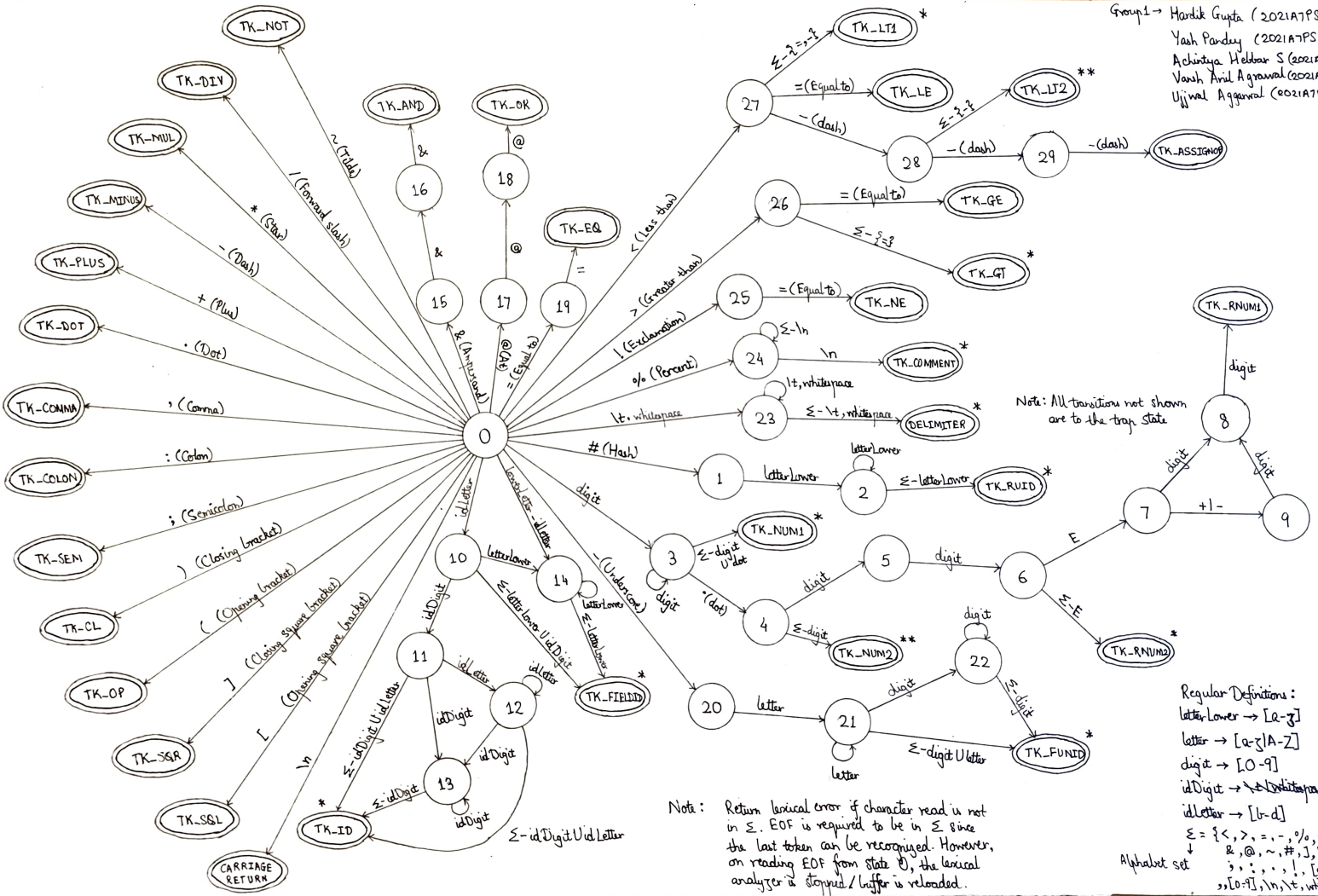
Group1 → Hardik Gupta (2021A7PS2421P)
Yash Pandey (2021A7PS0661P)
Achintya Hebbar S (2021A7PS1457P)
Vansh Anil Agrawal (2021A7PS2215P)
Ujjwal Aggarwal (2021A7PS2427P)
TK_NOT
~ (Tilde)
TK_DIV
/ (Forward slash)
TK_MUL
* (Star)
TK_MINUS
- (Dash)
TK_PLUS
+ (Plus)
TK_DOT
. (Dot)
TK_COMMA
, (Comma)
TK_COLON
: (Colon)
TK_SEM
; (Semicolon)
TK_CL
) (Closing bracket)
TK_OP
((Opening bracket)
TK_SQR
] (Closing square bracket)
TK_SQL
[(Opening square bracket)
CARRIAGE RETURN
\n
TK_AND
&
TK_OR
@
TK_EQ
=
TK_LT1
TK_LE
TK_LT2
TK_ASSIGNOP
TK_GE
TK_GT
TK_NE
TK_COMMENT
TK_DELIMITER
TK_RUID
TK_NUM1
TK_NUM2
TK_RNUM1
TK_RNUM2
TK_FIELDID
TK_FUNID
TK_ID
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
& (Ampersand)
@ (@@)
= (Equal to)
< (Less than)
> (Greater than)
! (Exclamation)
% (Percent)
\t, whitespace
(Hash)
letterLower
digit
idLetter
idDigit
_ (Underscore)
letter
. (dot)
E
+|-
Σ - letterLower
Σ - digit
Σ - idDigit U letter
Σ - letter
Σ - \n
Σ - \t, whitespace
Σ - digit U letter
Σ - idDigit U idLetter
Note: All transitions not shown are to the trap state
Note: Return lexical error if character read is not in Σ. EOF is required to be in Σ since the last token can be recognized. However, on reading EOF from state 0, the lexical analyzer is stopped / buffer is reloaded.
Regular Definitions:
letterLower → [a-z]
letter → [a-z|A-Z]
digit → [0-9]
idDigit → [2-7]
idLetter → [b-d]
Alphabet set Σ = {<, >, =, -, %, +, *, /, &, @, ~, #,], [,), (, ;, :, ., !, [a-z], [A-Z], [0-9], \n, \t, whitespace, EOF}

