CS & IT ENGINEERING

compiler Design

Lexical Analysis & Syntax Analysis

Lecture No. 2

By- DEVA Sir

01 Lexical Analysis

02 Symbol Table

03 Types of Errors

04

05

Group - 1 (phase)

1. Lexical Analysis → A
2. Syntax Analysis → B
3. Semantic Analysis → C
4. Code Generator → D

Group - 2 (Input to phase)

A. character stream

B. Token stream

C. Parse Tree

D. Intermediate Code

E. Semantic Tree

F. Assembly Code

Group - 1 (phase)

1. Lexical Analysis → B
2. Syntax Analysis → C
3. Semantic Analysis → E
4. Code Generator → F

Group - 2 (output of phase)

A. character stream

B. Token stream

C. Parse Tree

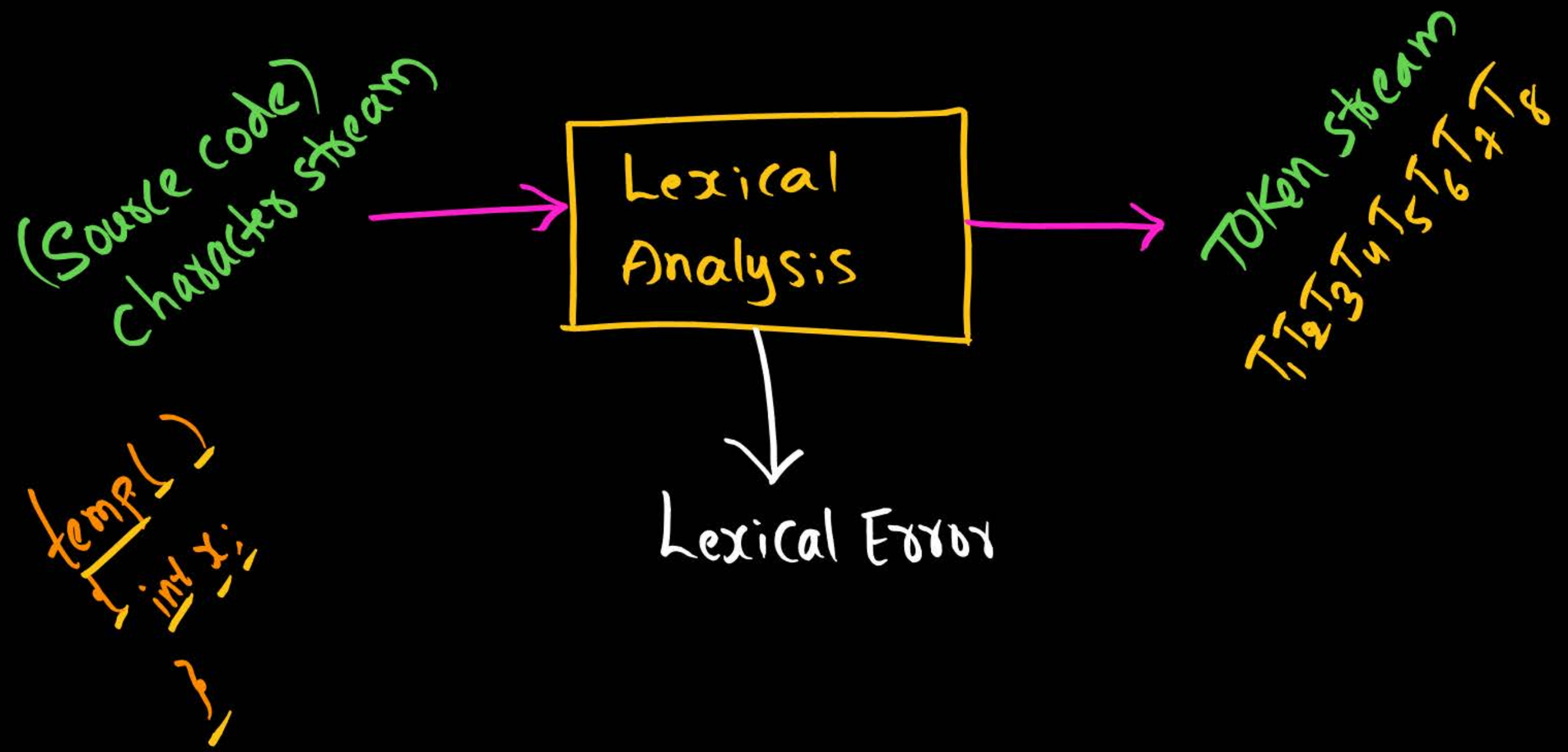
D. Intermediate Code

E. Semantic Tree

F. Assembly Code

Group-1 [Compiler]	Group-2 [TOC]
1. Lexical phase → D	A. Halting TM
2. Syntax phase → C	B. LBA
3. Semantic phase → B	C. PDA
4. Compiler → A	D. FA

Lexical Analysis :



Lexical Analysis

- Token Recognizer
- Token Generator/producer
- Scanner
- Linear phase
- Lexical phase
- First phase of compiler

Lexical Analysis

- It takes sequence of characters as i/p and produces sequence of tokens (character stream) (Token stream)
- It may produce Lexical Error if any.
- It groups one or more characters into a token
- It uses longest prefix Rule (Maximal munch Rule) to produce token
- It knows tokens, comments, white spaces
- It produces tokens but ignores white spaces & comments (deletes)

```
int x=10;
/*comment*/
main( )
{
   int y=20;
}
```

int — 1
x — 2
= — 3
10 — 4
; — 5
/* .... */ X
main — 6
(— 7
) — 8
{ — 9
int — 10
y — 11
= — 12
20 — 13
; — 14
} — 15

$T_1 T_2 T_3 T_4 T_5 T_6 \cdots T_{12} T_{13} T_{14} T_{15}$

Lexeme: Smallest unit of program

TOKen: Type of Lexeme

Lexeme ≅ TOKen

Types of Tokens :

1) Identifiers
   i) can't start with digit
   ii) includes only letters, digits, underscore
   iii) should not include any other character
   iv) Keyword never be identifier

