University of Asia Pacific

Department of CSE

Name: Rashik Rahman

Reg ID: 17201012

Year: 4th

Semester: 2nd

Course Code: CSE 429

Course Title: Compiler Design

Date: 25.11.2021

University of Asia Pacific

Admit Card

Final-Term Examination of Spring, 2021

Financial Clearance	PAID

Registration No : 17201012

Student Name : Rashik Rahman

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.

Admit Card Generation Time: 15-Nov-2021 01:33 PM

17201012
1

Answer to the Q.No. 1 (a)

TAC:

1. sum = 0
2. var1 = student1 < student2
3. if (var1) goto(5)
4. goto (8)
5. T1 = ~~sum~~ number +1
6. ~~sum = T1~~ number = T1
7. goto (10)
8. T2 = ~~sum~~ number -1
9. ~~sum~~ number = T2
10. T3 = sum +1
11. sum = T3
12. var2 = sum < 10
13. if(var2) goto (2)

Answer to the Q No. 1(b)

Quadruple:

Operation	Arg 1	Arg 2	Result
=	0		sum
<	student1	student2	var1
+	number	1	T_1
=	T_1		number
-	number	1	T_2
=	T_2		number
+	sum	1	T_3
=	T_3		sum
<	sum	10	var2

17201012

Triples:

~~Ope~~

Address	Operation	Arg1	Arg 2
0	=	0	
1	<	student1	student2
2	+	number	1
3	=	2	
4	-	number	1
5	=	4	
6	+	sum	1
7	=	6	
8	<	sum	10

Scanned with CamScanner

(4)

17201012

Answer to the Q. No. 1(e)

Reason behind identifying ~~the~~ leaders:

1. The first instruction of each routine is a leader
2. Any statement that is the target of a branch/goto is a leader
3. Any statement that is immediately follows a branch/goto ~~o~~ or a call instruction is a leader.

Identifying leader:

1. sum=0 ---------------. L1
2. var1 = student1 < student2 --- L2
3. if (var1) goto (5)
4. goto (8) ------------- L3
5. T_1 = number +1 ------------. L2, L3
6. number = T_1
7. goto(10)
8. T_2 = number -1 --------------. L3, L3
9. number = T_2
10. T_3 = sum +1 ------------- L2
11. sum = T_3
12. var2 = sum < 10
13. if (var2) goto (2)

172010012

Line 1 is L1 as it satisfies first reason for being a leader. Line 2,5,8,10 are leaders L2 as they satisfies second ~~con~~ reason for being a leader. Line 4,5,8 are leaders as they satisfy third reason for being a leader.

Answer to the Q.No. 1(d)

B1
1. sum = 0 ———— L1

B2
2. var1 = student1 < student2 ---- L2
3. if (var1) goto (5)

