**Name:** Syed Asad Abrar

**Roll No:** L16-4292

**Compiler Construction Assignment 3**

**Description of language:**

Data types: (INT, ^), (CHAR, ^)

Keywords: (IF, ^), (ELSE, ^), (WHILE, ^), (RET, ^), (‘jIn’, ^), (‘jOut’, ^)

Arithmetic operators: (‘+’, ^), (‘-’, ^), (‘\*’, ^), (‘/’, ^)

Relational operators (RO): (RO, LT), (RO, LE), (RO, GT), (RO, GTE), (RO, EQ), (RO, NE)

Comments: (‘/\*’, ^), (‘\*/’, ^)

Identifier: (ID, x) where x is any letter followed by any number of letters or digits

Numeric constants: (NC, 1), (NC, 2) … (NC, 9), (NC, 0)

Literal constants: (LC, “‘a’”) … (LC, “‘z’”), (LC, “‘A’”) … (LC, “‘Z’”)

String: (STR, x) where x is a string

Other tokens: (‘(’, ^), (‘)’, ^), (‘{’, ^), (‘}’, ^), (‘[’, ^), (‘]’, ^)

Assignment operator: (‘<-’, ^)

Semi-colon: (‘;’, ^)

Colon: (‘:’, ^)

Comma: (‘,’, ^)

**Token-lexeme pairs:**

1. (INT, ^), (ID, “numPrint”), (‘(’, ^), (INT, ^), (ID, “num”), (‘,’, ^), (INT, ^), (ID, “length”), (‘)’, ^)
2. (‘{’, ^)
3. (INT, ^), (ID, “i”), (‘,’, ^), (ID, “j”), (‘,’, ^), (ID, “first”), (‘,’, ^),(ID, “temp”), (‘;’, ^)
4. (CHAR,^), (ID, “a”), (‘;’, ^)
5. (ID, “a”), (‘<-’, ^), (STR, “‘x’”), (‘;’, ^)
6. (‘jOut’, ^), (‘(’, ^), (STR, “enter number”), (‘)’, ^), (‘;’, ^)
7. (‘jIn’, ^), (‘(’, ^), (ID, “i”), (‘)’, ^), (‘;’, ^)
8. (‘jOut’, ^), (‘(’, ^), (ID, “i”), (‘)’, ^), (‘;’, ^)
9. (ID, “i”), (‘<-’, ^), (ID, “length”), (‘;’, ^)
10. (WHILE, ^), (‘(’, ^), (ID, “i”), (RO, GT), (NC, 0), (‘)’, ^)
11. (‘{’, ^)
12. (ID, “first”), (‘<-’, ^), (NC, 0), (‘;’, ^), (‘/\*’, ^), (STR, “this line contains a comment”), (‘\*/’, ^)
13. (ID, “j”), (‘<-’, ^), (NC, 1), (‘;’, ^)
14. (WHILE, ^), (‘(’, ^), (ID, “j”), (RO, LT), (ID, “i”), (‘)’, ^)
15. (‘{’, ^)
16. (‘jOut’, ^), (‘(’, ^), (ID, “j”), (‘)’, ^), (‘;’, ^)
17. (ID, “j”), (‘<-’, ^), (ID, “j”), (‘+’, ^), (NC, 1), (‘;’, ^)
18. (‘}’, ^)
19. (‘/\*’, ^), (STR, “this is a comment”), (‘\*/’, ^)
20. (ID, “i”), (‘<-’, ^), (ID, “i”), (‘-’, ^), (NC, 1), (‘;’, ^)
21. (‘/\*’, ^), (STR, “This is a\n”)
22. (STR, “Multiline”)
23. (STR, “Comment”), (‘\*/’, ^)
24. (‘}’, ^)
25. (‘jOut’, ^), (‘(’, ^), (STR, “temp is ”), (‘)’, ^), (‘;’, ^)
26. (‘jOut’, ^), (‘(’, ^), (ID, “temp”), (‘)’, ^), (‘;’, ^)
27. (RET, ^), (ID, “i”), (‘;’, ^)
28. (‘}’, ^)

