Q1:

1. What are the missing phases of the compilation:
   Lexical Analysis
   ____
   Semantic Analysis
   Optimization
   ____

0. Below are the C (C99) expressions which will cause an error during compilation. Indicate in which compilation phase will the error be reported. (assuming there is no expression before them)

a. iNt a = 467;
a. char = ‘E’;
b. long C = 4.67;
c. for (int i = 4 ; i < 6) {}
d. float b = 46.7 * 4 +6);
e. int ece = “467”;

0.

a. Write a regular expression for the set of all strings over the alphabet of {x, y} that matches all strings that do not contain two (or more) x or y consecutively; and the length of the string must be even (excluding 0). Examples:
   In the set of the string: xy, yx, xyxy, yxyx
   Not in the set of the string: xxy, xyy, xyx, ϵ
   b. Draw the NFA of the above regular expression

a. Convert the NFA to DFA. List each state’s ϵ-closures.
a. Provide a minimum DFA using partitioning algorithm.

Q2: Consider the following grammar:

S → S + A | A
A → A * B | B
B → * C | id
C → (S) | id

Non-terminals are {S, A, B, C, D}, terminals are {id, +, *, (,)}.

1. Is this grammar left recursive? If true, eliminate the left recursion and rewrite the grammar in separate productions for each OR closure.
   I.e. if you have Z → X | Y, write Z → X and Z → Y instead.
   Number all the productions after you finish.

0. List the FIRST set of each non-terminal in the grammar you write **in part one**.

Non-terminal	FIRST

0. List the FOLLOW set of each non-terminal in your grammar of part one.

Non-terminal	FOLLOW

0. Draw the LL(1) parsing table below. You may use the number in part one.

Non-terminal	+	*	id	(	)	$

0. Does the following string belong to the language of this grammar? If so, write down a left-most derivation. If not, where will the error happen?

a. id + * id * * (id * id)

a. * id * + (id + * id)

Matched	Stack	Input	Move

Answer:

Q1

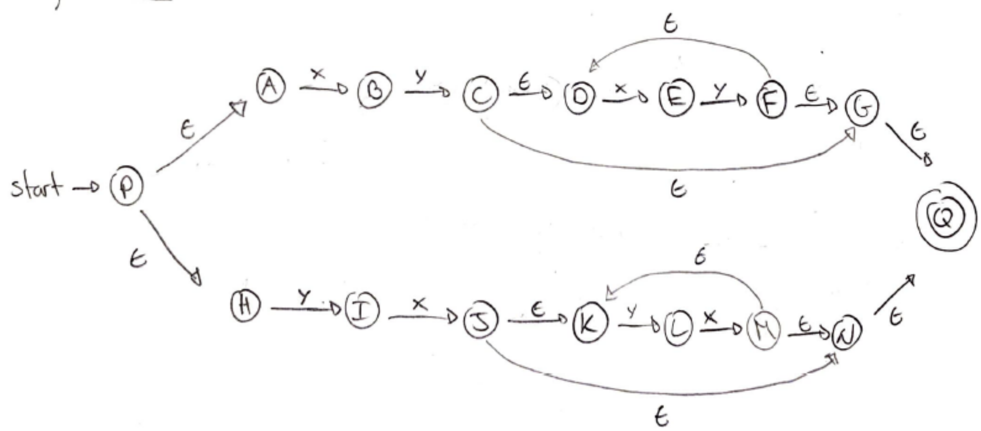
1. Parsing
   Code generation

0.

1.) Syntax analysis
2.) Syntax analysis
3.) Semantic analysis
4.) Syntax analysis
5.) Lexical analysis
6.) Semantic analysis

0. (xy)+ | (yx)+ **or** xy (xy)* | yx(yx)*

1) NFA :



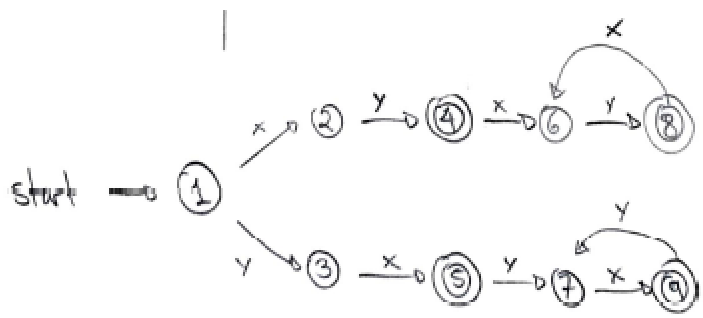
2) NFA to DFA

NFA State	ε - closure
A	∅
B	∅
C	D, G, Q
D	∅
E	∅
F	D, G, Q
G	Q
H	∅
I	∅
J	K, N, Q
K	∅
L	∅
M	K, N, Q
N	Q
→ P	A, H
* Q	∅

	DFA state	x	y
1	→ PAH	B	I
2	B	∅	CDGQ
3	I	JKNQ	∅
4	* CDGQ	E	∅
5	* JKNQ	∅	L
6	E	∅	FDGQ
7	L	MKNQ	∅
8	* FDGQ	E	∅
9	* MKNQ	∅	L

↑

DFA states have been renumbered for simplicity



← initial DFA

3) Apply partitioning algorithm.

