Question No. 3

Part-1

(a)

First(S) → { c }
First(A) → { a, c }
First(B) → { b, c }

Fallow(S) ⇒ { $ }
Fallow(A) ⇒ { a, b, c }
Fallow(B) ⇒ { b, c }

(b)

LL(1)

	a	b	c
S			S → cABc
A	A → aAq		A → c
B		B → bBb	B → c

√ (1)

(c)

Grammar is LL(1), as No, Follow(A1) ∩ Follow(A2) ≠ ∅, also no ambiguity in

Part-II

$S \rightarrow I$
$I \rightarrow$ if (C) then A
$C \rightarrow$ id R num
$A \rightarrow$ id $=$ num ;
$R \rightarrow < \mid >$

(5)

Blue → Non-terminals
Black → terminals
} for ease

Question No. 4

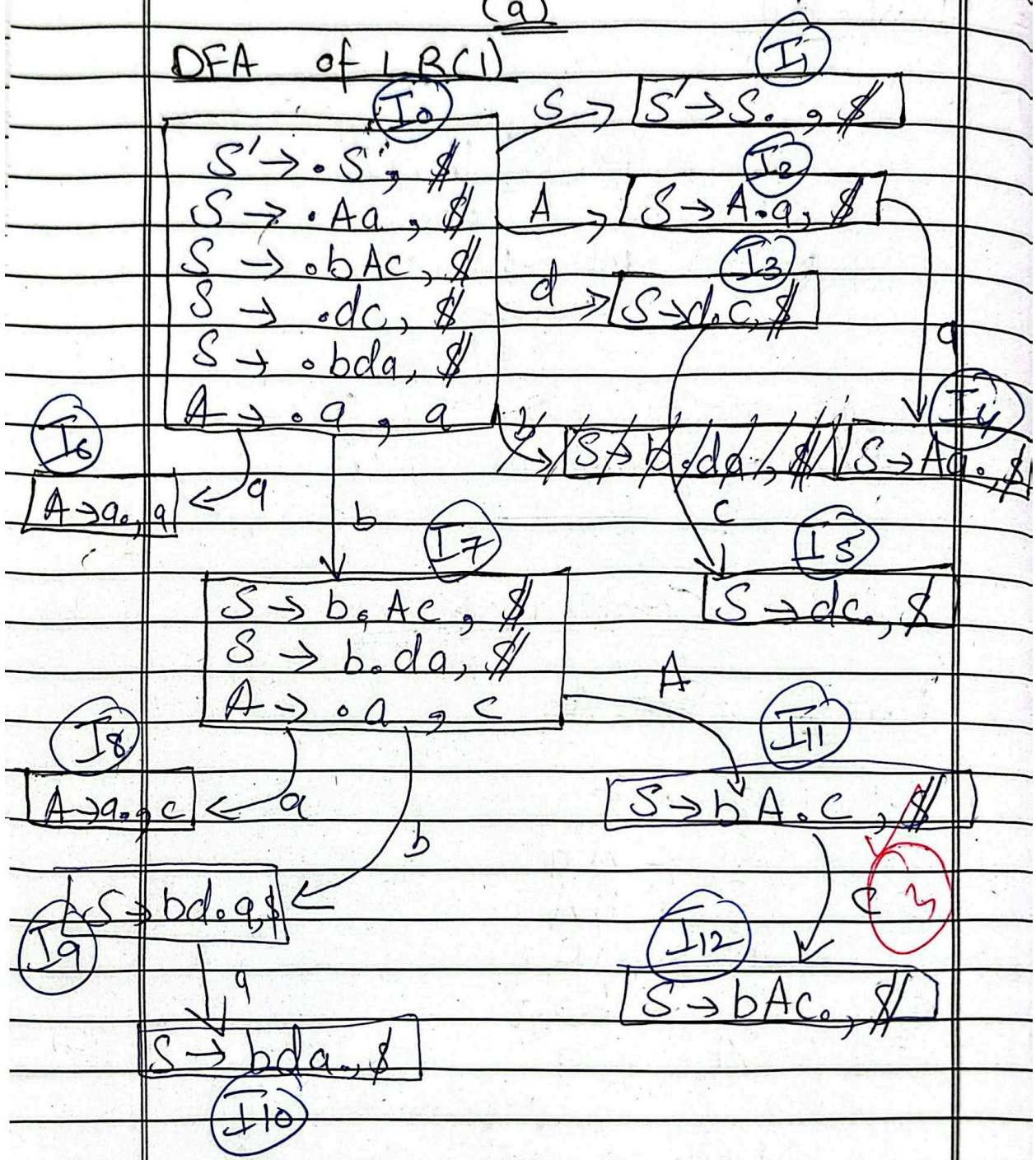
Part-I ~~$S \rightarrow Aq$~~

(1) $S' \rightarrow S$
(2) $S \rightarrow Aq$
(3) $S \rightarrow bAc$
(4) $S \rightarrow dc$
(5) $S \rightarrow bdq$
(6) $A \rightarrow q$

