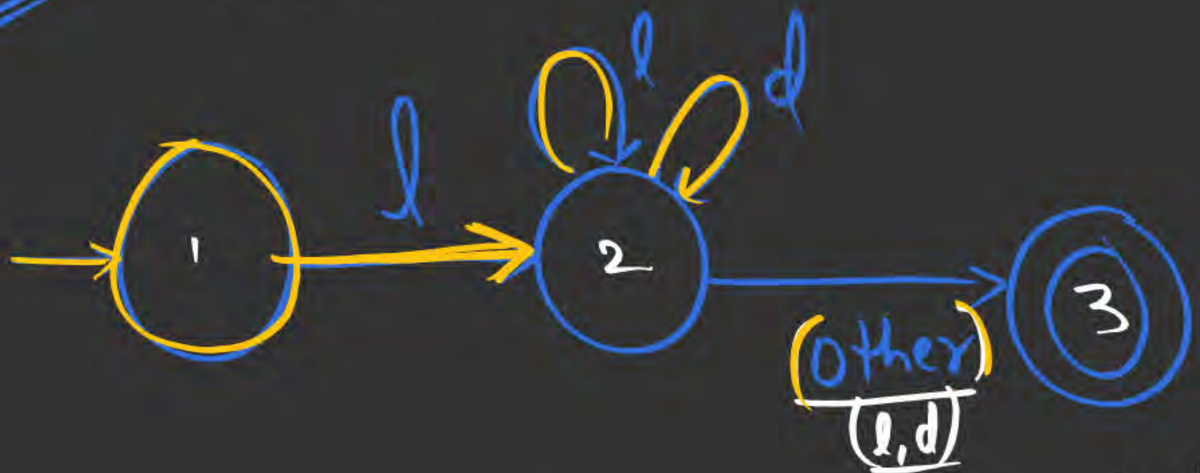
32

Identifier

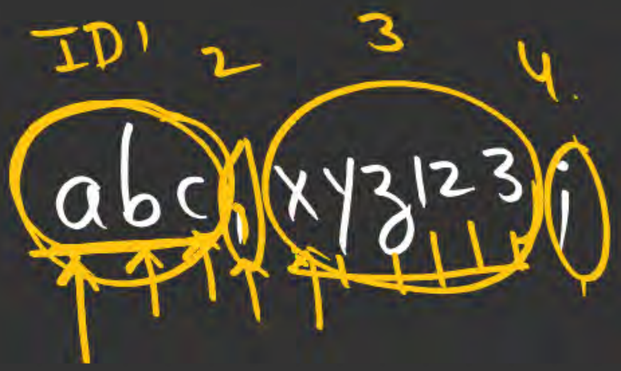
1. Identifier can be formed using letter and digit
2. Identifier can start with letter only (can't start with digit)
3. keyword name is not allowed as Identifier
4. Identifier can't include any Special characters (_)