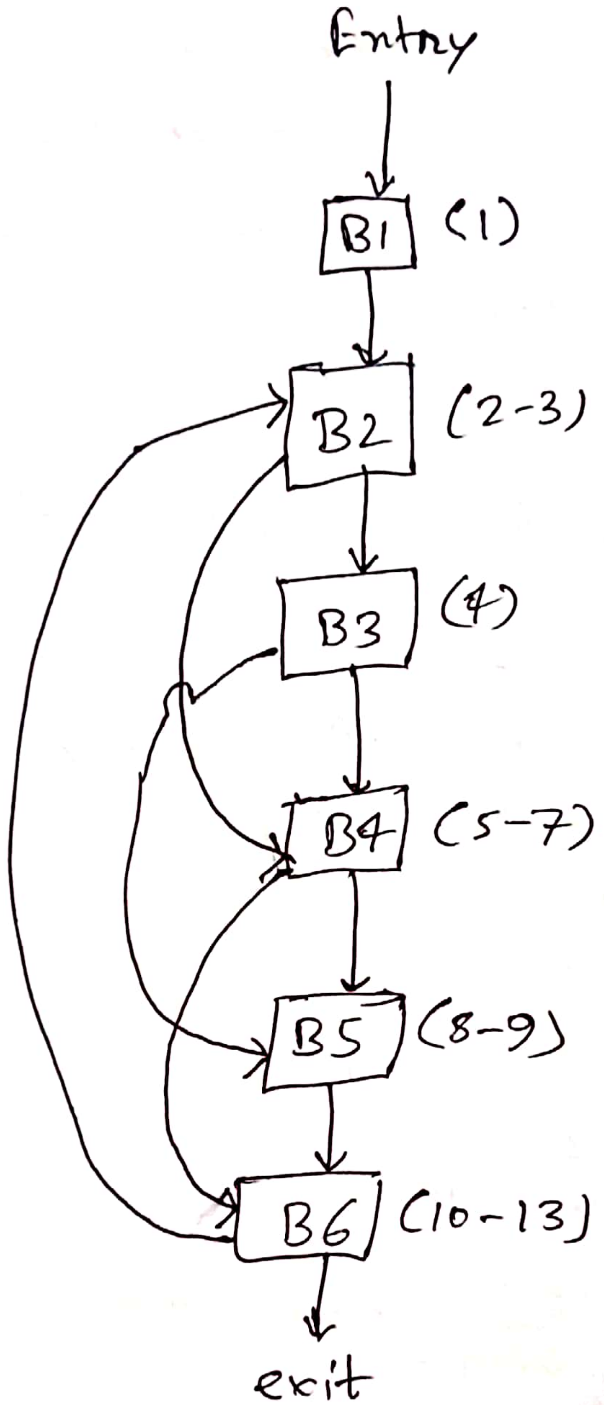
B3
4. goto (8) ———— L3

B4
5. T_1 = number + T_1 ———— L2, L3
6. number = T_1
7. goto (10)

B5
8. T_2 = number − 1 ———— L2, L3
9. number = T_2

B6
10. T_3 = sum+1 ———— L2
11. sum = T_3
12. var2 = sum<10
13. if (var2) goto (2)

Control flow graph:



Answer to the Q.No.2 (a)

Let,

A = REXPR
B = RTERM
C = RFACTOR
D = RPRIMARY

∴ ~~The~~ Now the grammer is,

0. $A \rightarrow A+B \mid B$
1. $B \rightarrow BCa \mid BCb \mid BC$
2. $C \rightarrow C^*D \mid D$
3. $D \rightarrow a \mid b$

(i) There's a ~~repeatetive occurance of ϕ~~ left factorin in ~~ϕ~~ production 1 as BC appear repeatively. So need to remove it by modifiying the production with followings

$B \rightarrow BCB'$
$B' \rightarrow a \mid b \mid \epsilon$

∴ RTERM → RTERM RFACTOR RTERM'

RTERM' → a | b | ε

Ans.

As
$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \gamma_1 \mid \gamma_2$
$\therefore A \rightarrow \alpha A' \mid \gamma_1 \mid \gamma_2$
$A' \rightarrow \beta_1 \mid \beta_2$

Here,
$B \rightarrow \underbrace{BC}_{\alpha}\underbrace{a}_{\beta_1} \mid \underbrace{BC}_{\alpha}\underbrace{b}_{\beta_2} \mid \underbrace{BC}_{\alpha}\underbrace{\epsilon}_{\beta_3}$

8

17201012

(ii) If more than one grammar production rules has a common prefix string, then the top-down parser cannot make a choice as to which of the production it should take to parse the string. To remove this confusion we use a technique named left factoring. It transforms the grammar to make it useful for top down parsers. Here we make one production for each common prefix and the rest of the derivation is added by new productions.

Ans.

(iii)

Here production 0, 1, 2 has ~~left re~~ immediate left recursion. We can do the following to remove it.

$A \Rightarrow B A'$

$\Rightarrow$ REXPR $\Rightarrow$ RTERM REXPR'

$A' \Rightarrow + B A'$

~~REXPR' = RTERM~~

$\Rightarrow$ REXPR' $\Rightarrow$ + RTERM REXPR' | ϵ

$A \rightarrow \underset{A}{\underline{A}} \underset{\alpha}{\underline{+B}} \mid \underset{\beta}{\underline{B}}$

$A = B A'$

$A' = +BA' \mid \epsilon$

Scanned with CamScanner

$\therefore C \to DC'$

$\Rightarrow$ RFACTOR $\to$ RPRIMARY RFACTOR'

$C \to * DC' | \epsilon$

$\Rightarrow$ RFACTOR $\to$ * RPRIMARY RFACTOR | ϵ

$\underbrace{C}_{A} \to \underbrace{C}_{A} \underbrace{*D}_{\alpha} | \underbrace{D}_{\beta}$