DFA on next page

(a)

DFA of LR(1)

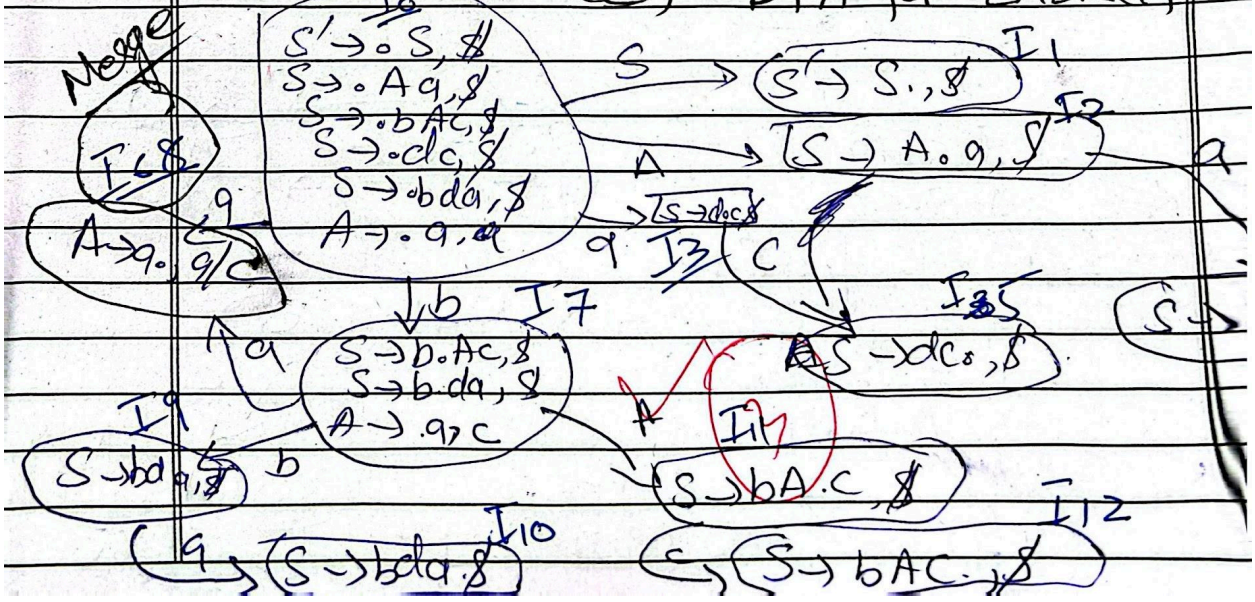

I0
S' → .S , $
S → .Aa , $
S → .bAc , $
S → .dc , $
S → .bda , $
A → .a , a
I1
S → S' → S. , $
I2
A → S → A.a , $
I3
d → S → d.c , $
I4
a → S → Aa. , $
I5
c → S → dc. , $
I6
a → A → a. , a
I7
b → S → b.Ac , $
S → b.da , $
A → .a , c
I8
a → A → a. , c
I9
b → S → bd.a , $
I10
a → S → bda. , $
I11
A → S → bA.c , $
I12
c → S → bAc. , $

(b)

Parse table for LR(1)

	a	b	c	d	$	S	Goto A
I_0	S6	S7		S3		1	2
I_1					accept/R1		
I_2	S4						
I_3			S5				
I_4					R2		
I_5					R4		
I_6	R6						
I_7	S8	S9					11
I_8			R6				
I_9	S10						
I_{10}					R5		
I_{11}			S12				
I_{12}					R3		

(c) DFA for LALR(1)

