Lexar Glossary

<Identifier>
is a sequence of letters, digits, or "_". However, the first character must be a letter. Upper and lower cases are the same**.**
Examples:

1. X - - -
2. X 1 2 3
3. a b c

<Integer>
is an unsigned decimal integer, i.e., a sequence of decimal digits**.**
Examples:

1. 00123
2. 12300
3. 00000

<Real>
is an integer followed by "." and Integer, e.g., 123.00
Examples:

1. 123.45
2. 0.000
3. 16.56

Keywords-

1. Function
2. Integer
3. Real
4. Boolean
5. If
6. Endif
7. Else
8. Return
9. Print
10. Scan
11. While
12. True
13. False

Operators-

1. +
2. -
3. *
4. \
5. = =
6. ! =
7. <
8. >
9. =<
10. =>
11. =

Separators-

1. $
2. (
3.)
4. ,
5. {
6. }
7. ;
8. :
9. |
10. -

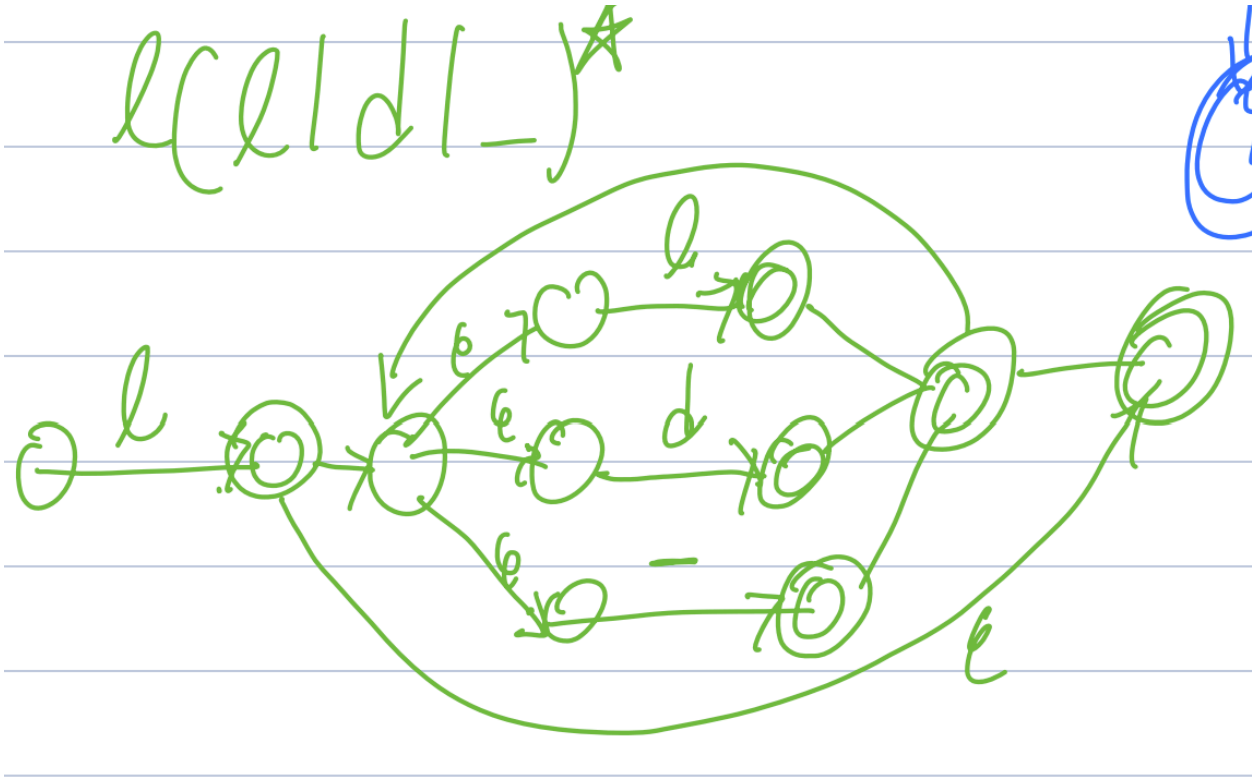
Assignment #1

Step 1: Write tokens in terms of regular expressions

- <Identifier> = **𝓵** (**𝓵** | **𝓭** | _)^*
- <Real> = **𝓭**^+ **.𝓭**^+
- <Integer> = **𝓭**^+
- **Keywords/ Operators/ Separators**
    - Fixed number of them so we just need to add them in manually.

