CS 419 Compiler

Project Form

Project Idea:

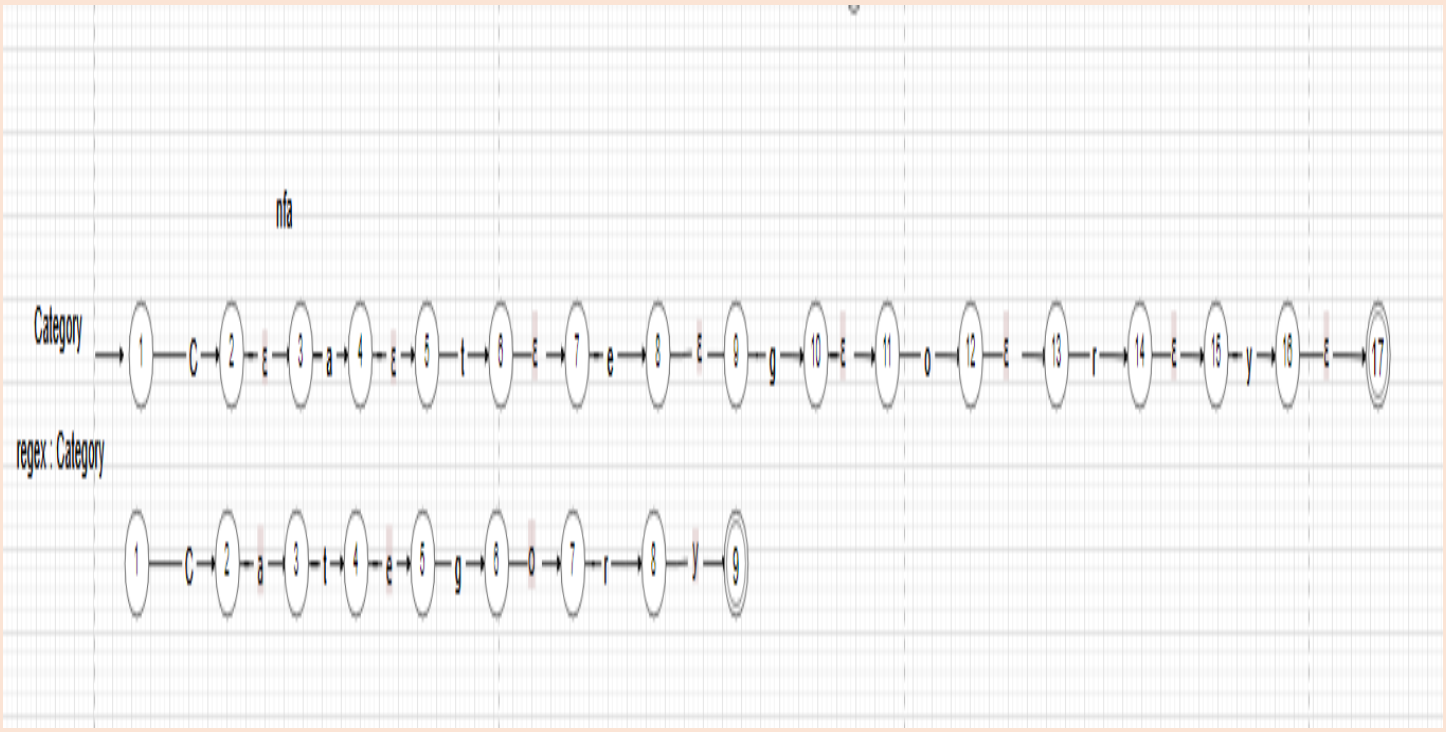
Project Idea #2

Team Members NO#: 7

ID	Name	Level& Department	Section(Day-from-to)	Role (Lead/Member)	Grade
201900754	محمود احمد صلاح علي	3		leader	
201900728	محمد محمود الدمرداش لاشين	3			
20180441	كيرلس ميشيل سعيد عطاالله	4			
20150644	يوسف محمود محمد رجب	4			
20180436	كريم محمد يوسف شطا	4			
20160067	اسلام سيد صلاح	4			
20170409	محمد احمد عياد	4			

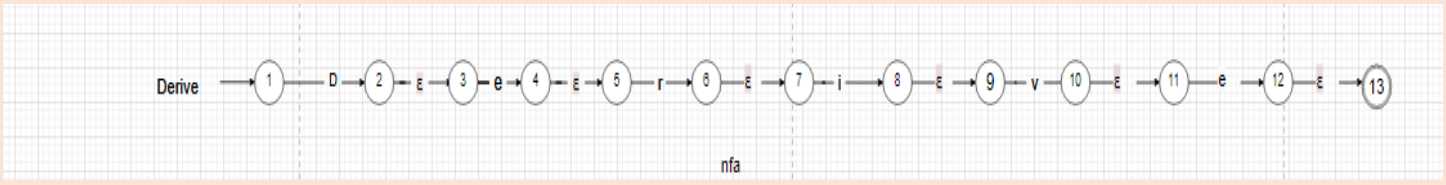
Regular Expression, Finite automata and Conversion from RegX to NFA, NFA to DFA

1-Category:

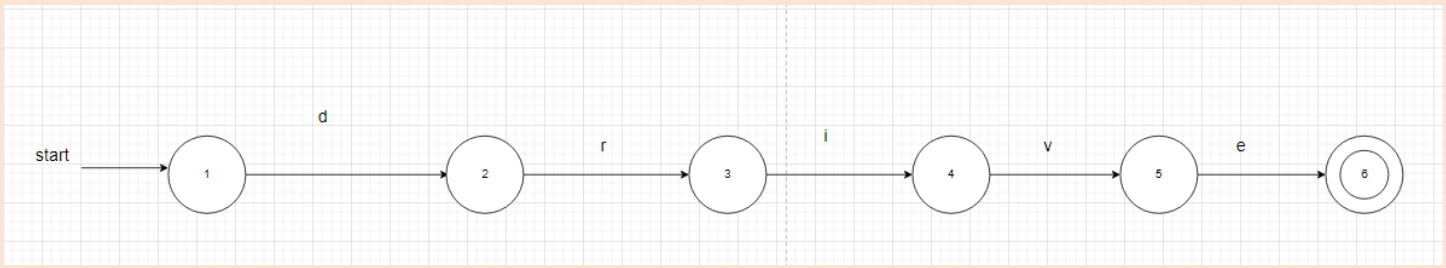


2-Derive:

N f a :

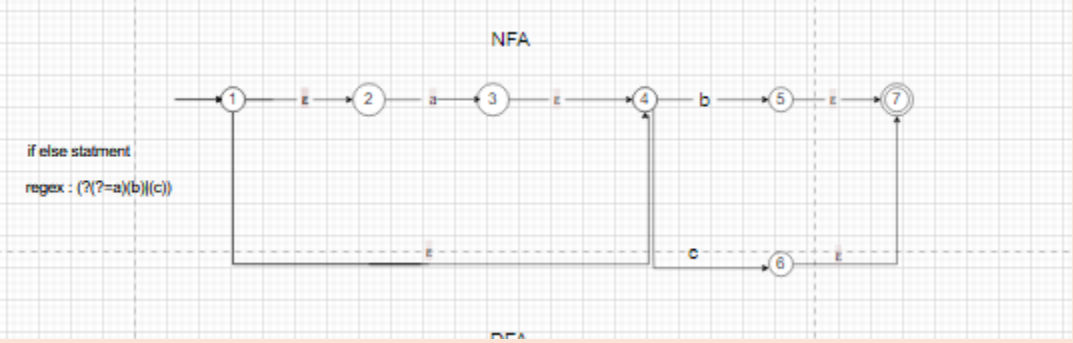


D f a :

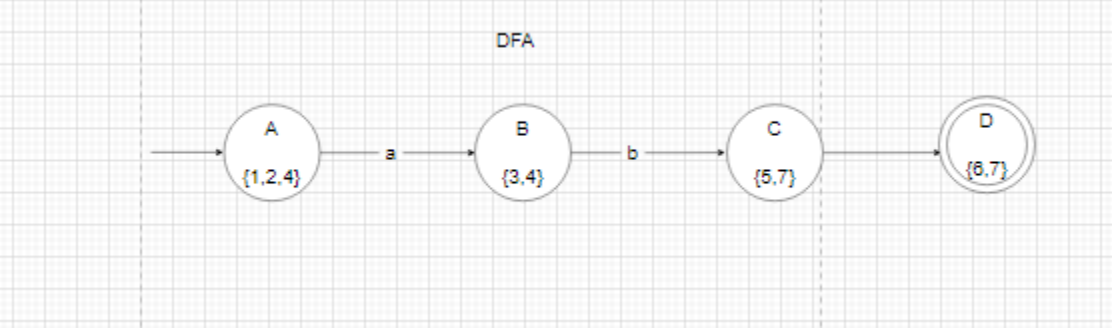


3-Else if:

N f a :

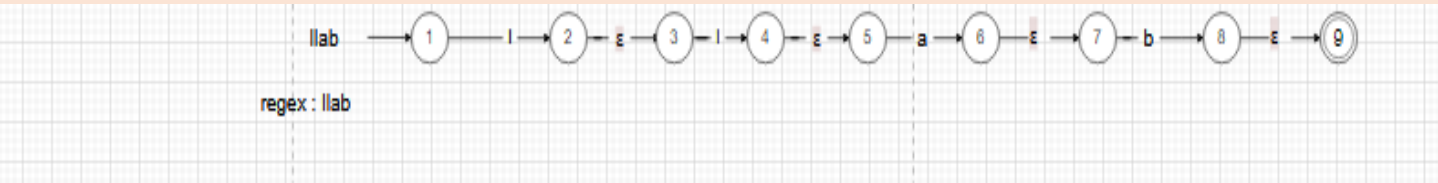


D f a :

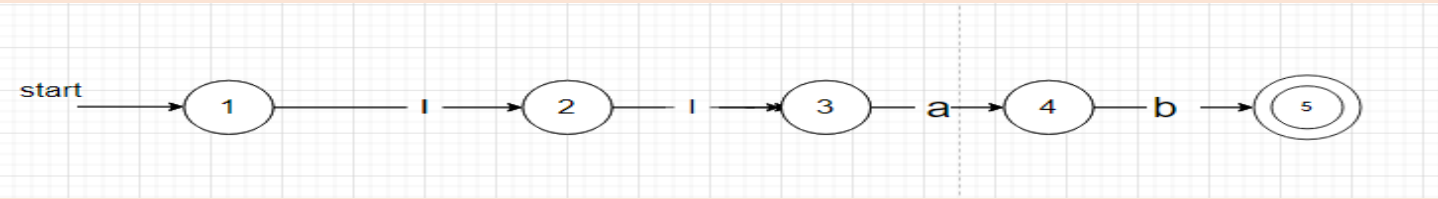


4-Ilap:

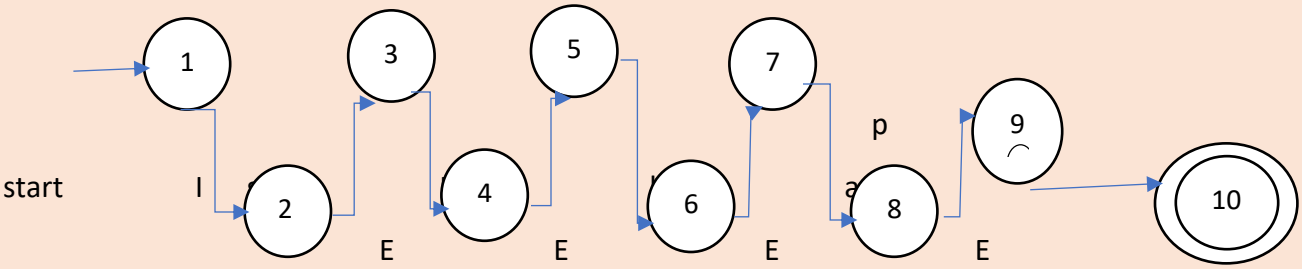
N f a :



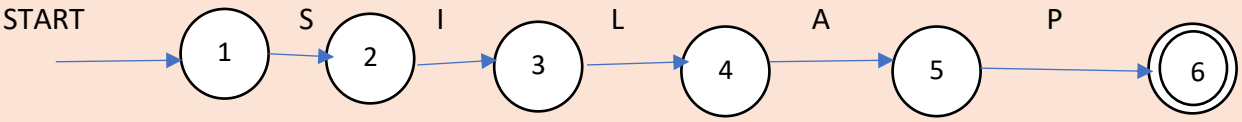
D f a :



5-Silap:



D F A:

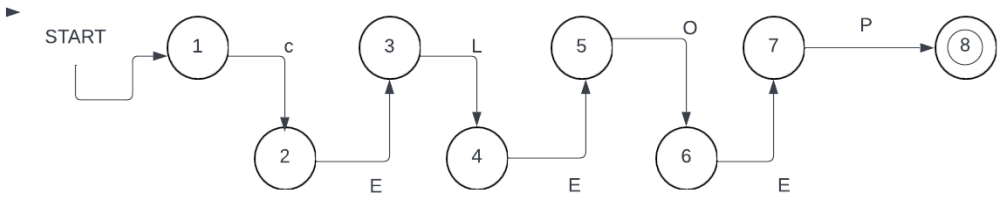


TRANSACTION TABLE:

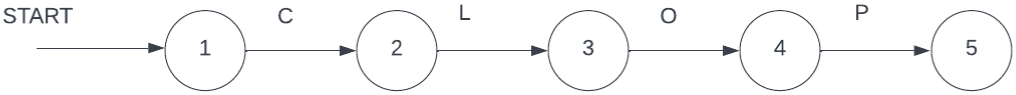
	S	I	L	A	P	
1	2					
2		3				
3			4			
4				5		
5					6	
6						

6-CLOP:

N F A:



D F A:

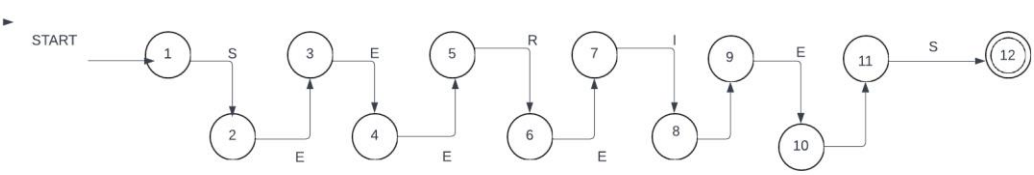


TRANZACTION TABLE:

	C	L	O	P	
1	2				
2		3			
3			4		
4				5	
5					

7-SERIES:

NFA :



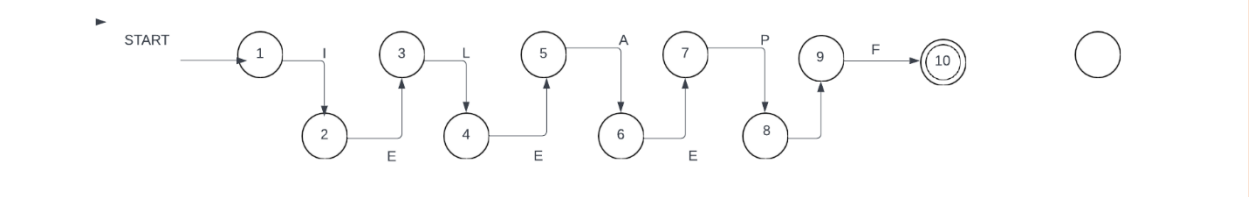
DFA:

TRANSACTION TABLE:

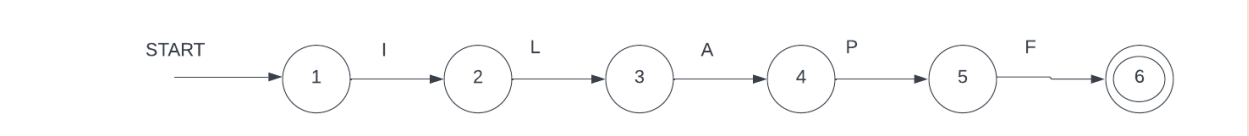
	S	E	R	I	E	S
1	2					
2		3				
3			4			
4				5		
5					6	
6						7
7						

8-ILAPF:

N F A :



D F A:

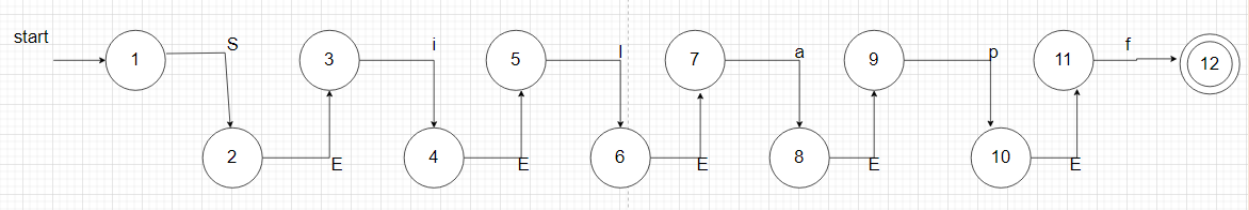


TRANZACTION TABLE:

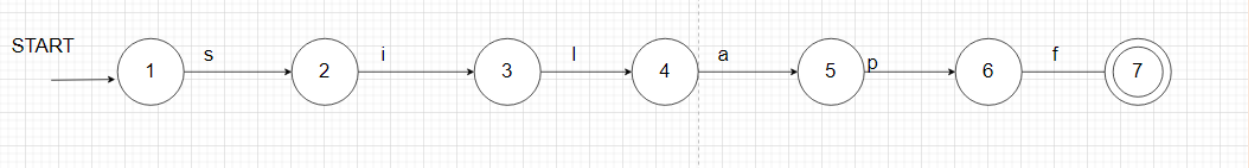
	I	L	A	P	F	
1	2					
2		3				
3			4			
4				5		
5					6	
6						

9-Silapf:

N F A :



D f a:

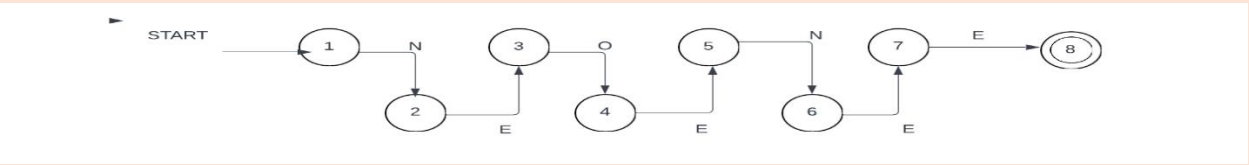


TRANSACTION TABLE :

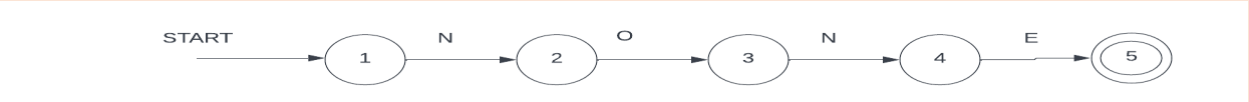
	S	I	L	A	P	F
1	2					
2		3				
3			4			
4				5		
5					6	
6						7
7						
8						

10-NONE:

N F A



D F A:

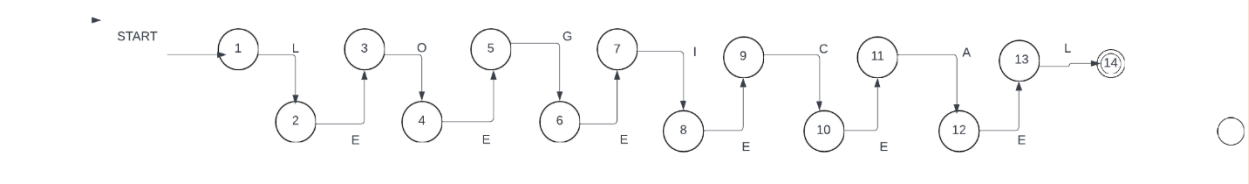


TRANZACTION TABLE :

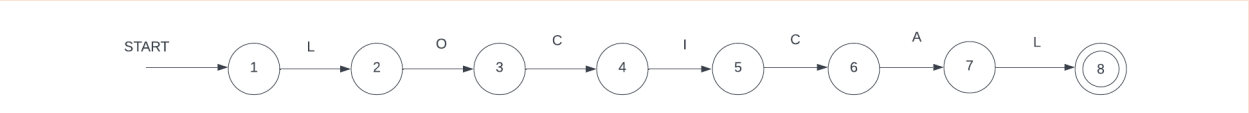
	N	O	E	
1	2			
2		3		
3	4			
4			5	
5				

11-LOGICAL:

N F A:



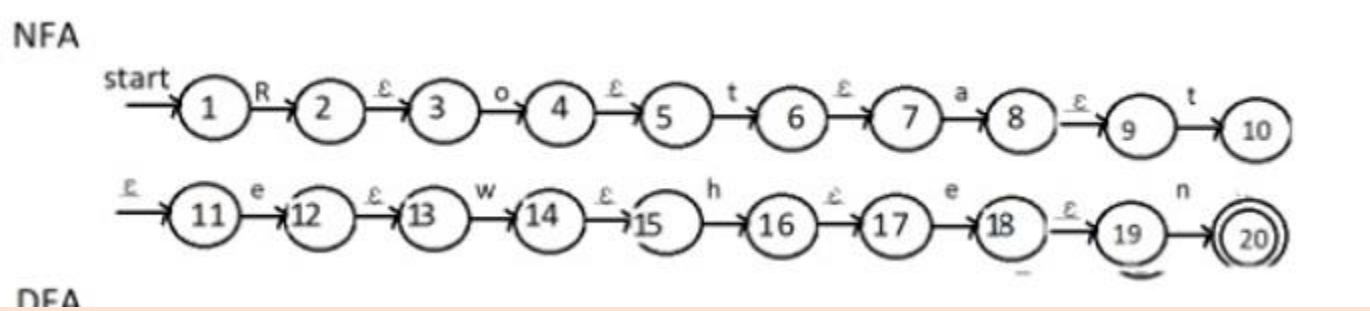
D F A:



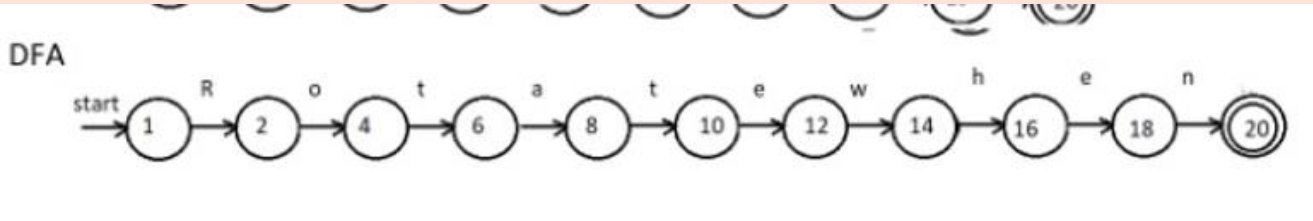
TRANSACTION TABLE :

	L	O	G	I	C	A
1	2					
2		3				
3			4			
4				5		
5					6	
6						7
7	8					
8						

12-Rotatewhen:



D f a :

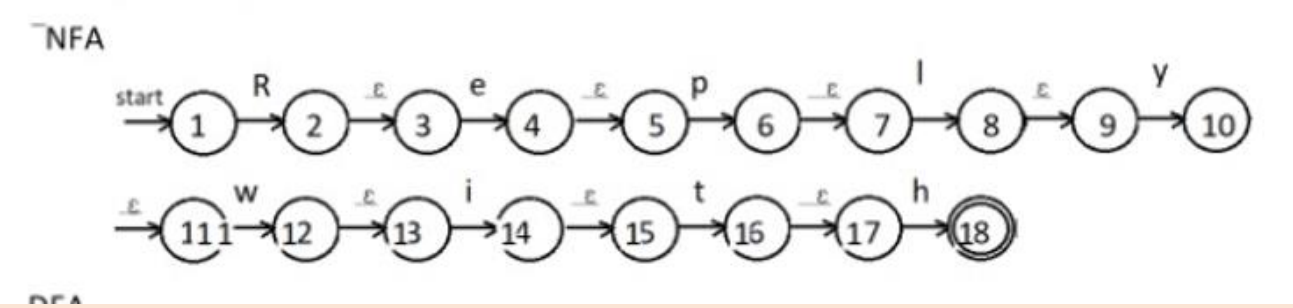


TRANSACTION TABLE :

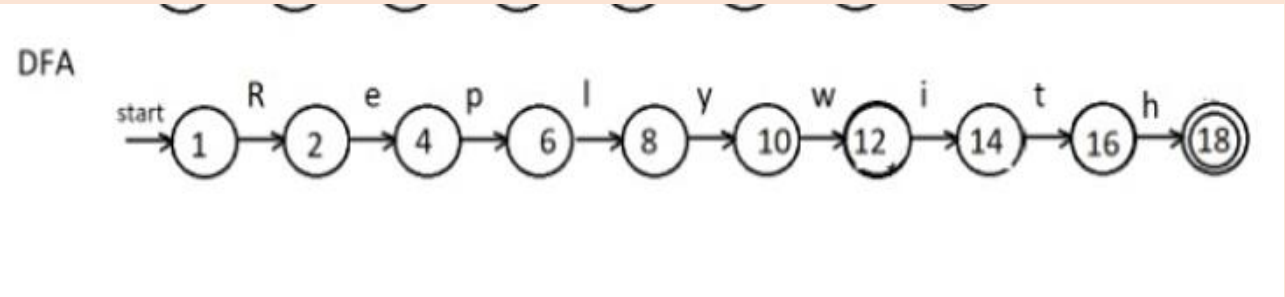
	R	o	t	a	t	e	w	h	e	n
1	2’									
2		4’								
4			6’							
6				8’						
8					10’					
10						12’				
12							14’			
14								16’		
16									18’	
18										20’
20										

12-Replywith:

N f a:



D f a:

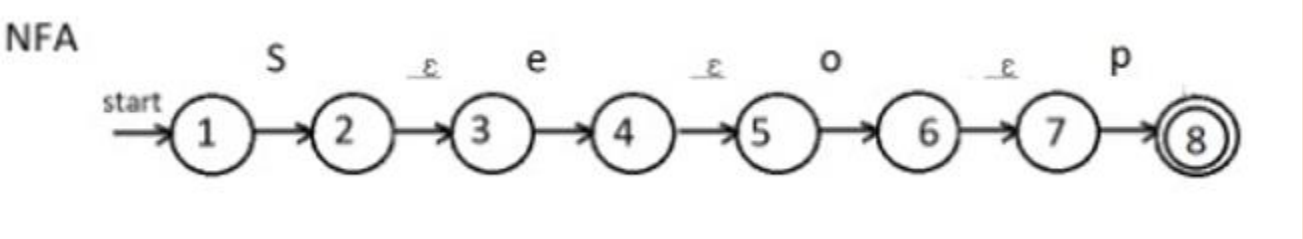


TRANSACTION TABLE :

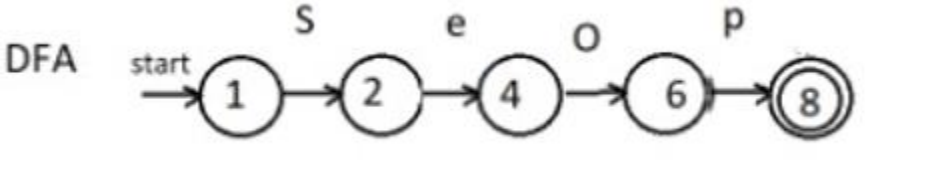
	R	e	p	l	y	w	i	t	h
1	2’								
2		4’							
4			6’						
6				8’					
8					10’				
10						12’			
12							14’		
14								16’	
16									18’
18									

13-Seop:

N f a:



D f a:

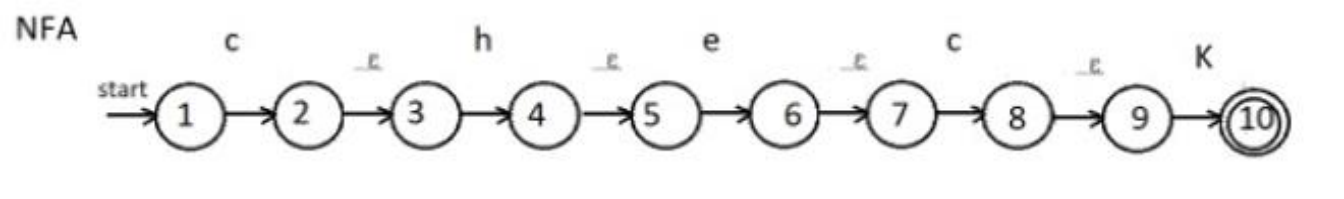


TRANSACTION TABLE :

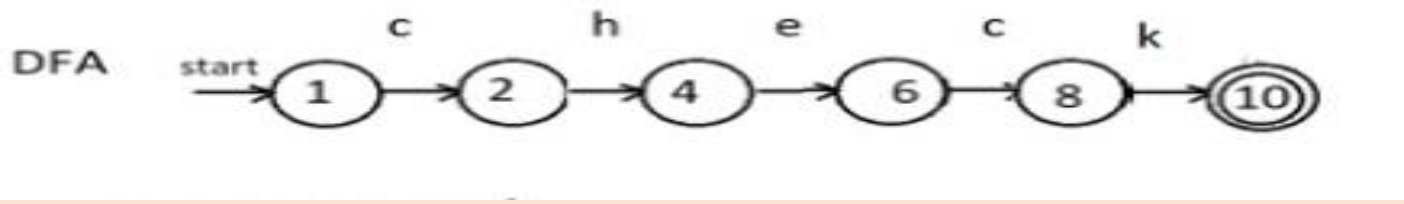
	S	e	o	p
1	2’			
2		4’		
4			6’	
6				8’
8				

14-Check:

N f a:



D f a:

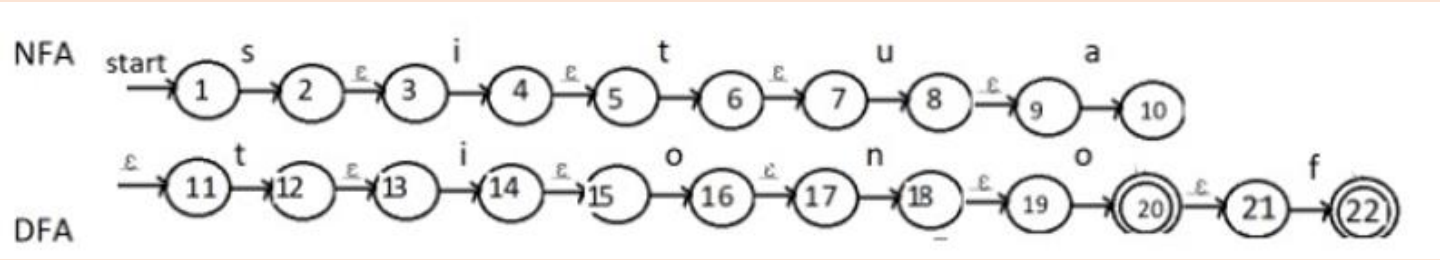


TRANSACTION TABLE :