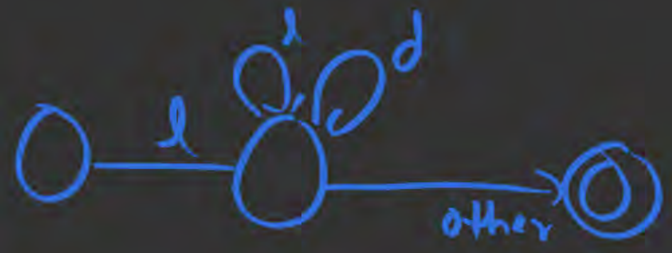
ID F.A

$l(l+d)^*$ other



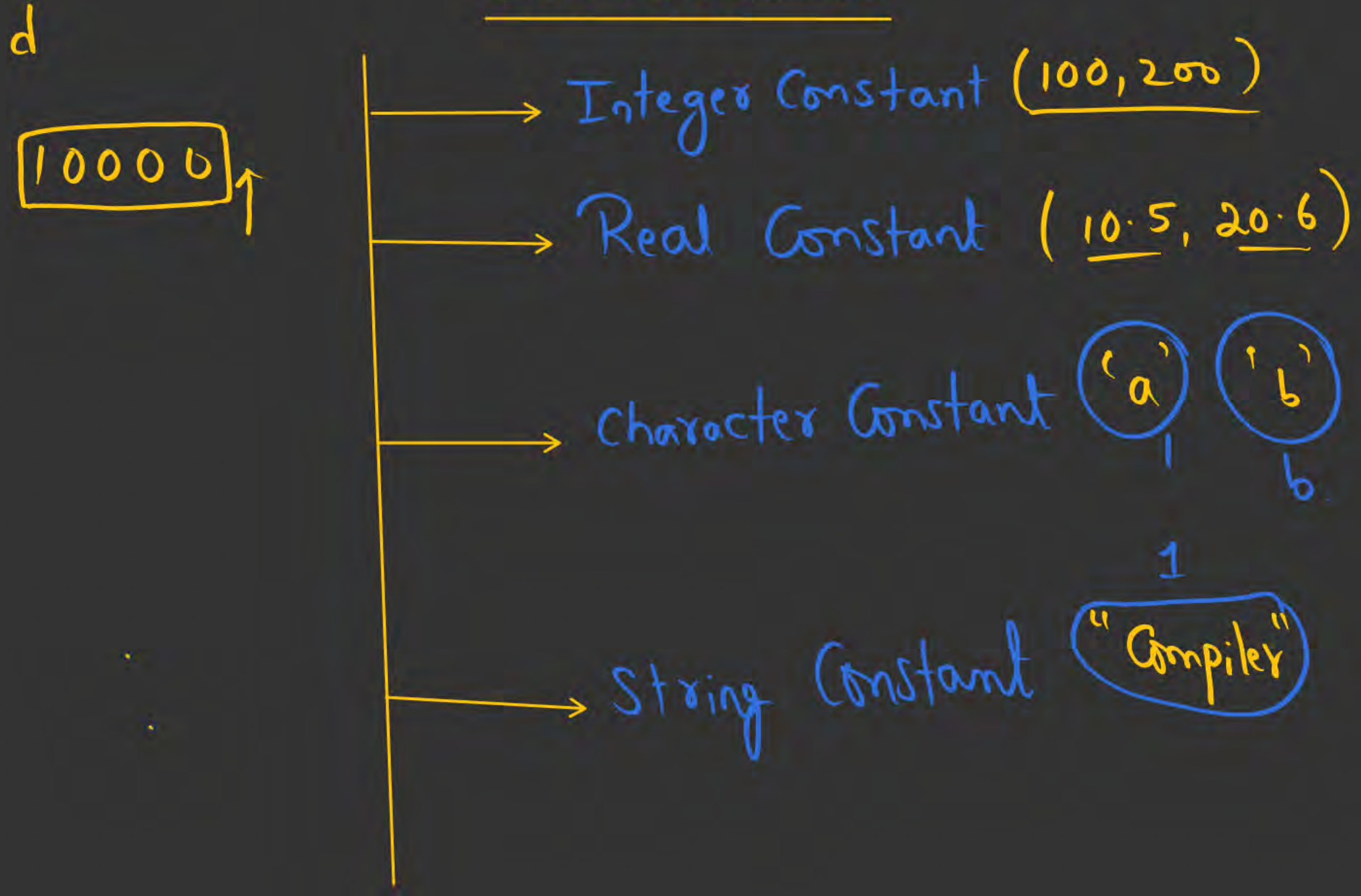
return ID




l
l
d
other


intfloatchar
;
-> 2
1
2

constants

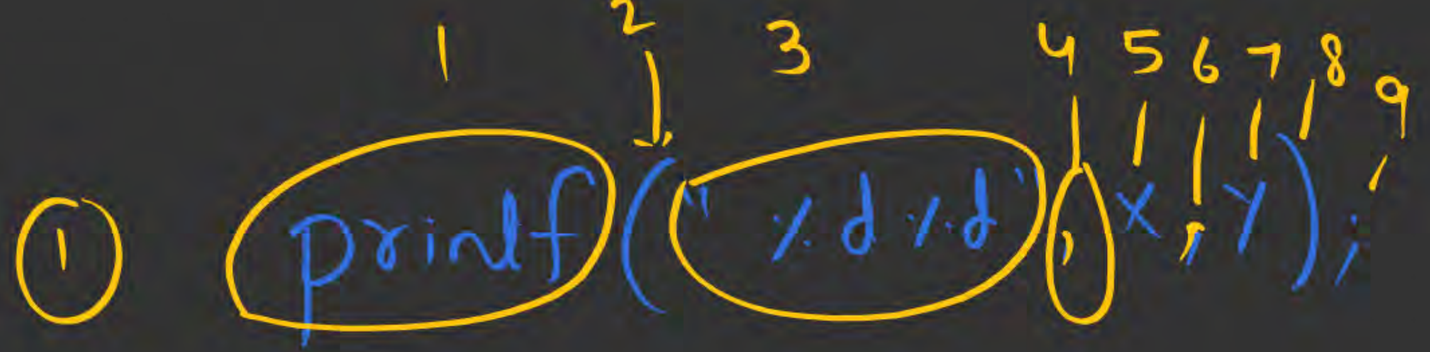

d
10000
Integer Constant (100, 200)
Real Constant (10.5, 20.6)
Character Constant 'a' 'b'
1
b
1
"Compiler"
String Constant

① int a = "Compiler" ; } 5

K.W ID OP

② int a = "Compiler" ; } NO Lexical error

③ int a = 20 + "gate" ; }


1
2
3
4
5
6
7
8
9
printf
"%d %d"
x
y

```
main()
{
① int a=10;
② int b;
③ char c="abc";
④ b=a+c;
}
```