Group No: 1

- Hardik Gupta 2021A7PS2421P
- Yash Pandey 2021A7PS0661P
- Achinthya Hebbar S 2021A7PS1457P
- Ujjwal Aggarwal 2021A7PS2427P
- Vansh Anil Agrawal 2021A7PS2215P

STATE	ACTIONS
TK_NUM1	1) Compute equivalent integer number. 2) Return (TK_NUM, number, lineNo)
TK_NUM2	1) Decrement forward pointer by 1. 2) Compute Equivalent integer number. 3) Return (TK_NUM, Number, lineNo)
TK_RNUM1	1) Increment forward pointer. 2) Compute Equivalent real number. 3) Return (TK_RNUM, Number, lineNo).
TK_RNUM2	1) Compute Equivalent real Number. 2) Return (TK_RNUM, Number, lineNo)
TK_FIELDID	1) Lookup lexeme in symbol table. 2) Tokenize as keyword if present else as TK_FIELDID. 3) Insert entry in symbol table if not keyword. 4) Return pointer to corresponding entry in symbol table return (getToken (TK_FIELDID, installID(), lineNo)
TK_RUID	1) Lookup lexeme in symbol table. 2) Insert entry in symbol table if not present. 3) Return pointer to corresponding entry in symbol table return (getToken (TK_RUID, installID(), lineNo)
TK_FUNID	1) Check size of lexeme. If greater than 30, report lexical error. 2) Lookup in symbol table and tokenize as keyword if present else as TK_FUNID. 3) Insert a new entry if not present. 4) Return token & pointer to corresponding found/inserted entry in symbol Table. return (getToken (TK_FUNID), installID(lexeme), lineNo).
TK_ID	1) Check size of lexeme. If greater than 20, report lexical error. 2) Lookup in symbol table. 3) Insert a new entry if not present. 4) Return token & pointer to corresponding found/inserted entry in symbol Table. • return (TK_ID, installID(lexeme), lineNo).
TK_LE, TK_LT1, TK_NE, TK_EQ, TK_GE, TK_MUL, TK_DIV, TK_AND, TK_OR, TK_SQR, TK_SQL, TK_OP, TK_CL, TK_NOT, TK_MINUS, TK_ASSIGNOP, TK_PLUS, TK_COMMA, TK_DOT, TK_COLON TK_SEM	1) Increment forward pointer by 1. 2) Return (TOKEN, lexeme, lineNo)
TK_LT2	1) Decrement forward pointer by 1. 2) Return (TK_LT, "<", lineNo)
NEW LINE	1) Increment LineNo. 2) Increment forward pointer. 3) Make begin pointer = forward pointer. 4) Return to start state
DELIMITER	1) Make begin pointer = forward pointer. 2) Return to start state.
TK_COMMENT	1) Make begin pointer = forward pointer 2) Return to start state

NOTE:
- lexeme returned for a token consists of the string from begin to forward-1 (inclusive)
- After returning (token, lexeme, lineNo); the state is set back to start(0) & begin = forward.