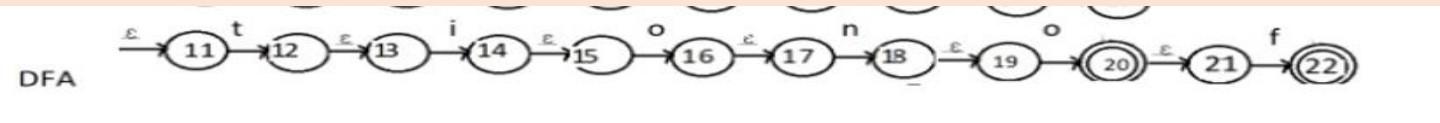
	C	h	e	c	k
1	2’				
2		4’			
4			6’		
6				8’	
8					10’
10					

15-Situationof:

N f a:



D f a:



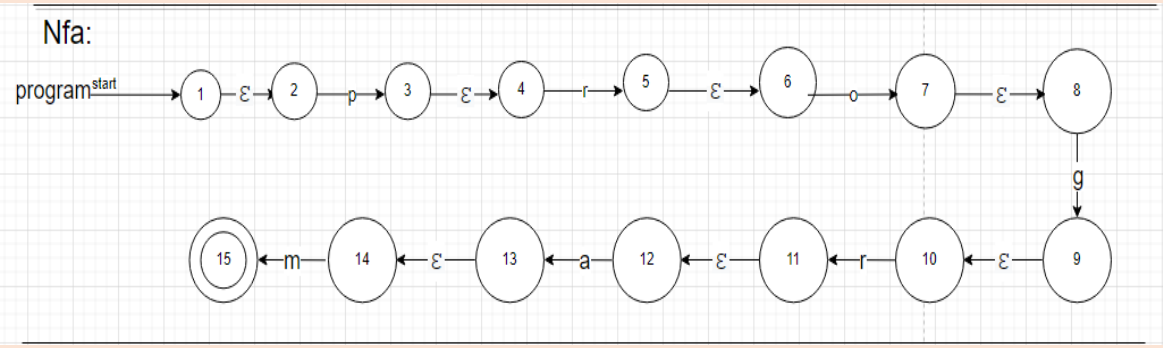
TRANSACTION TABLE :

	s	i	t	u	a	t	i	o	n	o	f
1	2’										
2		4’									
4			6’								
6				8’							
8					10’						
10						12’					
12							14’				
14								16’			
16									18’		
18										20’	
20											22’
22											

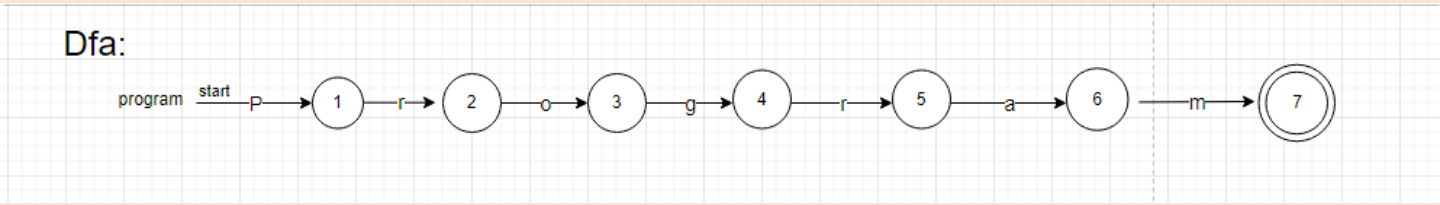
16-Program&End:

regex: program

N f a:



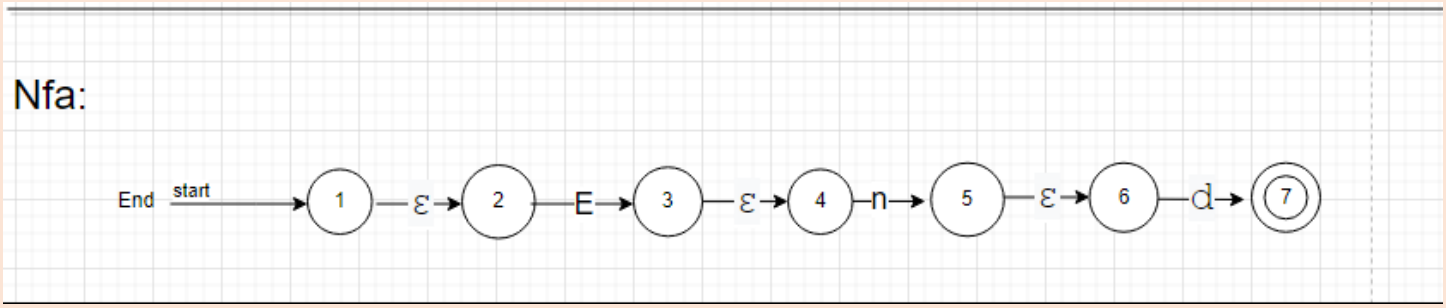
D f a:



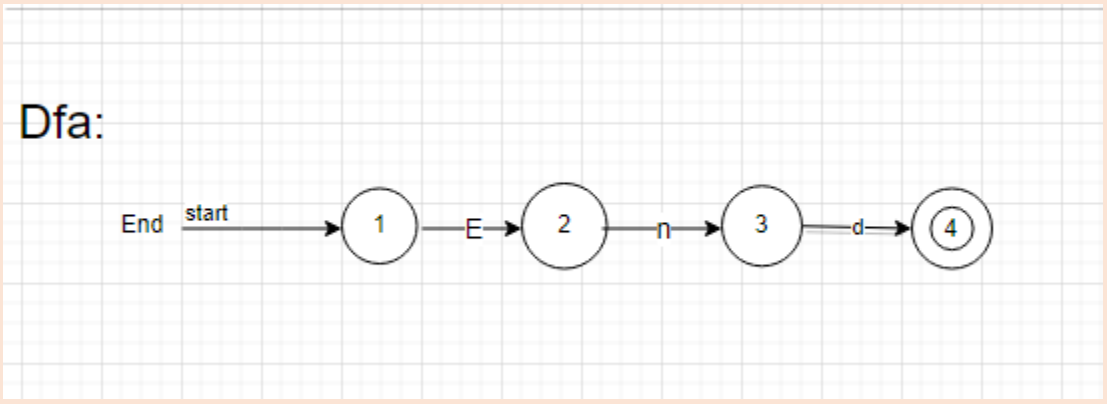
-END:

regex: End

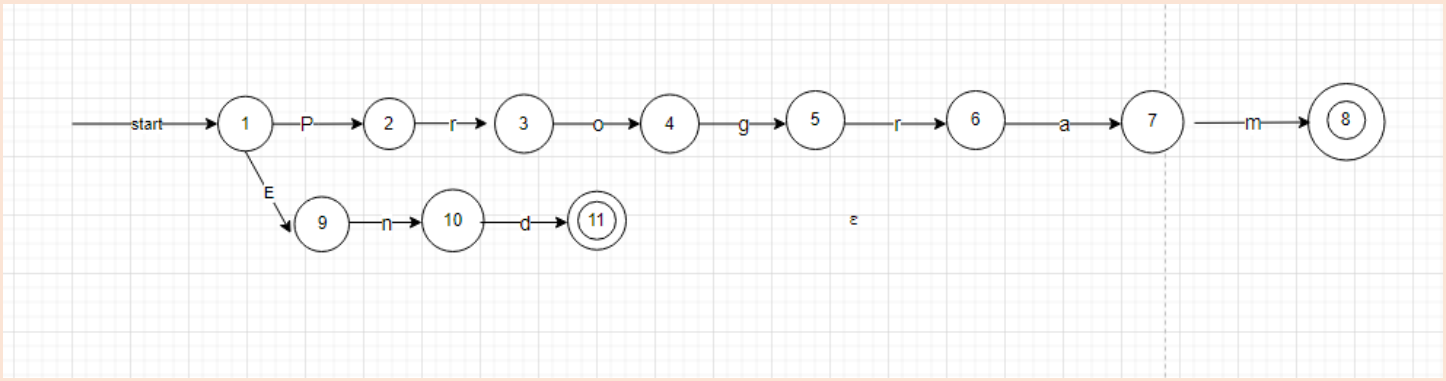
n f a:



D f a:



DFA of Program&End:

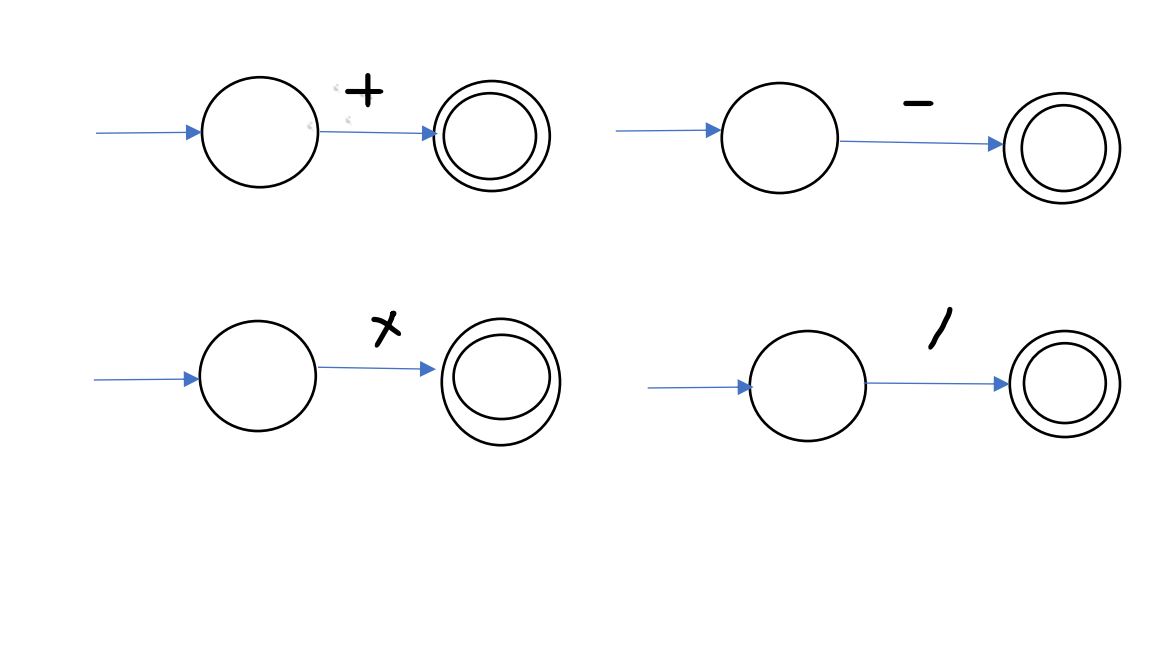


TRANSACTION TABLE :