(a) Lexical error

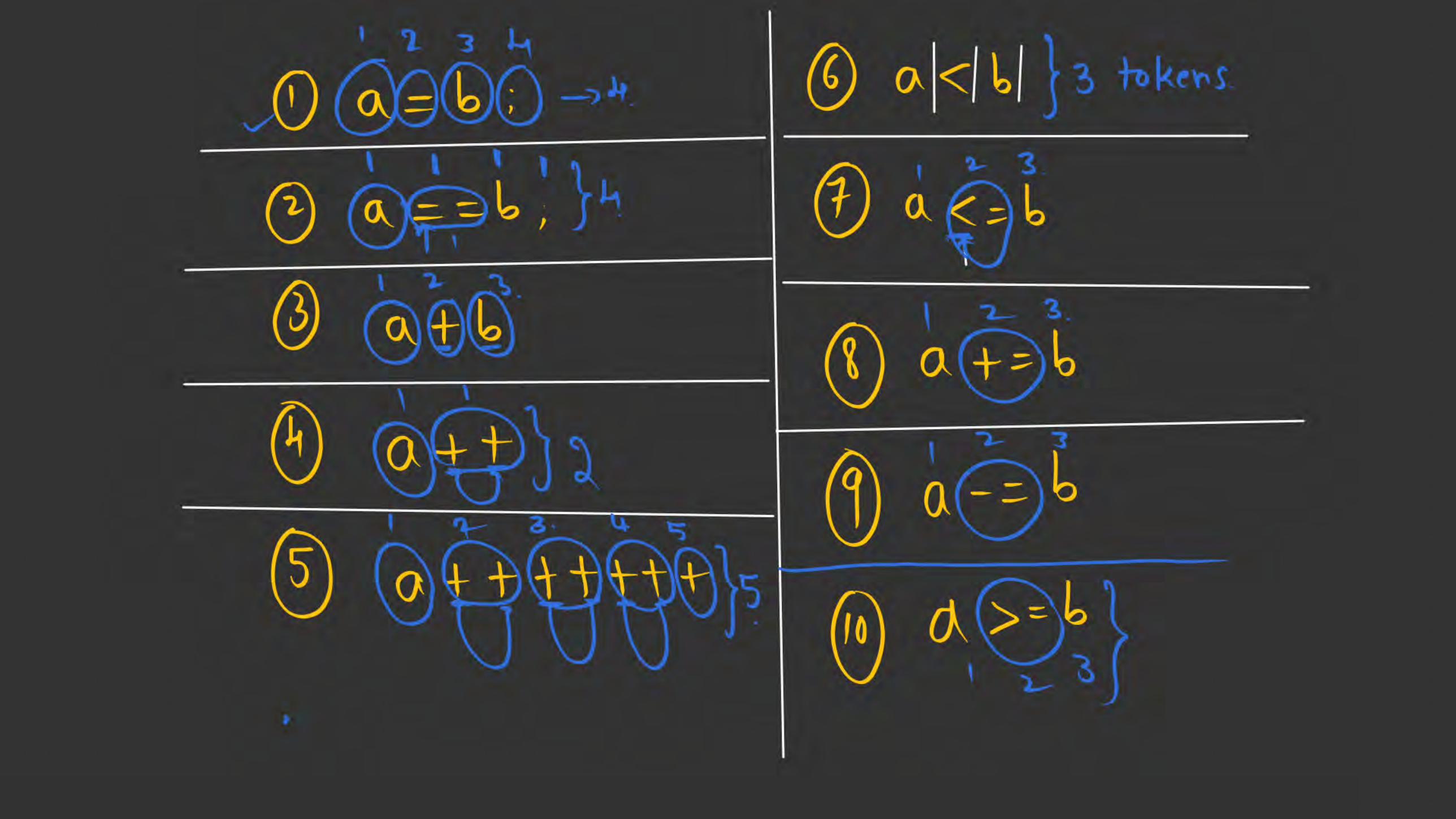
(b) NO lexical error

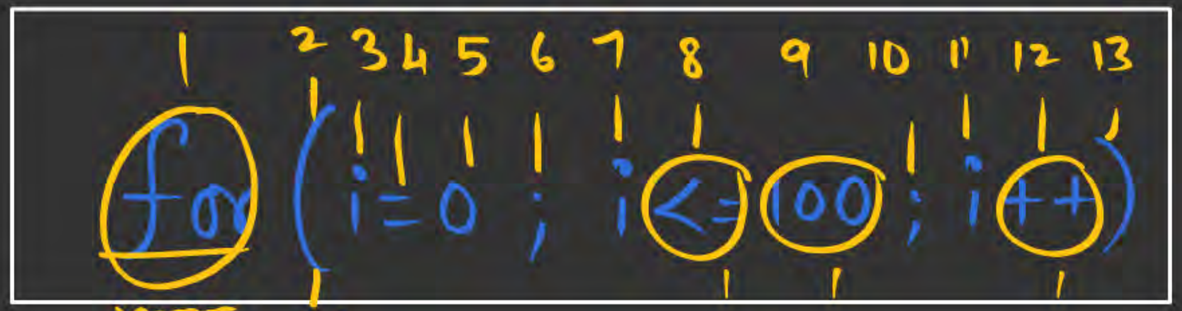
Operator

Increment/ Decrement	++ , --
Arithmetic Operator	+ - * / %
Relational Operator	< <= >= == !=
Assignment Operator	= , += , -= , *= , /= , %=
Bitwise operator	<< , >>

Operator (token)

Increment/ Decrement	++ , --
Arithmetic Operator	+ - * / %
Relational Operator	< <= >= == !=
Assignment Operator	= , += , -= , *= , /= , %=
Bitwise operator	<< , >>


1 2 3 4
① a = b ; → 4.
② a == b ; } 4
③ a + b
④ a ++ } 2
⑤ a ++ ++ ++ + } 5
⑥ a | < | b | } 3 tokens.
⑦ a <= b
⑧ a += b
⑨ a -= b
⑩ a >= b



How many tokens?

13

Topic : Transition diagrams

- Transition diagram for reserved words and identifiers

Topic : Lexical Analyzer

① int abc ; → 3

1 2 S.S

int : keyword

abc : ID

② int a , b ; → 5

1 2 3 4 5

③ abc = b + c ; → 6

1 2 3 4 5 6

Topic : Lexical Analyzer

④ if (a < b)

⑤ int a = 100, b = 200;

⑥ int abc1234float;