$\Rightarrow C \to DC'$

$C' \to * DC' | \epsilon$

$\therefore B \to B'$

$\Rightarrow$ RTERM $\to$ RTERM'

$B' \to CaB' | CbB' | CB' | \epsilon$

$\Rightarrow$ RTERM' $\to$ RFACTOR a RTERM' | RFACTOR b RTERM' | RFACTOR RTERM' | ϵ

$\underbrace{B}_{A} \to \underbrace{B}_{A}\underbrace{Ca}_{\alpha_1} | \underbrace{B}_{A}\underbrace{Cb}_{\alpha_2} | \underbrace{B}_{A}\underbrace{C}_{\alpha_3}$

$B \to B'$

$B' \to CaB' | CbB' | CB' | \epsilon$

(iv) A grammer becomes left recursive ~~by~~ if it has any non-terminal 'A' whose derivation contains 'A' itself as the left-most symbol. Top-down parser starts parsing from the start symbol which in itself is non-terminal. So when the parser encounters the same non-terminal in its derivation, it becomes hard for it to judge when to stop parsing the left non-terminal and it goes into infinite loop. By resolving this with ~~a~~ left recursion technique the parser ~~wo~~ will not go into infinite loop and thus be able to parse successfully.

Answer to the Q.No.2(b)

Recursive decent parser:

Recursive decent parser is a kind of top-down parser. It builts the parse tree from the top to bottom, starting with the start non-terminal.

Given CFG:

$E \rightarrow Ax \mid Bw$

$A \rightarrow xxC \mid xxyx$

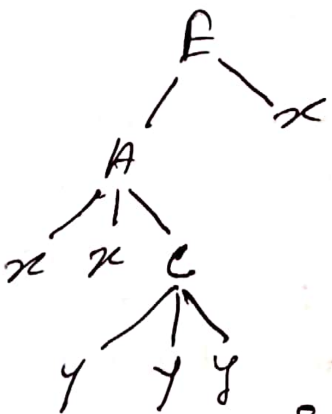
$C \rightarrow yyy$

$B \rightarrow xxD$

$D \rightarrow yyz$

$E \rightarrow Ax$

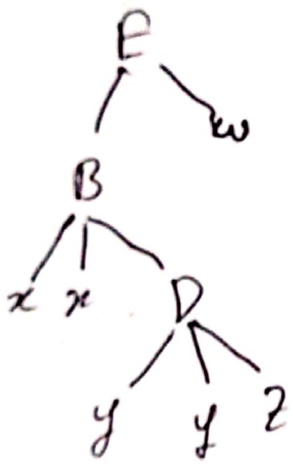
$A \Rightarrow$



→ failure so backtraking

17201012

failure backtracking



output: xxyyzw.