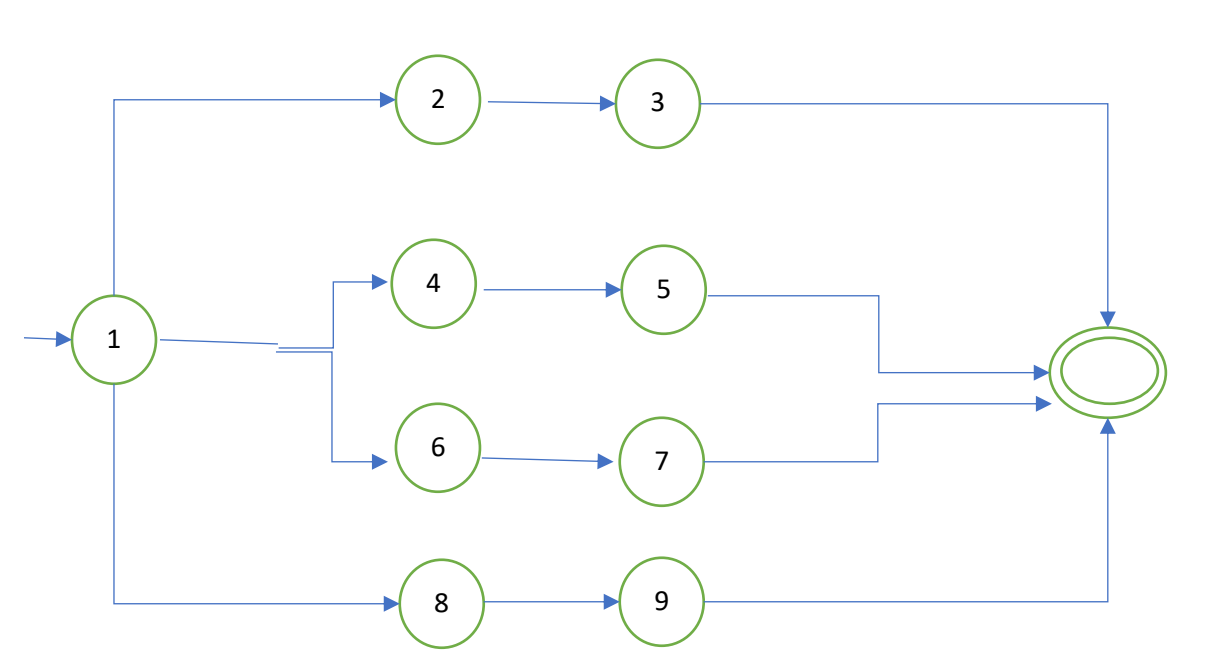
	p	r	o	g	a	m	e	n	d
1	2						9		
2		3							
3			4	5					
4									
5		6							
6					7				
7						8			
9								10	
10									11

18-Arithmetic Operation:

RE = + | - | * | /



N F A



TRANSACTION TABLE :

1? = {1,2,4,6,8}

2? = {2,3}

3? = {3}

4? = {4,5}

5? = {5}

6? = {6,7}

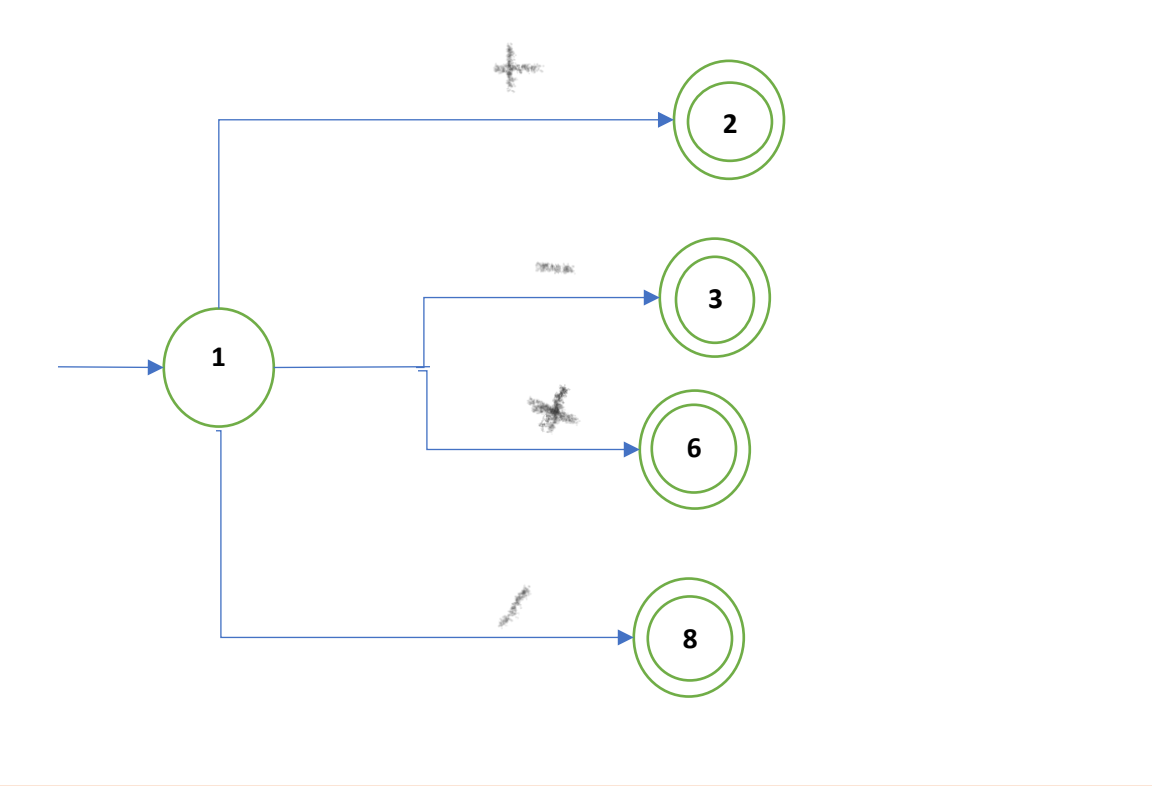
7? = {7}

8? = {8,9}

9? = {9}

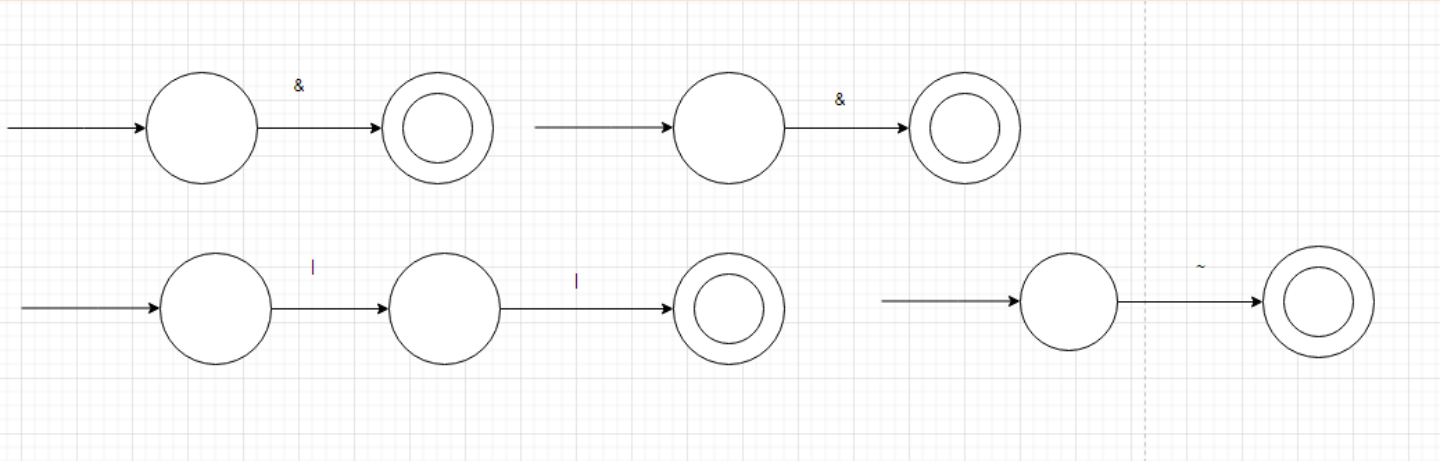
	+	-	*	/
1	**2**	**4**	**6**	**8**
2				
4				
6				
8				

D F A:

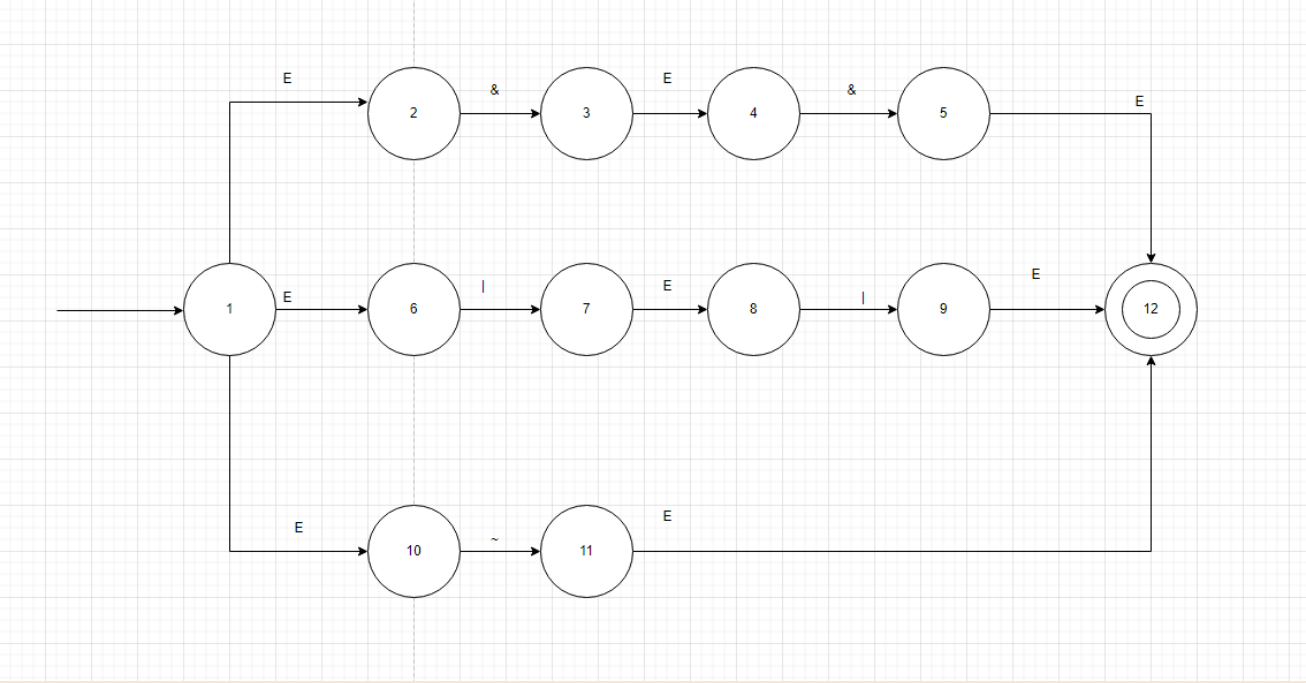


19-Logic operators:

