University of Asia Pacific

Admit Card

Mid-Term Examination of Fall, 2020

Financial Clearance	PAID

Registration No : 17101086

Student Name : Md. Remon Hasan Apu

Program : Bachelor of Science in Computer Science and Engineering

SI.NO.	COURSE CODE	COURSE TITLE	CR.HR.	EXAM. SCHEDULE
1	CSE 425	Computer Graphics	3.00	
2	CSE 426	Computer Graphics Lab	1.50	
3	CSE 429	Compiler Design	3.00	
4	CSE 430	Compiler Design Lab	1.50	
5	BUS 401	Business and Entrepreneurship	3.00	
6	BUS 402	Business and Entrepreneurship Lab	0.75	
7	CSE 457	Design and Testing of VLSI	3.00	
8	CSE 458	Design and Testing of VLSI Lab	0.75	
9	CSE 400	Project / Thesis	3.00	

Total Credit: 19.50

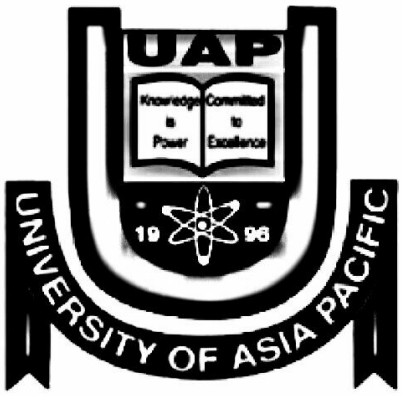
1. Examinees are not allowed to enter the examination hall after 30 minutes of commencement of examination for mid semester examinations and 60 minutes for semester final examinations.

2. No examinees shall be allowed to submit their answer scripts before 50% of the allocated time of examination has elapsed.

3. No examinees would be allowed to go to washroom within the first 60 minutes of final examinations.

4. No student will be allowed to carry any books, bags, extra paper or cellular phone or objectionable items/incriminating paper in the examination hall.
Violators will be subjects to disciplinary action.

This is a system generated Admit Card. No signature is required.



University of Asia Pacific

Department of Computer Science and Engineering

MID SEMESTER EXAM-FALL 2020

Course Name : Compiler Design

Course Code : CSE-429

Semester: 4th Year 2nd Semester

SUBMITTED By

Md. Remon Hasan Apu

ID: 17101086, Section: B

Ans: to the Q: No - 01

a)

regular expression:

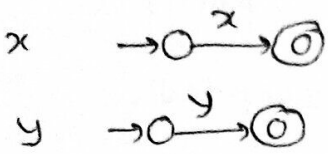
i) $(x+y)^* xyy (x+y)^*$

3

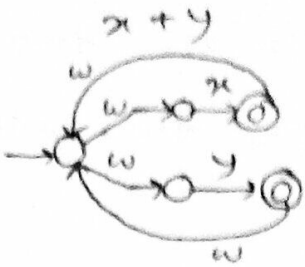
ii) regular expression to NFA

$(x+y)^* xyy (x+y)^*$

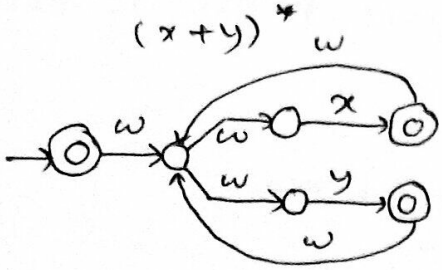
step 01:



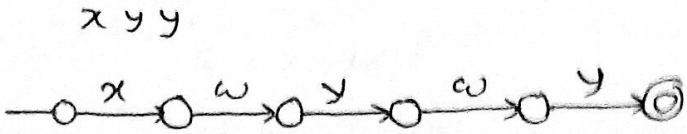
s-02:



s-03:

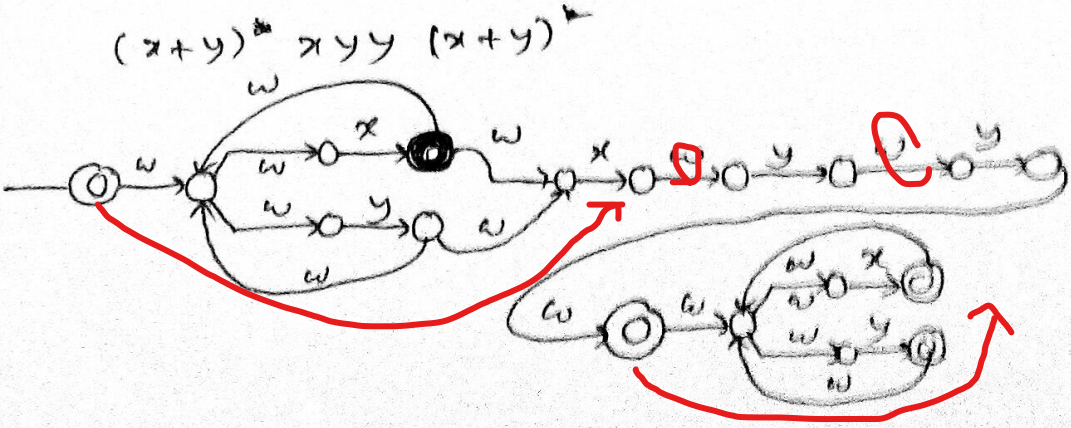


s-04:



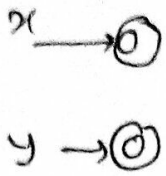
s-05

5



iii)

Here, From (ii) we get the final state as



3

The given string is xyyxyy. The given string is accepted by the above NFA. because we can contains the x, then two consicutive yy, then x and the final state is reached at y where two yy is possible.

so, this string is accepted.

b)

The laxical analysis phase is responsible for gerenating tokens.