2) Keywords
   32 Keywords

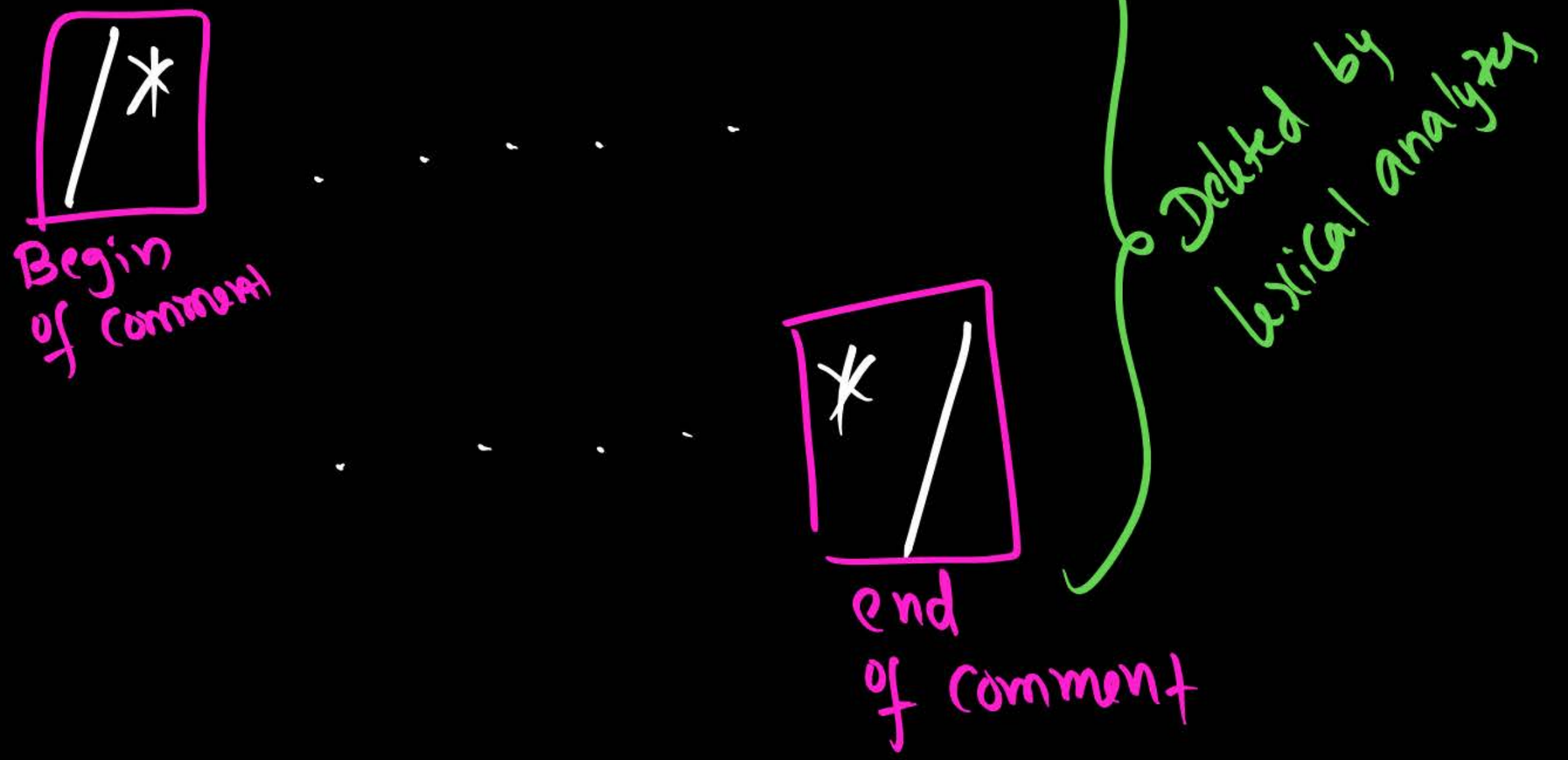
int	short	unsigned	if	goto	default case	struct	const
float	long	signed	else	while	for	union	volatile
char	void	typedef	switch	break	continue	enum	sizeof( )
double			do	return		register, auto, extern, static	

3) Operators
   +, -, *, /, %, ++, --, <, <=, >, >=, ==, !=,
   &&, ||, !, &, |, ^, ~, <<, >>, =, +=, -=, ...

4) Literals (constants)
   - Integer
     - Decimal 235
     - octal 0126
     - hexa 0x1a2f
   - Floating
     - 20.34
     - 20.34f
   - character
     - 'A'
     - '\n'
   - string
     - "abc"

5) Special symbols
   ; , (,) , [,] , { , } , :

Comments :


/*
Begin of comment
*/
end of comment
Deleted by lexical analyzer

Lexical Error :

① 1abc → not a token ⇒ Lexical Error (Invalid Sequence)

a1bc → Identifier → Token

1234 → Integer literal → Token

Note : Space always separates a token

② 1 abc ⇒ 2 tokens

1 : Integer

abc : Identifier

(3)

Lexical Error

- 0293 → lexical error (0 → octal 0 to 7)
- 0x2f → token (0x → hexadecimal)
- 0x2h → lexical error
- 0135 → token