RE = && | || | ~



N F A:



TRANSACTION TABLE :

1? = {1,2,6,10}

2? = {2,3}

3? = {3}

4? = {4,5}

6? = {6,7}

7? = {7}

8? = {8,9}

9? = {9}

5? = {5}

6? = {6,7}

7? = {7}

8? = {8,9}

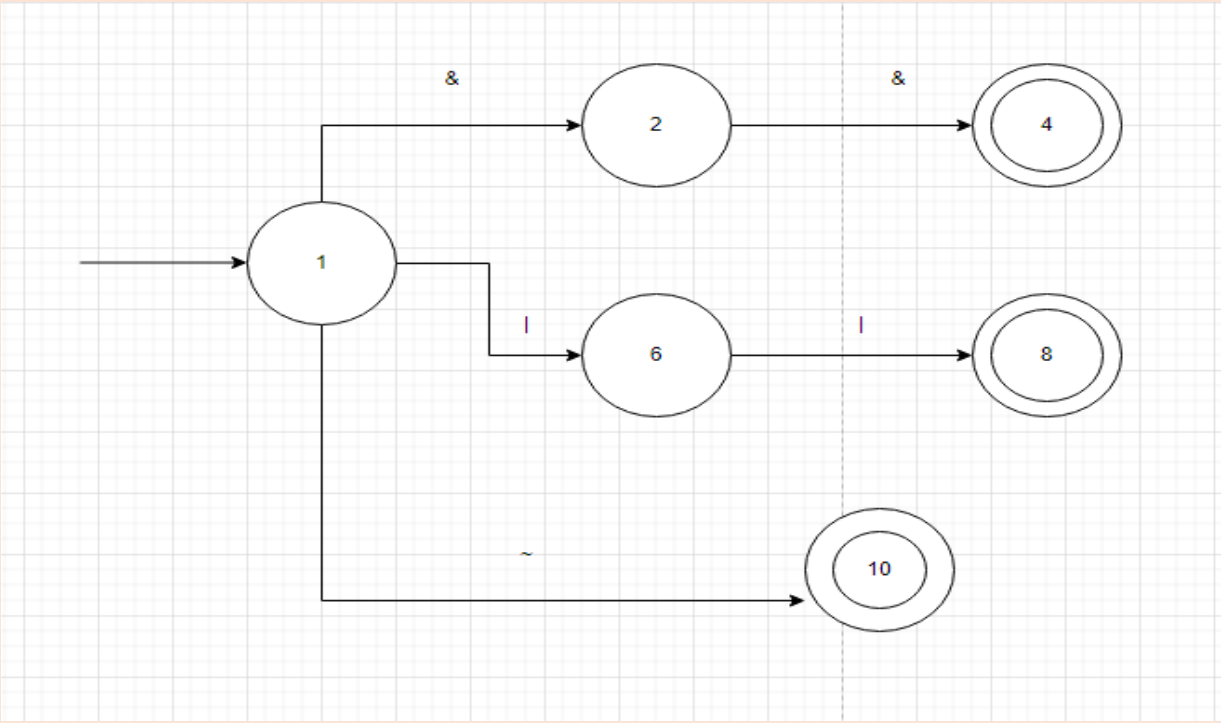
9? = {9}

10? = {10,11}

11? = {11}

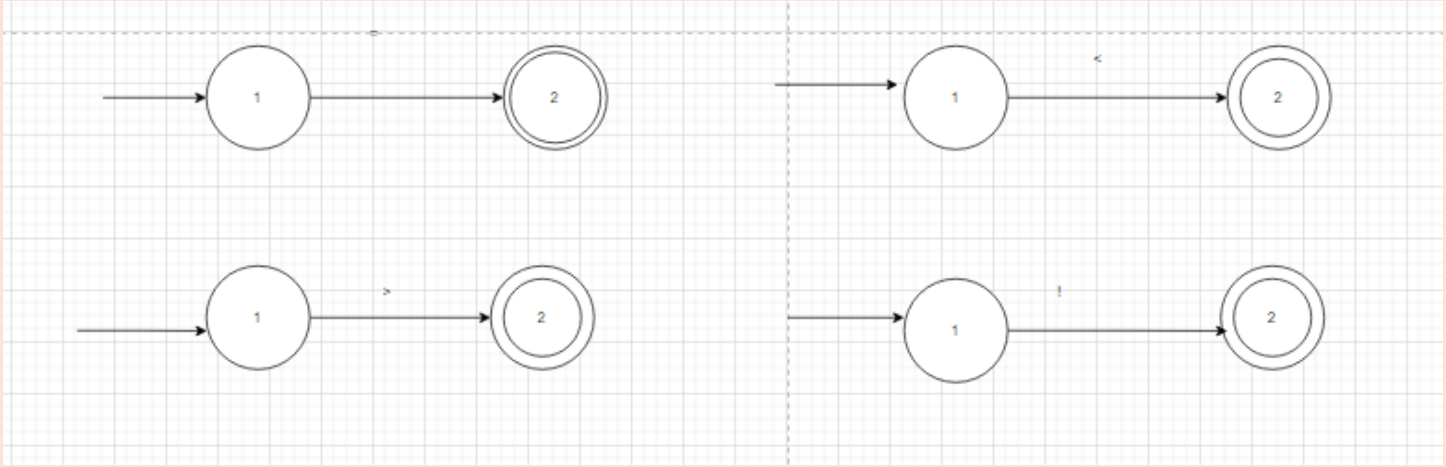
	&	&	|	|	~	
1	2		6		10	
2		4				
6				8		

D F A:



20-relational operators:

RE (==||!=|<=|>=)

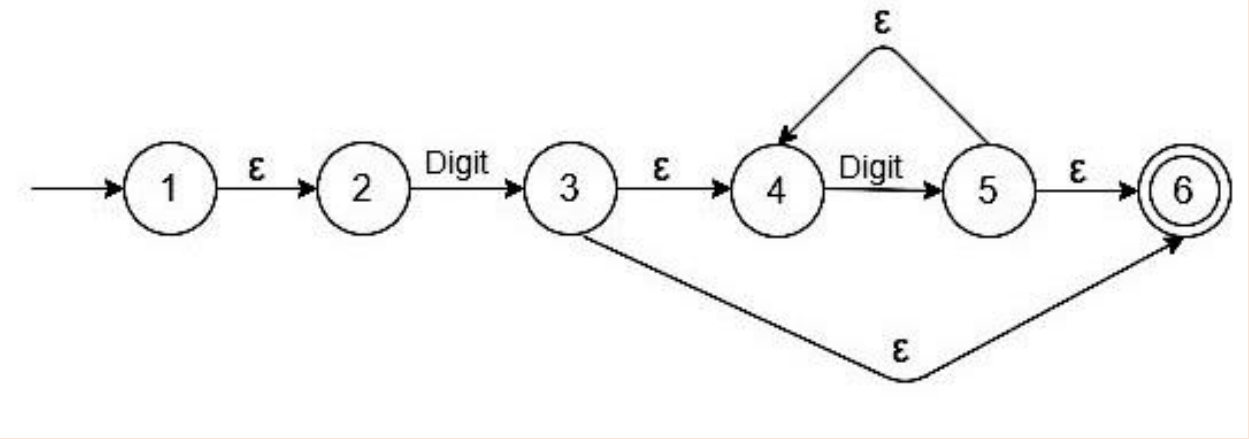


21-Numbers :

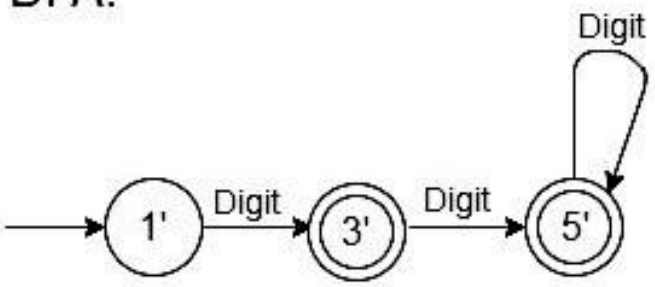
RE = Digit (Digit)*

RE = [0-9]+

NFA:



DFA:

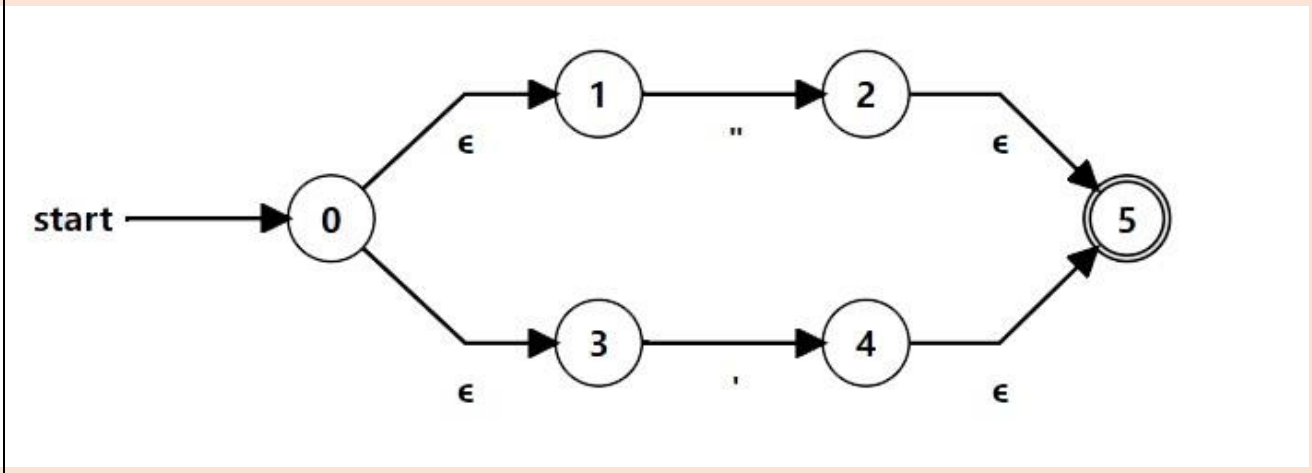


Transition Table	
	Digit
1'	3'
3'	5'
5'	5'