Example: x = a + b * c ;

↓

laxical Analysis

↓

Tokens = 8

id = id + id * id ;

Token	lexemes	pattern
Parenthesis	(), {}, []	Parenthesis
tok if	if	if
tok while	while	while
integer literal	4, 5, 2	digit followed by zero or more digit
operator	!=, ==, +, %, &&	all the operator
keyword	int, printf, %d,	any build in function from given language
Separator	,, ;, ""	all of the separator

Ans: to the Q: NO – 02

a) In lexical analyis the token is generated, on the other hand parse trees are generated in syntax analysis by belonging any grammer.

Difference between lexical analysis and syntax analysis

Lexical Analysis	Syntax analysis
i) all the process of sequence of characters is form a sequence of token	i) process of analyzing a string of symbols either in NL or CL with formal grammer.
ii) it includes the lexing and tokenization.	ii) it includes syntatic analysis and parsing
iii) first phase of compilation process	iii) second phase of compiler process.

Mentioning the above 3 differences using 1 example:

Example: ~~id = id + id * id~~ x = a + b * c ;

Lexical analysis

Tokens = 8

id = id + id * id ;

↳ Tokenization

Syntax analysis

Consider Grammer,

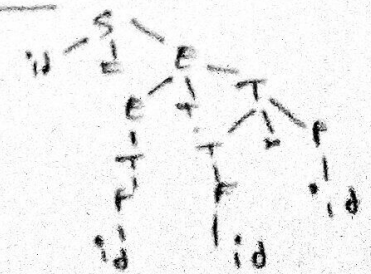
S → id = E

E → E + T / T

T → T * F / F

F → id

Parse tree



so we get,

id = id + id * id

8

b)

i) Regular expression,

Let,

letters = (A ... Z | a ... z)

underscore = (_)

dollar = ($)

The expression will be,

R_1 = ~~(A ... Z | a ... z)+ _ | -~

R_1 = ((A ... Z) | (a ... z) | (_))+ |
(0 ... 9)*
((A-Z) | (a ... z) | (_)* | ($))

3

R_2 = ((A ... Z) | (a ... z) | (_))+ |
($)* | ((A ... Z) | (a ... z) | (_))*
(0 ... 9)*

9 10

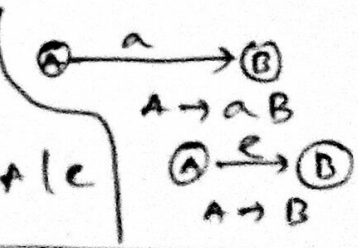
ii)

R_1 = ((A ... z) | (a ... z) | (_))+ |
(0 ... 9)* | (± 0.1 ... ± n)* | (± 1 ... ± n)* |
($)* | ((A ... z) | (a ... z) | (_))*

R_2 = ((A ... z) | (a ... z) | (_))+ |
($)* | (0 ... 9)* | (± 0.1 ... ± n)* |
(± 1 ... ± n)* | ((A ... z) | (a ... z) |
(_))*

0

c) limitations of regular expression over context free grammer:

Regular expression	context free grammer
i) capable of describing the syntax of tokens	i) all of each state there is a non-terminal symbol.
ii) syntactic construction is described by RE	ii) the state A has a ~~transit~~ transition to state B on a symbol a
iii) Example: (a|b)*abb	Example: S → aA|bA, A → aA| bA| 1A|ε 
iv) has proper procedure for LA and SA	iv) NO specific guideline for LA and SA

2

Ans: to the Q: NO – 04

a) $id = (86\%.2)$

$= 0$

i) $S \rightarrow F \mid DE \mid (S+F)$ — (I)

$F \rightarrow a \mid DB$ — (II)

$B \rightarrow cD \mid cDef \mid cDa \mid ef \mid \epsilon$ — (III)

$D \rightarrow abc \mid e \mid \omega$ — (IV)

Here in, There is no left recursion and ~~left factoring~~ but have left factoring in a_2. So removing left factoring,

$B \rightarrow \underline{cD}$

$B \rightarrow \underline{cD}ef$

$B \rightarrow \underline{cD}a$

$B \rightarrow ef$

$B \rightarrow \omega$

α = common prefix

Formula,

$A \rightarrow \alpha A'$

$A' \rightarrow \beta_1 \ldots$ rest

B

Now,

$B \rightarrow$ ~~cD ef~~ cDef $cDB' \mid ef \mid \epsilon$

$B' \rightarrow \omega \mid ef \mid a$

	first	follow
S → F | De | (S+F)	{a, c, w, (}	{$, +}
F → a | DB	{a, c, w, e}	{$, +,)}
B → cDef	{c}	{a, $, +,)}
B1 → w | efla	{w, e, a}	{·}
D → abc | e | ε	{a, c, ε}	{c}

first follow

(b)

++xyxy

E

⇒ E - xyxy

⇒ +E - xyxy

⇒

E

⇒ E - E

⇒ +E - E

⇒ +E - xyE

⇒ +

(b)

$+ E E$

$\Rightarrow + * E E$

$\Rightarrow + * - E E$

$\Rightarrow + * - x y E$

$\Rightarrow + * - x y x y$

Now,

$* E E$

$\Rightarrow * + E E E$

$\Rightarrow * + - E E E$

$\Rightarrow + + - x y E E$

$\Rightarrow * + - x y E E$

$\Rightarrow * + - x y x y$

5

So,

this grammar is not ambiguous