④ /* comment

Begin of Comment

is started /*

There is no end rule for comment

⇒ Lexical error

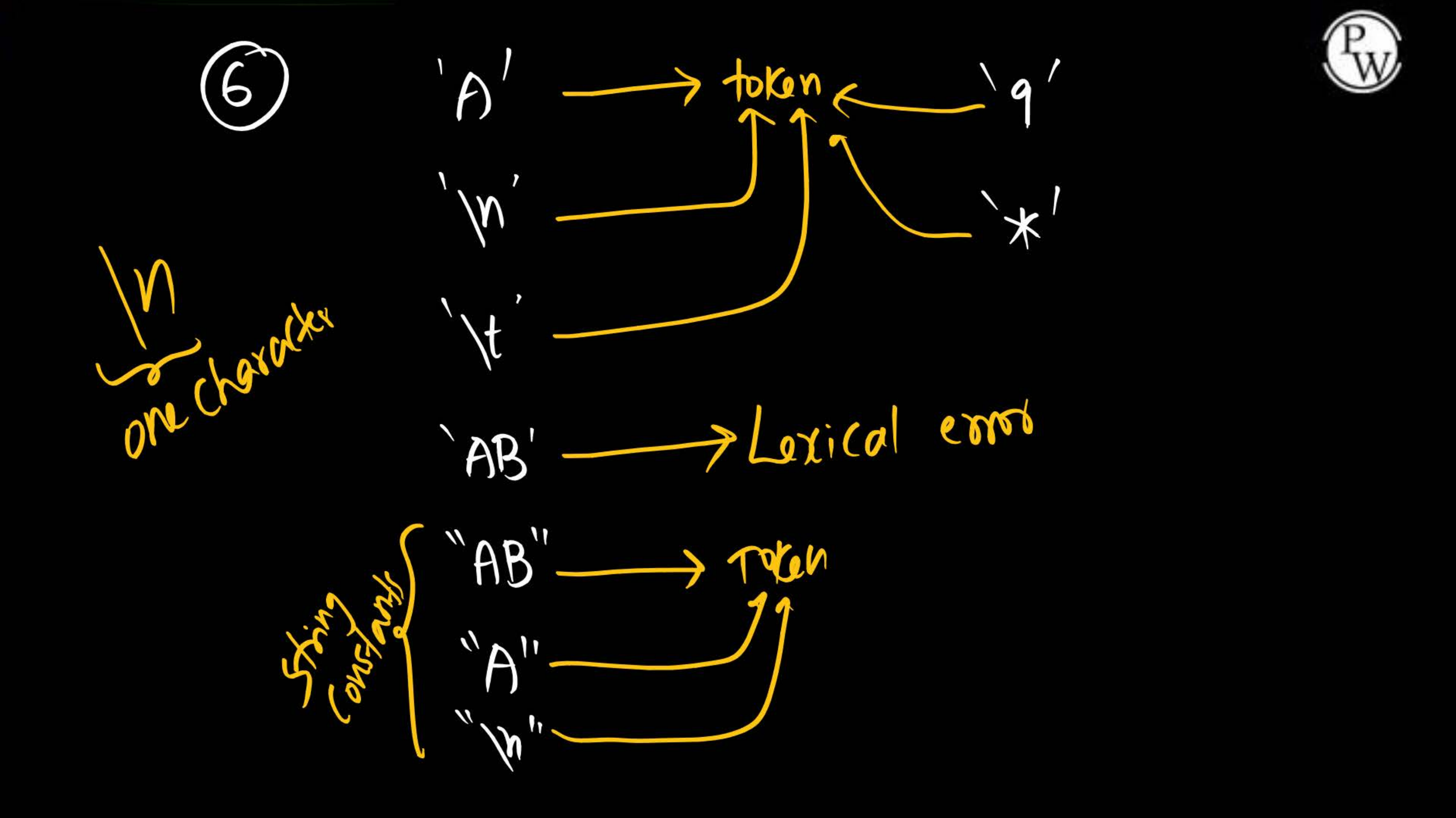
⑤ /* comment

Start

/* is started */

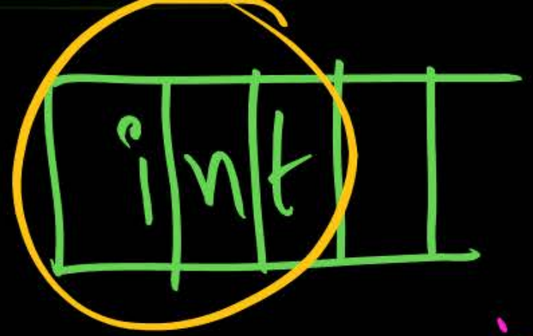
end

There is no lexical error


6
'A'
'\n'
'\t'
token
'q'
'*'
\n
one character
'AB'
Lexical error
"AB"
"A"
"\n"
String Constants
Token

Lexical Analysis

- Regular Expression
- FA (DFA, NFA)



i ✓

in ✓

int ✓ longest prefix

Finding no. of tokens in the code.