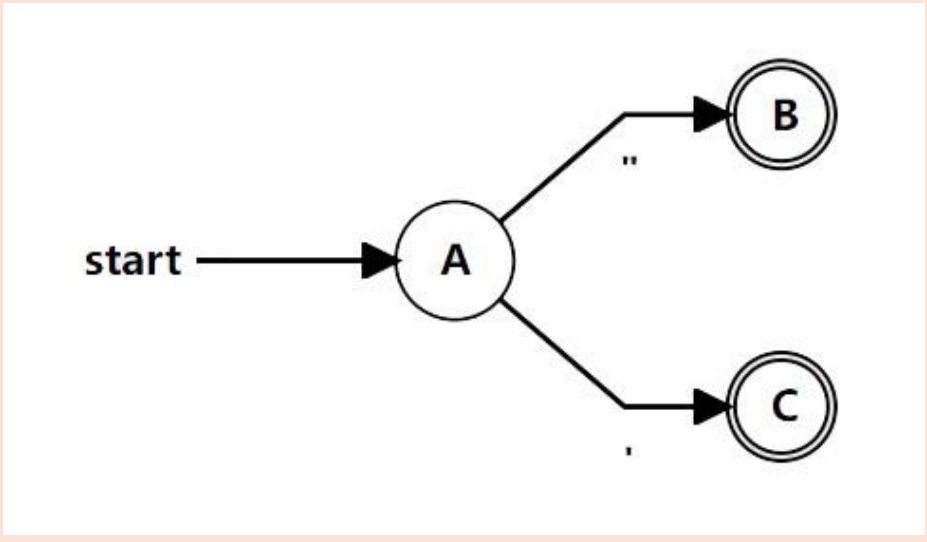
22-Quotation Marks

RE = ("|')

NFA:



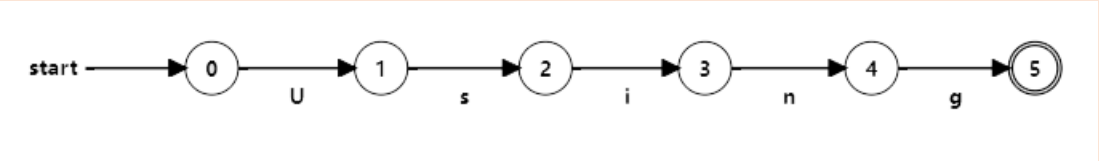
DFA:



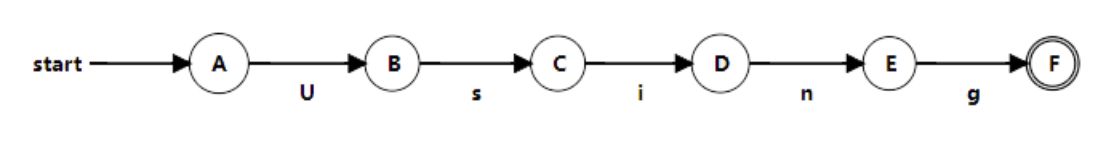
23-Using

RE = Using

NFA:



DFA:

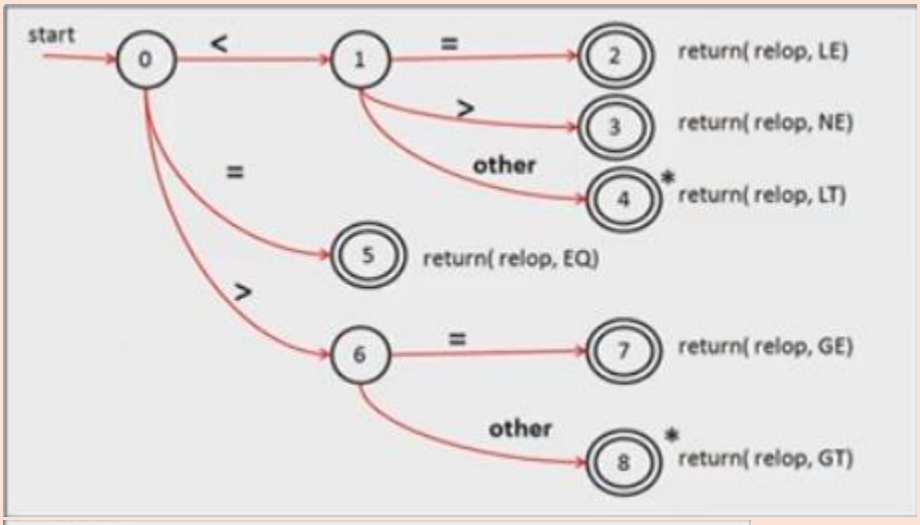


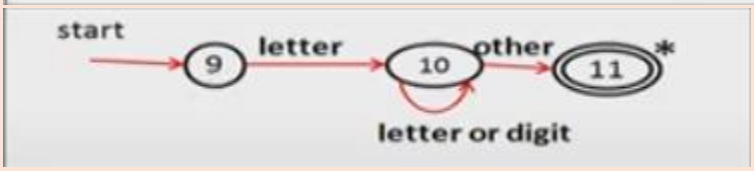
TRANSACTION TABLE :

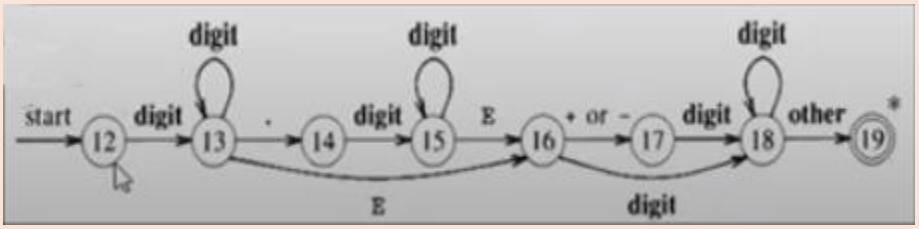
	u	s	i	n	g
1	2				
2		3			
3			4		
4				5	
5					6

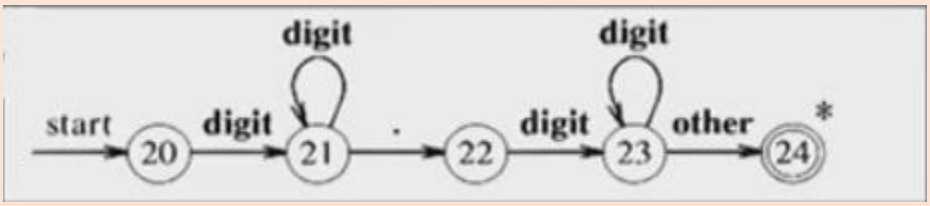
-Scanner:

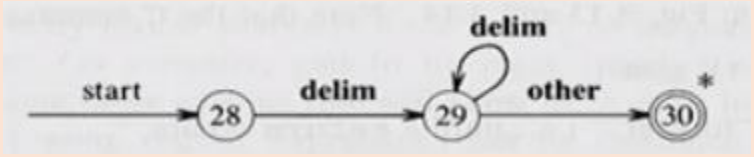
THE TRANSITION TABLE TO IMPLEMENT THE SCANNER:











Parse tree and Abstract syntax tree

1. Program → Program ClassDeclaration End.

Program -> Program`
Program` -> ClassDeclaration End. Program` | ε

First(Program) = First(Program`) = { Category | ε }
First(Program`) = {First(ClassDeclaration) , ε } = { Category | ε }

Follow(Program) = { $ }
Follow(Program`) = Follow(Program) = { $ }

****

2. ClassDeclaration→ Category ID{ Class_Implementation} | Category ID Derive { Class_Implementation}

ClassDeclaration -> Category ID ClassDeclaration`
ClassDeclaration` -> { Class_Implementation} | Derive { Class_Implementation}

First(ClassDeclaration) = { Category}
First(ClassDeclaration`) = { { , Derive }

Follow(ClassDeclaration) = First(End) = { End }
Follow(ClassDeclaration`) = Follow(ClassDeclaration) = { End }
****

3. Class_Implementation→ VarDeclaration Class_Implementation| MethodDeclaration Class_Implementation | Comment Class_Implementation | using_command Class_Implementation| Func _Call Class_Implementation |empty

First(Class_Implementation) =
{ Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , < , - , using , ID , ε }

Follow(Class_Implementation) = { } }
****

4. MethodDeclaration→ Func Decl ; | Func Decl { VarDeclaration Statements }

First(MethodDeclaration) = First(Func Decl) = { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical }

Follow(MethodDeclaration) = {First(Class_Implementation) - ε } U Follow(Class_Implementation) = { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , < , - , using , ID , } }

5. Func Decl →Type ID (ParameterList)

First(Func Decl) = First(Type) = { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical }

Follow(Func Decl) = { ; , { }
****

6. Type → Ilap | Silap | Clop | Series | Ilapf | Silapf | None | Logical

First(Type) = { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical }

Follow(Type) = { ID }
****

7. ParameterList →empty | None | Non-Empty List

First(ParameterList) = { ε, None , Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical }

Follow(ParameterList) = {) }
****

8. Non-Empty List→ Type ID | Non-Empty List , Type ID

First(Non-Empty List) = { Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical }

Follow(Non-Empty List) = Follow(ParameterList) = {) }
****

9. VarDeclaration→ empty | Type ID_List ; VarDeclaration

First(VarDeclaration) = { ε, Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical }

Follow(VarDeclaration) = {First(Class_Implementation) - ε } U Follow(Class_Implementation) U {First(Statements) - ε} U Follow(MethodDeclaration) U = =
{ Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , < , - , using , ID , } , Assignment , If, Rotatewhen , Continuewhen , terminatethis , read , write , Replywith , =}

****

10. ID_List →ID | ID_List , ID

First(ID_List) = { ID }
Follow(ID_List) = { ; }
****

11. Statements→empty | Statement Statements

First(Statements) = { ε , Assignment , If, Rotatewhen , Continuewhen , terminatethis , read , write , Replywith }

Follow(Statements) = { } }
****

12. Statement→Assignment | If _Statement | Rotatewhen _Statement | Continuewhen_Statement | terminatethis _Statement|read (ID); | write (Expression); | Replywith _ Statement

First(Statement) = { Assignment , If, Rotatewhen , Continuewhen , terminatethis , read , write , Replywith }