Topic : Lexical Analyzer

⑦ `char b = 'a';`

⑧ `char a = "abc";`

⑨ `printf("Compiler Design");`

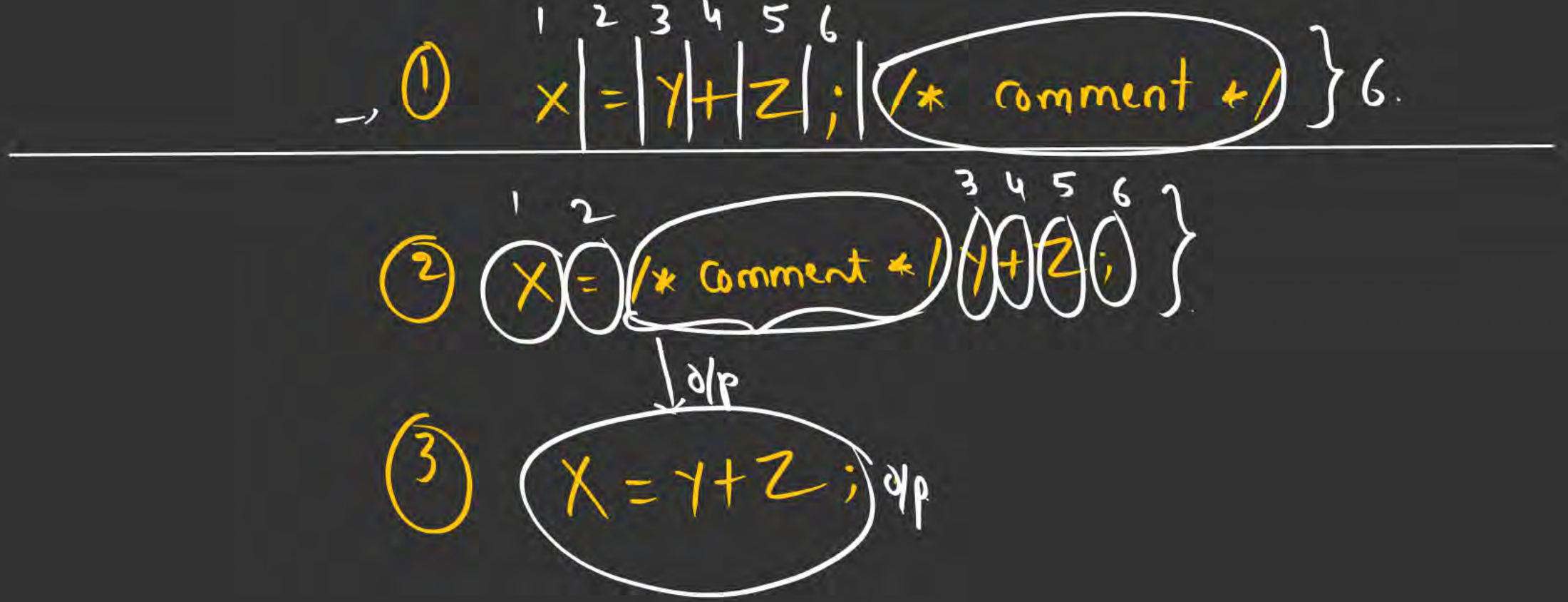
(10) `x = y++;`

(11) `x = y++++2;`

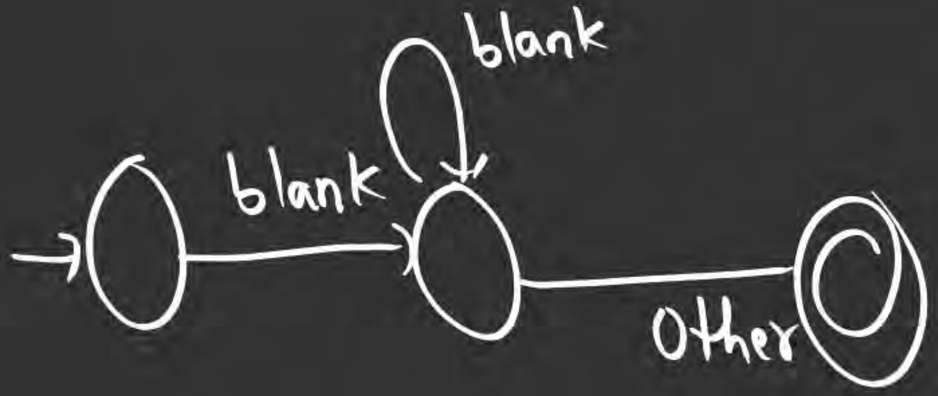
(12) `x = *y*2;`

(13) `x = y+125;`

Comment



return (no token)



return (no token)