① T_1 T_2 T_3

int	x	;

⇒ 3 tokens

---

② | int | temp | ; | ⇒ 3

---

③ int x , y ; ⇒ 5

Finding no. of tokens in the code.

④
```
main( )
{
  main( ) ;
}
```
⇒ 9

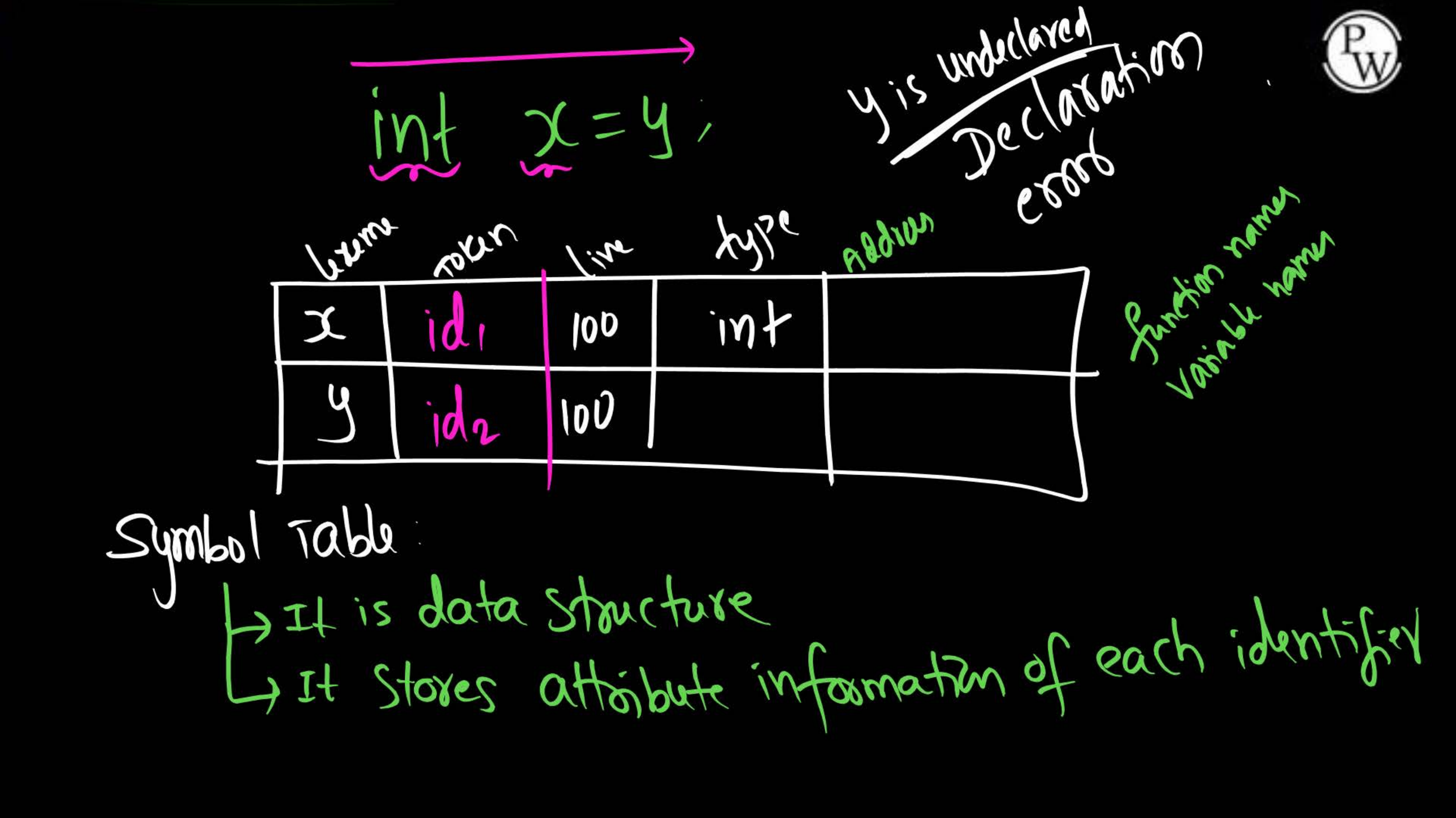
No compilation Error
Runtime error
(stack overflow)