Follow(Statement) = { First(Statements) - ε } U Follow(Statements)
= {Assignment , If, Rotatewhen , Continuewhen , terminatethis , read , write , Replywith , } }
****

13. Assignment→ VarDeclaration = Expression;

First(Assignment) ={ First(VarDeclaration) - ε } U = U First(Expression) =
{ Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , = , ID , Number }

Follow(Assignment) = Follow(Statement) =
{Assignment , If, Rotatewhen , Continuewhen , terminatethis , read , write , Replywith , } }
****

14. Func _Call → ID (Argument_List) ;

First(Func _Call) = { ID }
Follow(Func _Call) = {First(Class_Implementation) - ε } U Follow(Class_Implementation) =
{ Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , < , - , using , ID , } }

****

15. Argument_List →empty | NonEmpty_Argument_List

First(Argument_List) = { ε , ID , Number }

Follow(Argument_List) = {) }
****

16. NonEmpty_Argument_List →Expression | NonEmpty_Argument_List , Expression

First(NonEmpty_Argument_List) = First(Expression) = { ID , Number}

Follow(NonEmpty_Argument_List) = Follow(Argument_List) = {) }
****

17. Block Statements→{statements }

First(Block Statements) = { { }

Follow(Block Statements) = Follow(If _Statement) ∪ Follow(Rotate _Statement) ∪ Follow(Continuewhen _Statement) =
= {Assignment , If, Rotatewhen , Continuewhen , terminatethis , read , write , Replywith , } }

****

18. If _Statement→ if (Condition _Expression) Block Statements

First(If _Statement) = { if }

Follow(If _Statement) = Follow(Statement) =
{Assignment , If, Rotatewhen , Continuewhen , terminatethis , read , write , Replywith , } }

****

19. Condition _Expression→ Condition |Condition Condition _Op Condition

First(Condition _Expression) = First(Condition) = First(Expression) = { ID , Number}

Follow(Condition _Expression) = {) }

20. Condition _Op → and | or

First(Condition _Op) = { and , or }

Follow(Condition _Op) = First(Condition) = { ID , Number}
****

21. Condition→ Expression Comparison _Op Expression

First(Condition) = First(Expression) = { ID , Number}

Follow(Condition) = Follow(Condition _Expression) U Follow(Condition _Op) = {) , ID , Number }

****

22. Comparison _Op → == | != | > | >= | < | <=

First(Comparison _Op) = { = , ! , > , < }

Follow(Comparison _Op) = First(Expression) = { ID , Number}
****

23. Rotate _Statement → Rotate when(Condition _Expression) Block Statements

First(Rotate _Statement) = { Rotate }

Follow(Rotate _Statement) = NA
****

24. Continuewhen _Statement →Continuewhen (expression ; expression ; expression) Block Statements

First(Continuewhen _Statement) = { Continuewhen)

Follow(Continuewhen _Statement) = Follow(Statement) =

{Assignment , If, Rotatewhen , Continuewhen , terminatethis , read , write , Replywith , } }
****

25. Replywith _Statement→ Replywith Expression ; | returnID ;

First(Replywith _Statement) = { Replywith , returnID }

Follow(Replywith _Statement) = Follow(Statement) =
{Assignment , If, Rotatewhen , Continuewhen , terminatethis , read , write , Replywith , } }

****

26. terminatethis _Statement→ terminatethis;

First(terminatethis _Statement) = { terminatethis }

Follow(terminatethis _Statement) = Follow(Statement) =
{Assignment , If, Rotatewhen , Continuewhen , terminatethis , read , write , Replywith , } }

****

27. Expression → Term |Expression Add_Op Term

First(Expression) = { ID , Number}

Follow(Expression) = { ; ,) , = , ! , > , < }
****

28. Add_Op → + | -

First(Add_Op) = { + , - }

Follow(Add_Op) = First(Term) = { ID , Number }
****

29. Term→ Factor| Term Mul_Op Factor

First(Term) = { ID , Number }

Follow(Term) = Follow(Expression) = { ; ,) , = , ! , > , < }

30. Mul_Op→* | /

First(Mul_Op) = { * , / }

Follow(Mul_Op) = First(Factor) = { ID , Number}

****

31. Factor→ ID| Number

First(Factor) = { ID , Number}

Follow(Factor) = Follow(Term) = Follow(Expression) = { ; ,) }
****

32. Comment →<* STR *> | -- STR

First(Comment) = { < , - }

Follow(Comment) = {First(Class_Implementation) - ε } U Follow(Class_Implementation) =
{ Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , < , - , using , ID , } }
****

33. using_command →using(F_name.txt);

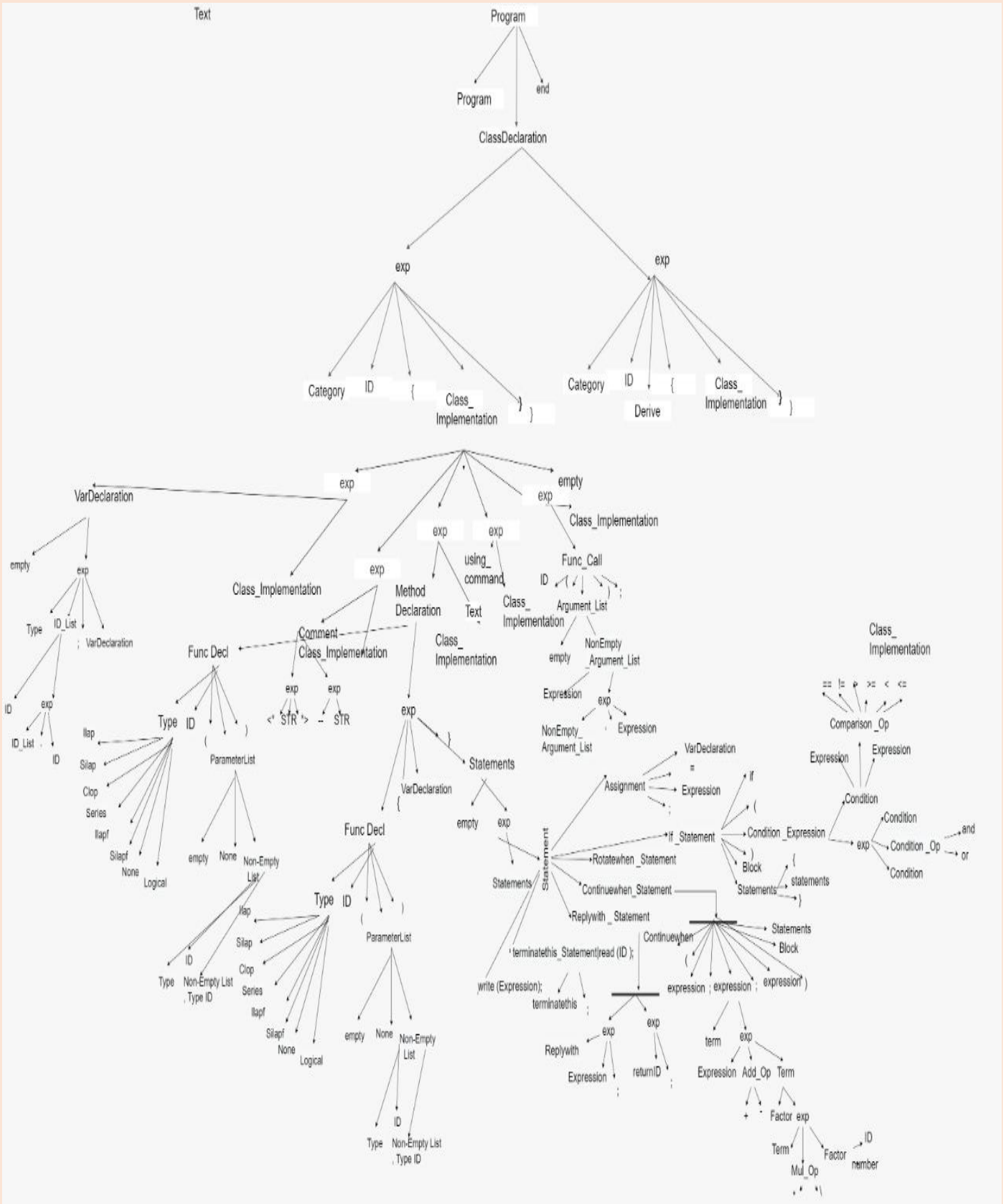
First(using_command) = { using }

Follow(using_command) = {First(Class_Implementation) - ε } U Follow(Class_Implementation) =
{ Ilap , Silap , Clop , Series , Ilapf , Silapf , None , Logical , < , - , using , ID , } }

****

34. F_name →STR

First(F_name) = { STR }
Follow(F_name) = { .txt }


Text
Program
Program
end
ClassDeclaration
exp
exp
Category
ID
{
Class_
Implementation
}
}
Category
ID
{
Derive
Class_
Implementation
}
}
exp
empty
exp
Class_Implementation
VarDeclaration
Class_Implementation
exp
exp
exp
using_
command
Func_Call
ID
(
)
;
Method
Declaration
Text
Class_
Implementation
Argument_List
empty
NonEmpty
_Argument_List
Comment
Class_Implementation
Class_
Implementation
empty
exp
Type
ID_List
;
VarDeclaration
Func Decl
exp
exp
ID
exp
<
STR
>
-
STR
Expression
exp
NonEmpty_
Argument_List
Expression
Class_
Implementation
== != > >= < <=
Comparison_Op
ID_List
,
ID
Type
ID
(
)
Expression
Expression
Condition
Condition
Condition_Op
and
or
Condition
Ilap
Silap
Clop
Series
Ilapf
Silapf
None
Logical
ParameterList
empty
None
Non-Empty
List
exp
}
Statements
VarDeclaration
{
Assignment
VarDeclaration
=
Expression
;
if
(
If_Statement
Condition_Expression
)
Block
Statements
{
statements
}
Rotatewhen_Statement
Continuewhen_Statement
Replywith_Statement
Statements
Statement
empty
exp
Func Decl
Type
ID
(
)
ParameterList
Ilap
Silap
Clop
Series
Ilapf
Silapf
None
Logical
Type
ID
,
Non-Empty List
, Type ID
empty
None
Non-Empty
List
Type
ID
,
Non-Empty List
, Type ID
terminatethis_Statement(read (ID);
write (Expression);
terminatethis
;
Continuewhen
(
expression
;
expression
;
expression
)
Statements
Block
Replywith
Expression
;
exp
returnID
;
exp
term
exp
Expression
Add_Op
Term
+
-
Factor
exp
Term
Mul_Op
*
/
\
Factor
ID
number