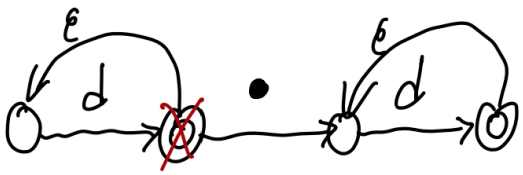
Step 2: Build a NFSM that recognizes Regular Expressions

ID-

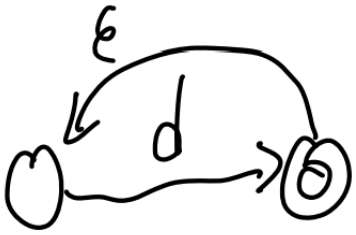


FSM 1

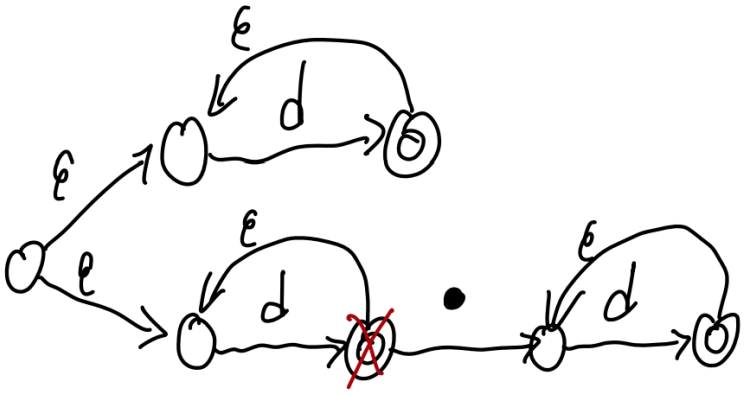
< Real >



< Integer >

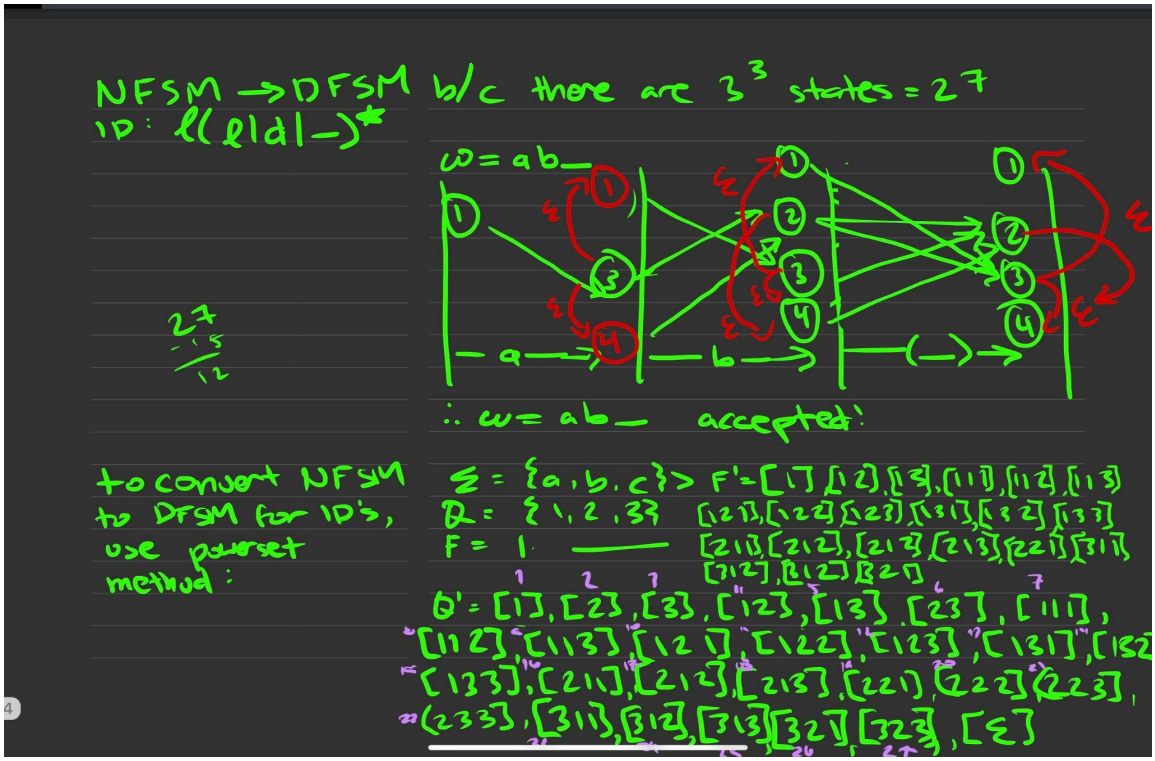


< Integer > + < Real >

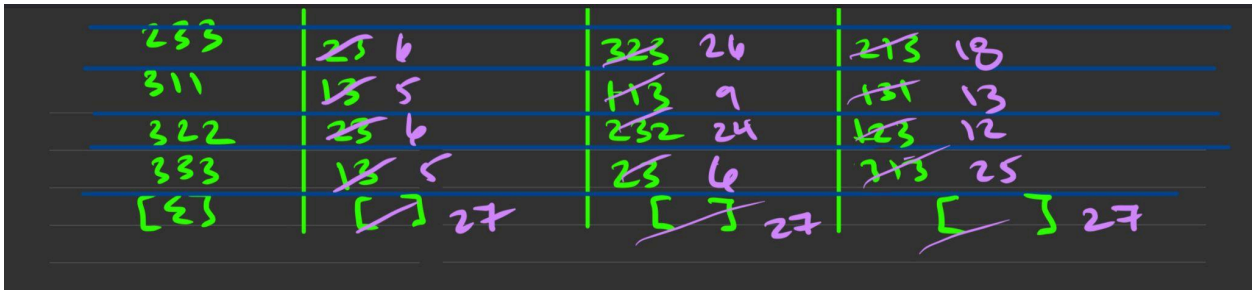


Step 3: Convert the NFSM to a DFSM

ID NFSM- DFSM



	l	d	(_)
[1]	1	4	5
[2]	7	10	11
[3]	14	15	17
[12]	17	16	18
[13]	13	15	25
[23]	2	12	18
[11]	1	7	16
[112]	4	16	17
113	5	23	12
121	4	8	16
122	4	17	17
123	5	4	26
131	15	5	23
132	6	12	24
133	5	25	18
211	4	8	10
212	10	19	4
213	15	24	10
221	11	17	8
222	2	15	21
223	6	21	23