⑤
```
int = float ;
```
⇒ 4

No lexical Error
Syntax Error
Compilation Error

⑥
```
int x = y ;
```
⇒ 5

No lexical error
No Syntax error
Semantic error (y is not declared)
Compilation error


int x = y;
y is undeclared
Declaration error
lexeme
token
line
type
addres
x
id1
100
int
y
id2
100
function names
Variable names
Symbol Table:
It is data structure
It stores attribute information of each identifier

Depends on scope, Symbol tables will be maintained.

Finding no. of tokens in the code.

⑦ `x++;` ⇒ 3 tokens

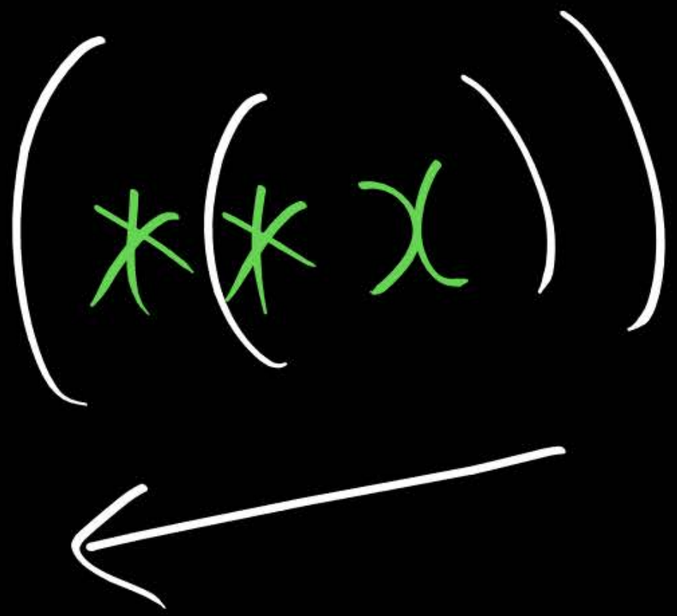
No lexical error
No syntax error
Semantic error (x is not declared)

⑧ `int *x;` ⇒ 4 tokens

no compilation error

⑨ `int **x;` ⇒ 5 tokens

no error



** is not operator in C

* → dereference

Finding no. of tokens in the code.

(10)

```
printf("x=%d", x);
```

⇒ 7 tokens

Semantic Error

(11)

```
int x;
2 = x;
```

⇒ 7 tokens

Semantic Error (lvalue required)