**Regular definitions:**

1. RO -> LT | GT | LE | GTE | EQ | NE
2. ID -> letter(letter | digit)\*

letter -> A | B | … | Z | a | b | … | z

digit -> 0 | 1 | 2 | … | 9

1. NC -> (sign)digit(digit)\*

sign -> + | - | ^

digit - > 0 | 1 | 2 | … | 9

1. LC -> ‘letter’

letter -> A | B | … | Z | a | b | … | z

1. STR -> x

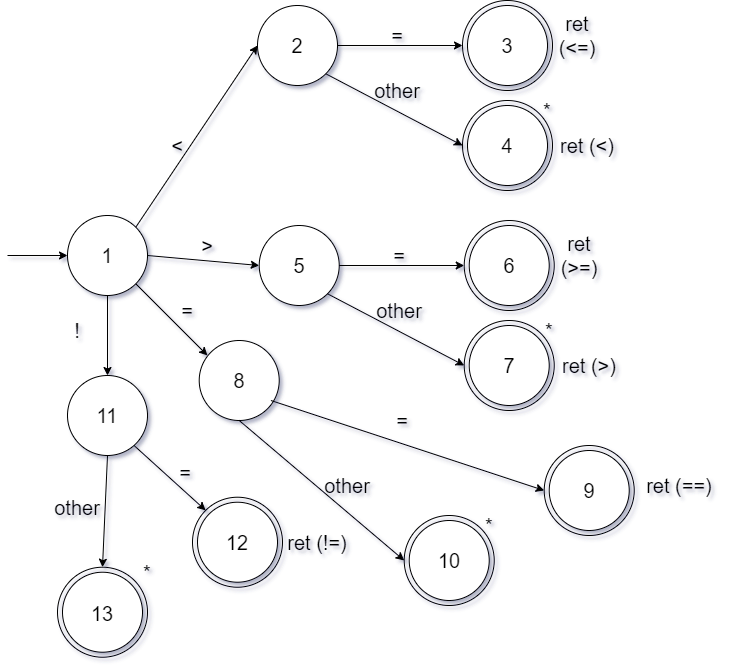
x -> letter(x) | digit(x) | space(x) | ^

letter -> A | B | … | Z | a | b | … | z

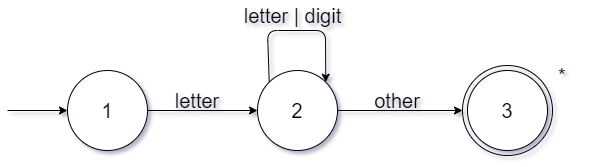
digit -> 0 | 1 | 2 | … | 9

space -> ‘ ’

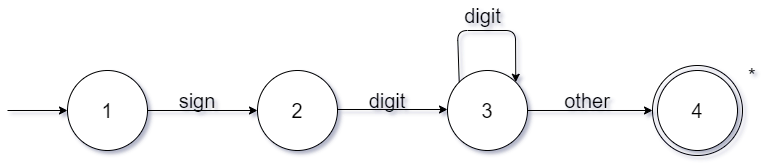
**Transition Diagrams:**

****Relational Operators (RO):

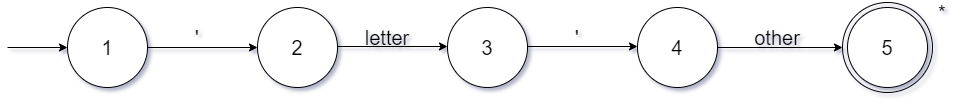
Identifier:



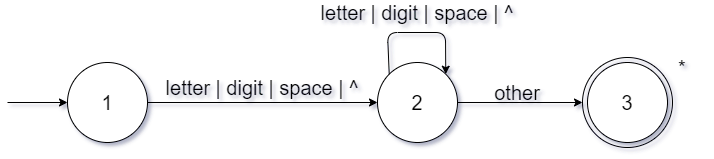
Numerical Constant:



Literal Constant:



String:



**Complete grammar:**

P -> D P | ^

D -> T ID R

T -> int | char

R -> AV ; SL | (NPL) {SL}

NPL -> PL | ^

PL -> T ID opt

opt -> , PL | ^

SL -> S SL | ^

S -> T ID AV ; | ID <- VAL ; | jOut(“STR”); | jIn(ID); | jOut(VAL); | while(REXP) {SL} | return VAL; | if (REXP) {SL} IE

AV -> , ID AV | ^

VAL -> LC | EXP

EXP -> VAL2 EXP2

EXP2 -> + VAL2 EXP2 | - VAL2 EXP2 | ^

VAL2 -> VAL4 VAL3

VAL3 -> \* VAL4 VAL3 | / VAL4 VAL3 | ^

VAL4 -> ID | NC | ( EXP )

REXP -> ID RO ID | ID RO NC | NC RO NC | NC RO ID

IE -> else optC {SL} IE | ^

optC -> if(REXP) | ^

P = program

D = definition

T = type

PL = parameter list

NPL = null parameters list (to deal with no parameters)

opt = optional parameters

SL = statement list

AV = additional variables

VAL = value

LC = literal constant

EXP = expression

AO = arithmetic operator

RO = relational operator

NC = numerical constant

ID = identifier

REXP = relational expression

IE = if extension

optC = optional condition