INT NFSM- DFSM

REAL NFSM-DFSM

NFSM ⇒ DFSM $= d^+ \cdot d^+ = 3^3 = 27$ states

<real>

$\Sigma = \{d^+, \ldots, d^+\}$

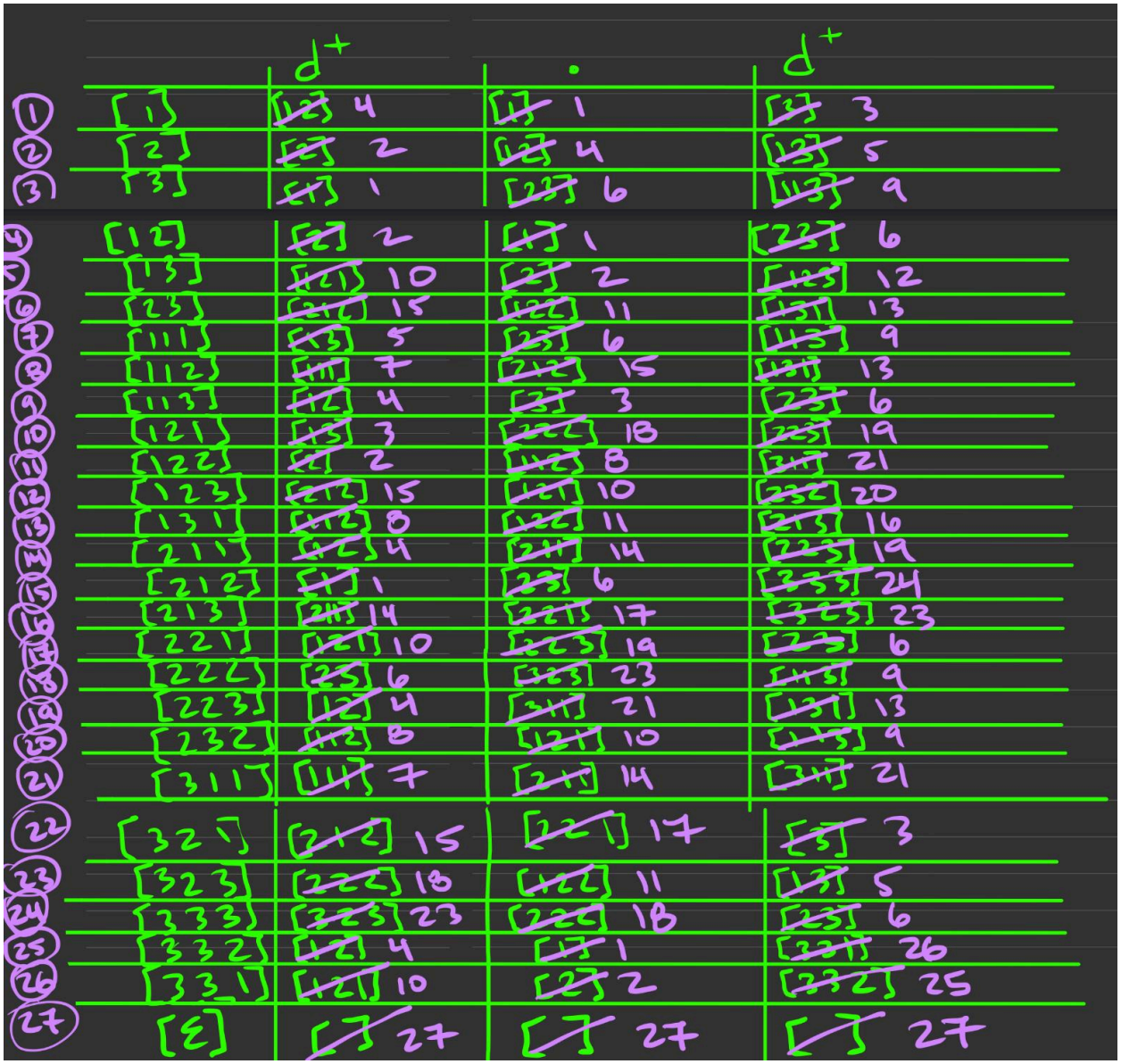
$Q = 1, 2, 3$

$q_0 = 1$

$F = 3$

$Q' = \{[1], [2], [3], [12], [13], [23], [111], [112], [113], [121], [122], [123], [131], [211], [212], [213], [221], [222], [223], [232], [311], [321], [323], [333], [332], [331], [\varepsilon]\}$

$F' = \{[3], [13], [23], [113], [123], [131], [213], [223], [232], [311], [321], [323], [333], [332], [331]\}$



Step 4: Modify the DFSM to recognize an individual token

- One accepting state per token type
- Meaning of acceptance changes
- Sometimes, a char pointer needs to back up by one character (unget) C++

Test Cases-
#1- Output correct?

[* test 1 small*]

while (fahr <= upper) a = 23.00; [* this is a sample *]

a+b = c;

2
_a_

while

$ new

Output:

Token	**Lexeme**
keyword	while
separator	(

identifier	fahr
operator	<=
identifier	upper
separator	)
identifier	a
operator	=
real	23.00
separtor	;
Illegal	_a_
identifier	a
operator	+
identifier	b
operator	=
identifier	c
Integer	2
keyword	while
separator	$
identifier	new

#2- Output correct?
[*test 2 medium*]

```
[* this is comment for this sample code which
converts Fahrenheit into Celcius *]
$
function convertx (fahr integer)
{
return 5 * (fahr -32) / 9;
```

```
}
$
integer low, high, step; [* declarations *]
$
scan (low, high, step);
while (low <= high )
{ print (low);
print (convertx (low));
low = low + step;
}
endwhile
$
```

Output:

Token	**Lexeme**
separator	$
keyword	function
identifier	convertx
separator	(
identifier	fahr
keyword	integer
separator	)
separator	{
keyword	return
integer	5
operator	*
separator	(
identifier	fahr
operator	-

integer	32
separator	)
operator	/
integer	9
separator	;
separator	}
separator	$
keyword	integer
identifier	low
separator	,
identifier	high
separator	,
identifier	step
separator	;
separator	$
keyword	scan
separator	(
identifier	low
separator	,
identifier	high
separator	,
identifier	step
separator	)
separator	;
keyword	while

separator	(
identifier	low
operator	<=
identifier	high
separator	)
separator	{
keyword	print
separator	(
identifier	low
separator	)
separator	;
keyword	print
separator	(
identifier	convertx
separator	(
identifier	low
separator	)
separator	)
separator	;
identifier	low
operator	=
identifier	low
operator	+
identifer	step
separator	;

separator	}
keyword	endwhile
separator	$

#3- Output correct?
[* test 3 large*]

```
print random
print time

def generate_random_code(lines):
    code = []
    for _ in range(lines):
        line = generate_random_line()
        code.append(line)
    return code

def generate_random_line():
    line = ""
    num_operations = random.randint(1, 3)
    for _ in range(num_operations):
        operation = random.choice(["if", "elif", "else", "for", "while", "def", "return", "print"])
        line += operation + " "
        if operation in ["if", "elif", "else", "for", "while", "def"]:
            line += generate_random_condition() + ":"
        elif operation == "print":
            line += generate_random_string() + " "
        elif operation == "return":
            line += generate_random_number() + " "
    return line
    print 3.14
    $

    2/0 = number

    $
    $
```

Output:

Token	**Lexeme**
keyword	print
identifier	random
keyword	print
identifier	time
identifier	def
identifier	generate_random_code
separator	(
identifier	lines
separator	)
separator	:
identifier	code
operator	=
separator	[
separator	]
keyword	for
illegal	_
identifier	in
identifier	range
separator	(
identifier	lines
separator	)
separator	:

identifier	line
operator	=
identifier	generate_random_line
separator	(
separator	)
identifier	code
separator	.
identifier	append
separator	(
identifier	line
separator	)
keyword	return
identifier	code
identifer	def
identifier	generate_random_line
separator	(
separtor	)
separtor	:
identifier	line
operator	=
illegal	“
illegal	“
identifier	num_operations
operator	=
identifer	random

separator	.
identifer	randint
separator	(
integer	1
separator	,
integer	3
separator	)
keyword	for
illegal	_
identifer	in
identifer	range
separator	(
identifier	num_operations
separator	)
separator	:
identifier	operation
operator	=
identifier	random
separator	.
identifier	choice
separator	(
separator	[
separator	“
keyword	if
separator	“

separator	,
separator	“
keyword	elif
separator	“
separator	,
separator	“
keyword	else
separator	“
separator	,
separator	“
keyword	for
separator	“
separator	,
separator	“
keyword	while
separator	“
separator	,
separator	“
identifier	def
separator	“
separator	,
separator	“
keyword	return
separator	“
separator	,

separator	“
identifier	print
separator	“
separator	]
separator	)
identifier	line
operator	+=
identifier	operation
operator	+
separator	“
separator	“
keyword	if
identifier	operation
identifier	in
operator	[
illegal	“
keyword	if
illegal	“
operator	,
illegal	“
keyword	elif
illegal	“
operator	,
illegal	“
keyword	else

illegal	“
operator	,
illegal	“
keyword	for
illegal	“
operator	,
illegal	“
keyword	while
illegal	“
operator	,
illegal	“
identifier	def
illegal	“
separator	]
separator	:
identifier	line
operator	+=
identifier	generate_random_condition
separator	(
separator	)
operator	+
illegal	“
separator	:
illegal	“
keyword	elif

identifier	operation
operator	==
illegal	“
identifier	print
illegal	“
separator	:
identifier	line
operator	+=
identifier	generate_random_string
separator	(
separator	)
operator	+
illegal	“
illegal	“
keyword	elif
identifer	operation
operator	==
illegal	“
keyword	return
illegal	“
separator	:
identifier	line
operator	+=
identifier	generate_random_number
separator	(

separator	)
operator	+
illegal	“
illegal	“
keyword	return
identifier	line
identifier	print
real	3.14
separator	$
integer	2
operator	/
integer	0
operator	=
identifier	number
separator	$
separator	$