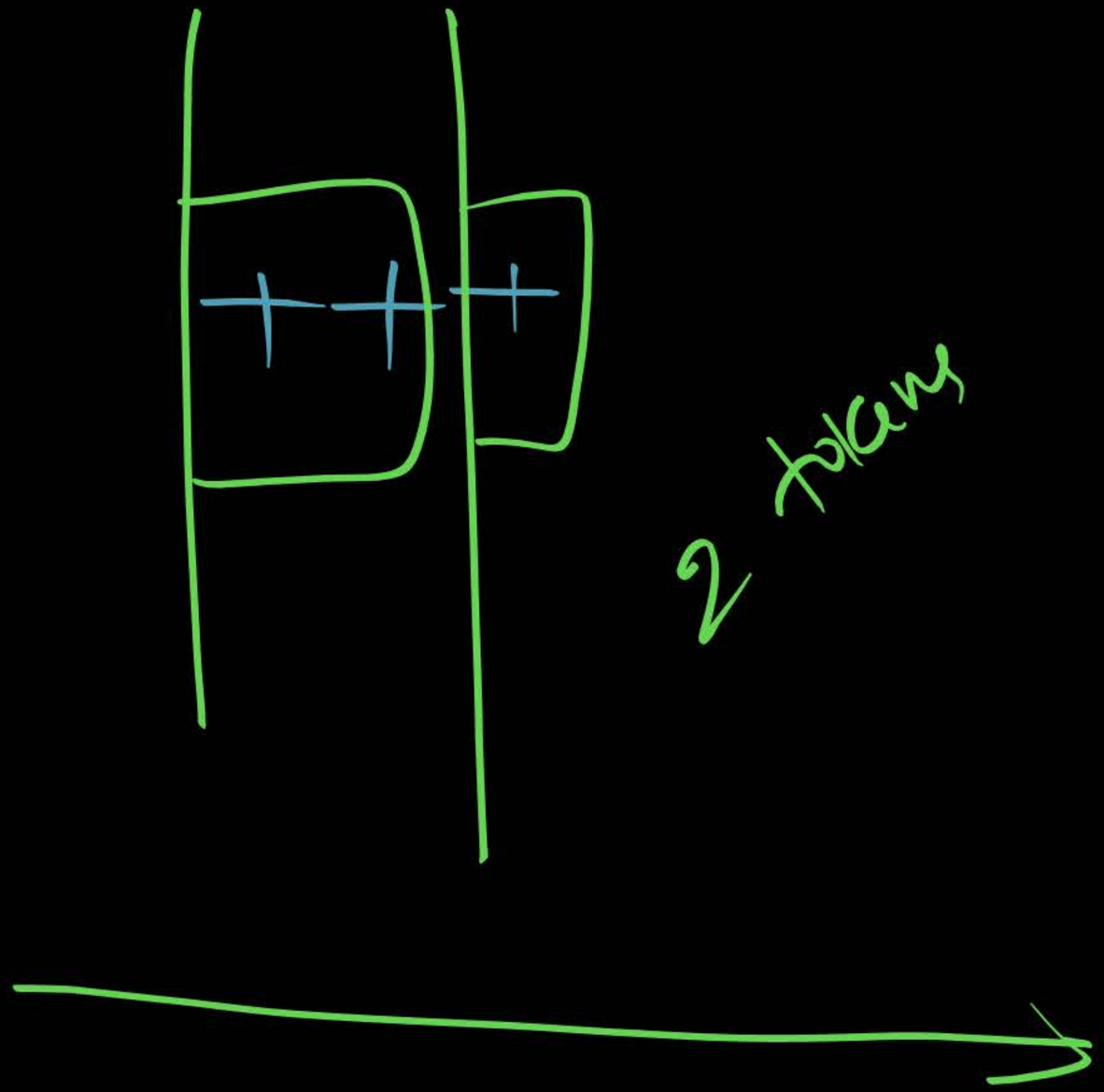
compilation error

(12)

```
int x;
x = x+++2;
```

⇒ 10 tokens

no compilation error


2 dalam

Finding no. of tokens in the code.

(13) `int x, y, z=10;`

⇨ 9

no error

---

(14) `int x=y=z=10;`

⇨ 9

Semantic error

---

(15) `int x=1, y=2;`

`x = (1, 2, 3);`

⇨ 19

no error

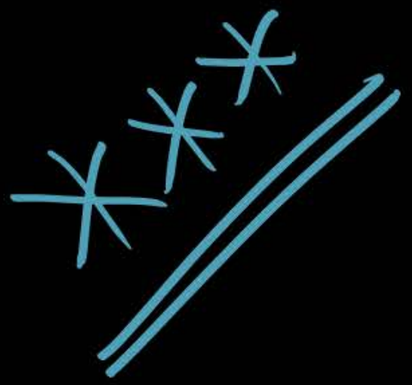
Finding no. of tokens in the code.

(16) /* comment
int x = 10 /* comment */ ⇒ 0 tokens
No error

(17) in/*comment*/t = 10; ⇒ in t = 10; ⇒ 5
Semantic error

(18) int x;
char y;
y = x;
⇒ 10 tokens

int size 2 or 4
char size 1
Semantic error



```
in t = 10;
```

~~Syntax Error~~

Semantic Error

Note:

```
typedef int in;
in t = 10;
```

no compilation error

Finding no. of tokens in the code.