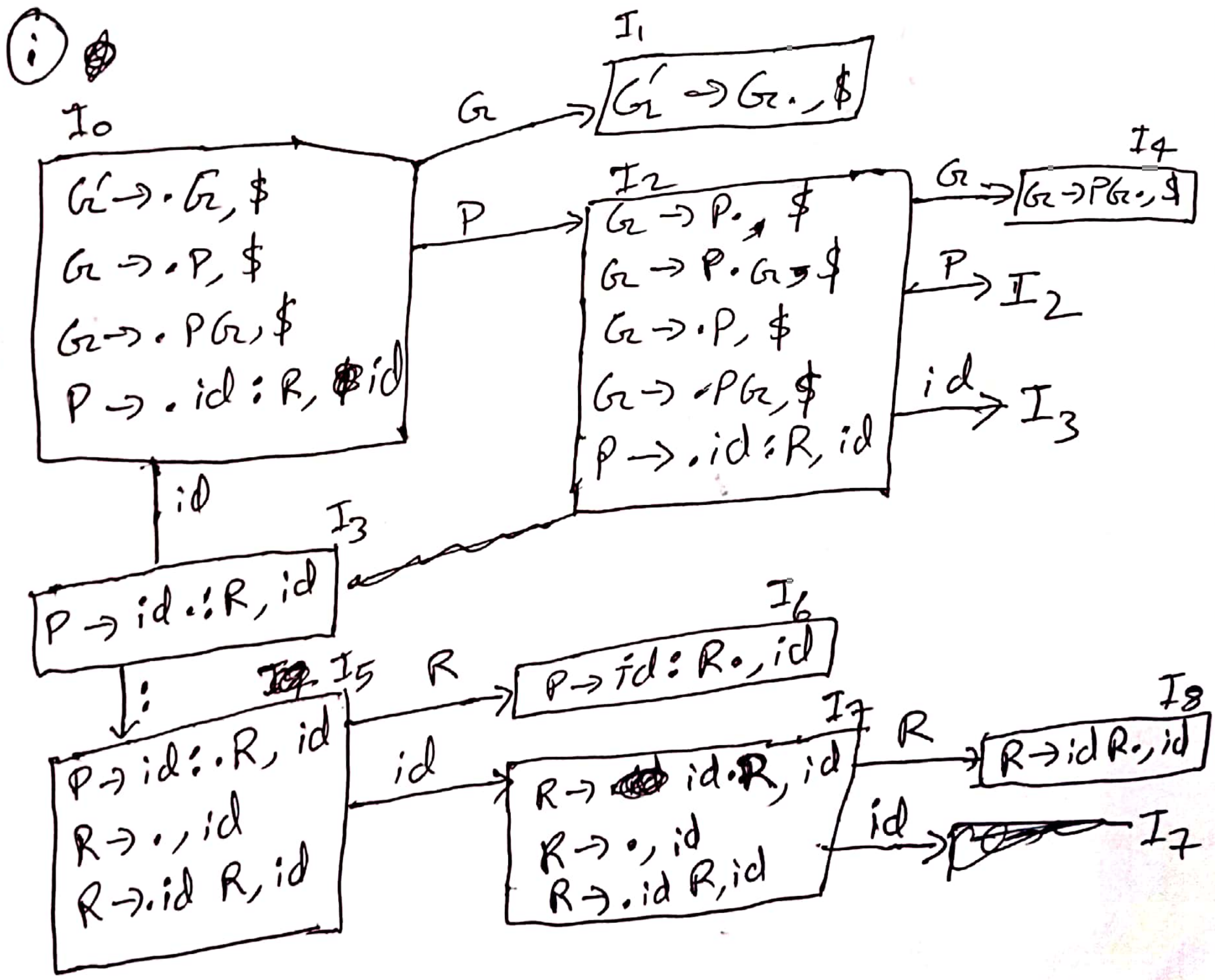
(11)
17201012

Answer to the Q. No. 4(OR)

Given,

	first	follow
$G \to P \mid PG$	{ id }	{ $ }
$P \to id : R$	{ id }	{ $, id }
$R \to \epsilon \mid id\ R$	{ id, ϵ }	{ $, id }

(i)



172010142

(ii)

0. $G' \rightarrow G$

1. $G \rightarrow P$

2. $G \rightarrow PG$

3. $P \rightarrow id : R$

4. $R \rightarrow \epsilon$

5. $R \rightarrow id\ R$

	Action			Goto		
state	:	id	\$	G	P	R
0		S_3		I_1	I_2	
1			Acc			
2		S_3		I_4	I_2	
3	S_5					
4			R_2			
5		S_7				I_6
6		R_3				
7		S_7				I_8
8		R_5				

CLR parse table.

iii

The parse table will parse string because there are no ~~sut~~ conflicts like S/S, S/R, R/S or R/R in the parse ~~to~~ table. So ~~there will be~~ the table will parse the string.

Answer to the Q.No. 3

(i)

first(S) = {if, while, print, id, int}

first(E) = {id, int}

first(T') = {else}

first(T) = {if, while, print, id, int}

first(S'') = {then}

first(S') = {id, int}

first(P) = {if, while, print, id, int,}

first(P') = {ϵ, if, while, print, id, int}

follow(E) = {then, if, while, print, id, int, else}

follow(S) = {if, while, print, id, int, else}

follow(S') = {if, while, print, id, int, else}

follow(S'') = {if, while, print, id, int, else}

follow(T) = {if, while, print, id, int, else}

follow(T') = {if, while, print, id, int, else}

follow(P) = {\$}

follow(P') = {\$}

15

17201012

(ii) LL(1)

	if	while	print	then	else	id	int	$
P	P->P'	P->P'	P->P'			P->P'	P->P'	
P'	P'->P	P'->P	P->P'			P'->P	P'->P	P'->ε
S	S->if S	S->while E S	S->Print E			S->E	S->E	

Scanned with CamScanner