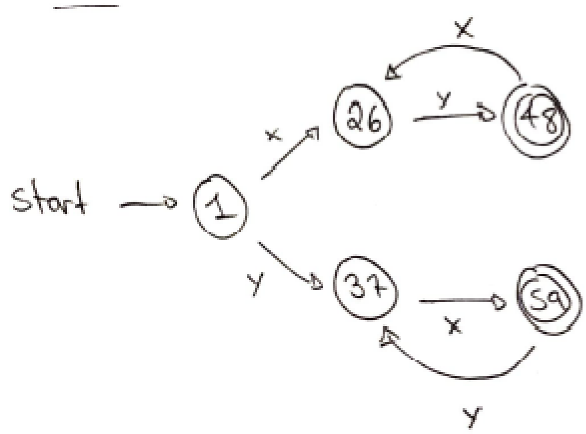
	{1	2	3	6	7}	{4	5	8	9}
x	2	∅	5	∅	9	6	∅	6	∅
y	3	4	∅	8	∅	∅	7	∅	7

continued:

	{1}	{2	6}	{3	7}	{4	8}	{5	9}
x	2	∅	∅	5	9	6	6	∅	∅
y	3	4	8	∅	∅	∅	∅	7	7

↑ these are our minimal states

Minimum state DFA:



Q2

1. Yes.

S → A S' ①
S' → + A S' ②
S' → ϵ ③
A → B A' ④
A' → * B A' ⑤
A' → ϵ ⑥
B → * C ⑦
B → id ⑧
C → (S) ⑨
C → id ⑩

0.

Non-terminal	FIRST
S	*, id
S'	+, ϵ
A	*, id
A'	*, ϵ
B	*, id
C	(, id

0.

Non-terminal	FOLLOW
S	$,)
S'	$,)
A	$, +,)
A'	$, +,)
B	*, $, +,)
C	*, $, +,)

0.

	+	*	id	(	)	$
S		①	①			
S'	②				③	③
A		④	④			
A'	⑥	⑤			⑥	⑥
B		⑦	⑧			
C			⑩	⑨		

0.

a. Yes.

Matched	Stack	Input	Move
	S$	id + * id * * (id * id)	
	A S'$	id + * id * * (id * id)	①
	B A' S'$	id + * id * * (id * id)	④
	id A' S'$	id + * id * * (id * id)	⑧
id	A' S'$	+ * id * * (id * id)	match
id	S'$	+ * id * * (id * id)	⑥
id	+ A S'$'	+ * id * * (id * id)	②
id +	A S'$	* id * * (id * id)	match
id +	B A'$	* id * * (id * id)	④
id +	* C A'$	* id * * (id * id)	⑦
id + *	C A'$	id * * (id * id)	match
id + *	id A'$	id * * (id * id)	⑩
id + * id	A'$	* * (id * id)	match
id + * id	* B A'$	* * (id * id)	⑤
id + * id *	B A'$	* (id * id)	match

id + * id *	* C A'$	* (id * id)	⑦
id + * id * *	C A'$	(id * id)	match
id + * id * *	(S) A'$	(id * id)	⑨
id + * id * * (	S) A'$	id * id)	match
id + * id * * (	A S') A'$	id * id)	①
id + * id * * (	B A' S') A'$	id * id)	④
id + * id * * (	id A' S') A'$	id * id)	⑧
id + * id * * (id	A' S') A'$	* id)	match
id + * id * * (id	* B A' S') A'$	* id)	⑤
id + * id * * (id *	B A' S') A'$	id)	match
id + * id * * (id *	id A' S') A'$	id)	⑧
id + * id * * (id * id	A' S') A'$	)	match
id + * id * * (id * id	S') A'$	)	⑥
id + * id * * (id * id	) A'$	)	③
id + * id * * (id * id)	A'$		match
id + * id * * (id * id)	$		⑥

b. No. * id * + (id + * id)

Que 5 marking scheme is updated:

1. 1.5 points for any justification
2. 1 mark for writing a grammar (even if it doesn't use a new token), 2 for adding a new token, 1.5 marks for explanation why new token is necessary
3. 3 marks (marked according to the grammar they provided in part 2)
4. 3 marks
5. 1.5
6. 1.5

Part A) Should the minus sign/negative numbers be included in number literals in a language with precedence like C?
There are two possible answers:

1. Yes, as a minus sign followed by a number literal should always be parsed as a negative number
2. No, as the parser already has a rule for TOK_MINUS expression, and a number literal is already an expression

Part B) Suppose the language in the lab also supports prefix increment/decrement operators, i.e. ++x and --x.
What grammar rules (ignoring conflicts with other rules) need to be introduced to support these prefix operators?
Do any new tokens need to be defined (why or why not)?
New tokens must be introduced for **++** and **--**.
The grammar rule should be
expression := TOK_DOUBLE_PLUS expression | TOK_DOUBLE_MINUS expression
It does not work with the existing tokens as the grammar rule
expression := TOK_PLUS TOK_PLUS expression
would match **+ +x** (with a space)

Part C) Suppose number literals do not include the minus sign, and the language supports prefix operators as in Part B.
How would the expression --3 get lexed and parsed?
The expression **--3** would get parsed as **TOK_DOUBLE_MINUS TOK_INTEGER** as the lexer greedily matches the longest token it can first find, which would be **--**.
After that, it would match the **3**.

Part D) Suppose number literals do include the minus sign, and the language supports prefix operators as in Part B.
How would the expression --3 get lexed and parsed?
It gets parsed the same was as in Part C, for the same reason.

Part E) Suppose number literals do not include the minus sign. For our lab grammar, how does the expression x -4 get lexed and parsed?
It gets lexed as an identifier "x", the operator "-", and the number "4". It gets parsed as "x subtract 4"

Part F) Suppose number literals do include the minus sign. For our lab grammar, how does the expression x -4 get lexed and parsed?
It gets lexed as an identifier "x", and the number "-4". It is a syntax error