(19)

```
int x=1;
char y;
y=(char)x;
```

⇒ 15 tokens

No error

---

(20)

```
char *p="gate";
```

⇓

6 tokens

Errors

Compile time Errors

Preprocessor Errors

```
#inlude<stdio.h>
```

not preprocessor command

Compilation Errors

- Lexical
- Syntax
- Semantic

Runtime Errors

- File not found
- Divide by Zero
- out of bound access in array
- Segmentation fault
- pointer related errors
- stack overflow
- Infinite loop

Question:

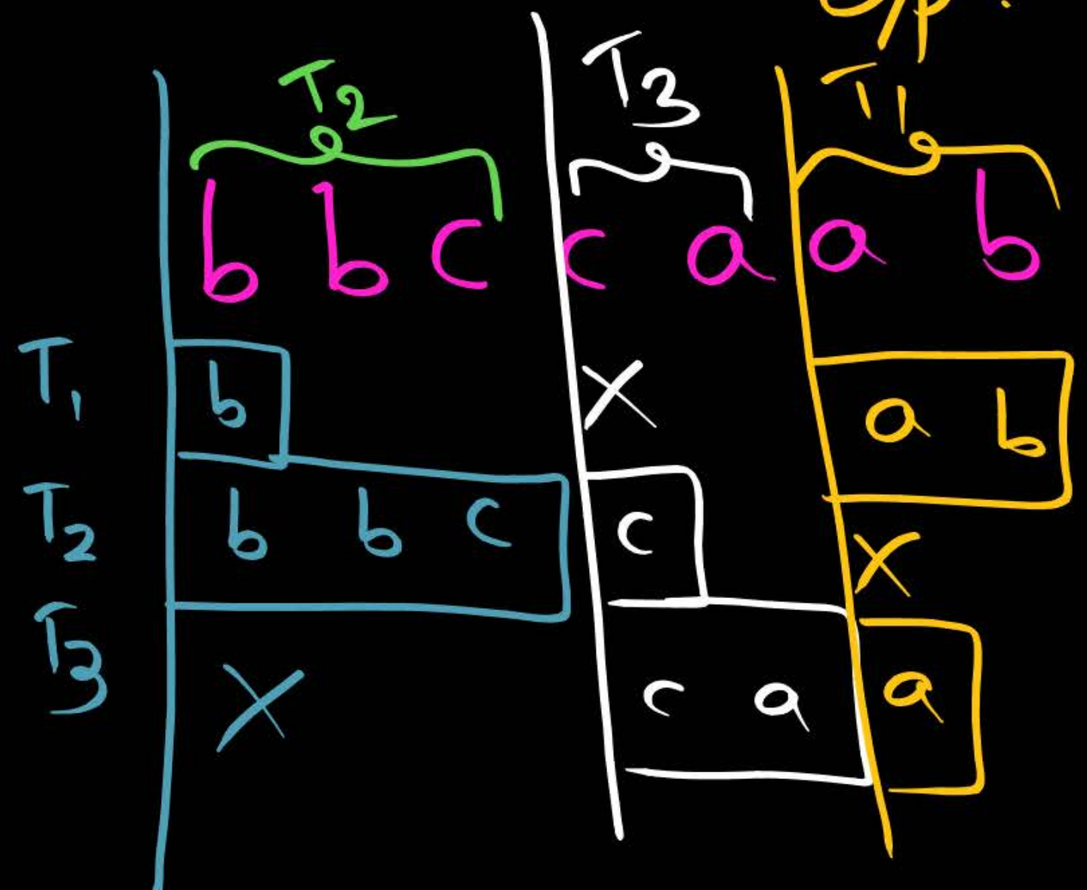
$T_1 : a^*b$

$T_2 : b^*c$

$T_3 : c^*a$

I/p : bbccaab

O/p : $T_2 T_3 T_1$



Summary

- Lexical Analysis

Next : Syntax Analysis


THANK YOU
GW
SOLDIERS !