Topic : The Role of Lexical Analyzer

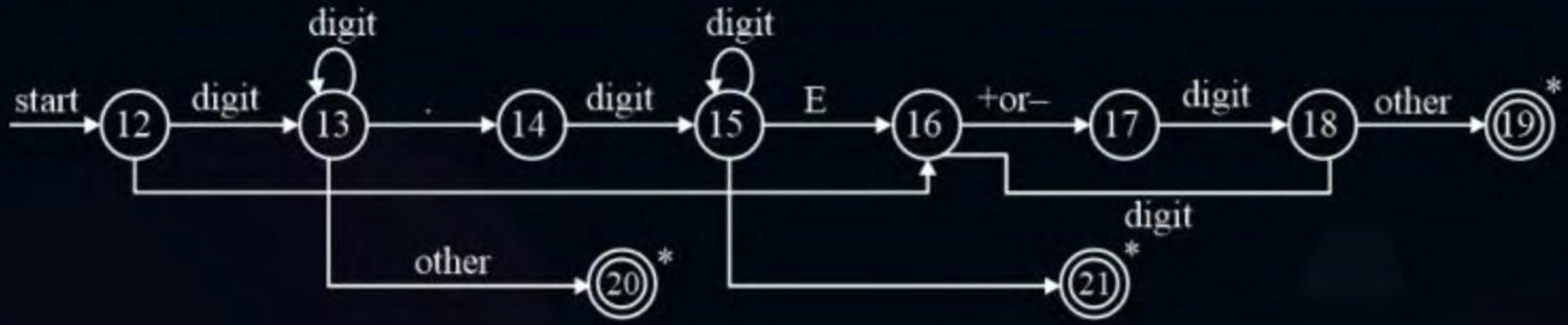
- **Lexeme:** A lexeme is a sequence of characters in the source program that is matched by the pattern for a token.

Topic : Tokens, Patterns and Lexemes

- A token is a pair a token name and an optional token value
- A pattern is a description of the form that the lexemes of a token may take
- A lexeme is a sequence of characters in the source program that matches the pattern for a token

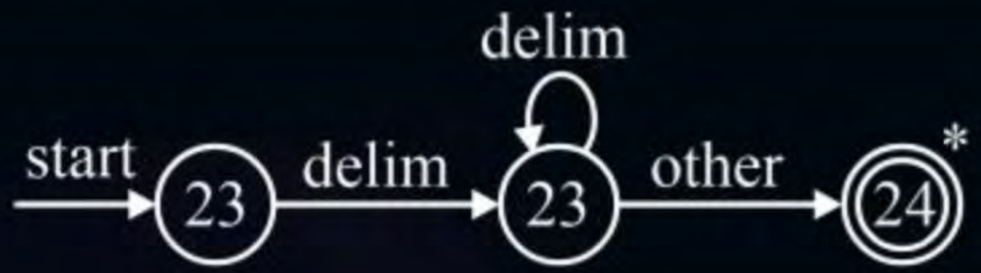
Topic : Transition diagrams (cont.)

- Transition diagram for unsigned numbers



Topic : Transition diagrams (cont.)

- Transition diagram for whitespace



#Q. How many tokens are generated by the lexical analyzer, if the following program has no lexical error?

```
main()
{
    int x, y;
    fl/*gate */oat z;
    x =/*exam*/10;
    y = 20;
}
```

```
main()
{
    int x, y;
    fl oat z;
    x = 10;
    y = 20;
}
```

(a) Lexical error

(b) No lexical error

Lexical error

1. int1 a,b; → ✓

2. int 1abc; → Lexical error
   - int → K.W
   - 1abc → Lexical error

3. char a = "Compiler" → Lexical err

4. char b = 'a' → Lexical error

- X ① K.W
- X ② I.D
- X ③ Const
- X ④ Op
- X ⑤ S.S

a = 100;

#Q. Consider the C program

```
main()
{
    int x = 10;
    x = x+y+z; /*compiler design*/
}
```

How many tokens are identified by lexical analyzer?

#Q. Consider the following program segment:

```
main ()
{
        int a, b;
                a= 5 + 8 +;
                printf("%d", a);
                /* b = 5 ; */
}
```

The number of token present in the above program segment

#Q. Consider the following expression of C program

abcd + (2 – 5 + × 6/2 − +;

How many tokens are generated by the above expression during lexical analysis?

#Q. Which of the following is not a token in C language?

A Semicolon

B Identifier

C Keyword

D White space

#Q. Consider the following program segment:

```
main ()
{
        int a, b, c;
                a = 50;
                b = &a;
        printf("%d", b);
}
```

The number of tokens in the above C code are _____.

#Q. Find the type of error produced by the following C code.

```
main()
{       in/*comment t x*/;
        floa/*comment*/t gate
}
```

A Lexical error

B Syntax error

C Both (a) and (b)

D None of these

#Q. If (z == a) function1 (10);
The number of tokens in the above statement are

A 9

B 11

C 10

D 12

Home Work

#Q. Which of the following C program statement having Lexical error.

1. iit a, b;
2. int 1 a,b;
3. int x1,b;
4. int a="hello";
5. int b = 5.89;
6. char a=a;
7. char b="hello";
8. int d="hello"
9. a=b;+c;

THANK - YOU