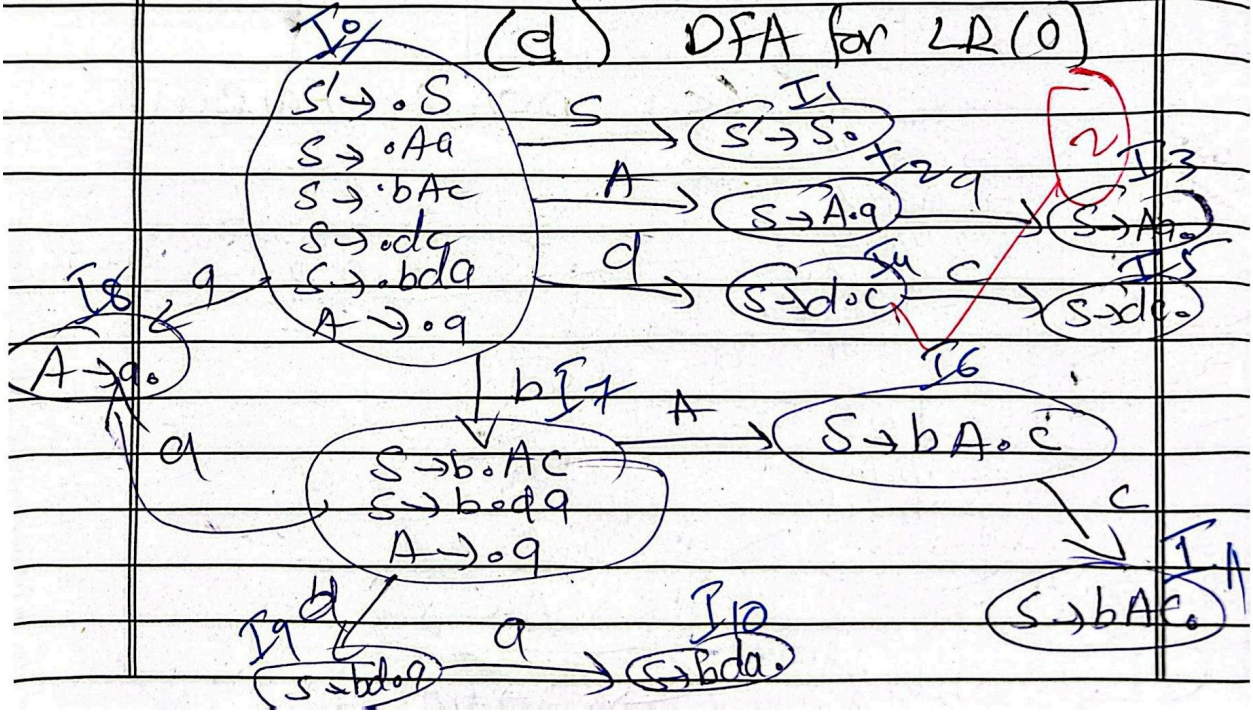


(c)

Parse table for LALR(1)

	a	b	c	d	S	A
I_0	S68	S7	S3		1	2
I_1				R1/accept		
I_2	S4					
I_3			S5			
I_4				R2		
I_5				R4		
I_{68}	R6		R6			
I_7	S68		S9			
I_9	S10		R6			
I_{10}	~~S10~~			R5		
I_{11}			S12			
I_{12}				R3		

(d) DFA for LR(0)



(e) SLR(1) Table

	Action					Goto	
	a	b	c	d	$	S	A
I_0	S8	S7		S4		1	2
I_1					R1/accept		
I_2	S3						
I_3					R2		
I_4		S5					
I_5					R4		
I_6		S11					
I_7	S8			S9			6
I_8	R6	R6					
I_9	S10						
I_{10}					R5		
I_{11}					R3		

Part-(f)

This grammar is also SLR(1) no conflicts

Part-II

When we merge I_5 & I_9 we get I_{59} as

I_{59}

$A \rightarrow d.,\ a/c$
$B \rightarrow d.,\ a/c$

Now, there is a Reduce-Reduce conflict in LALR(1)

bcz both R5 & R6 will happen at a/c
As

	a	c
I_{59}	R5/R6	R5/R6

(5)

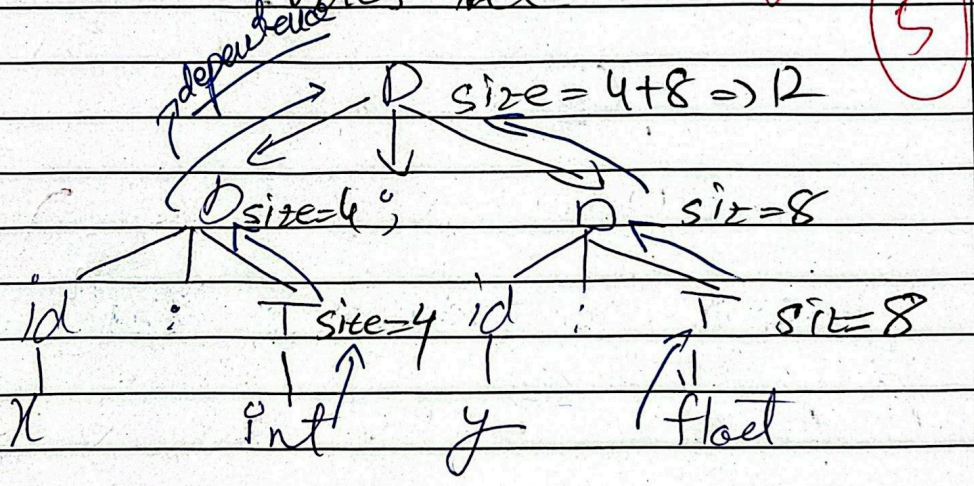
Because of reduce-reduce conflict it is not LALR(1)

Question No. 5

Part-1

Grammar Rule	Semantic Rule
$D_1 \rightarrow D_2 ; D_3$	D.size = D2.size + D3.size
$D \rightarrow id : T$	D.size = T.size
$T \rightarrow$ int	T.size = 4
$T \rightarrow$ float	T.size = 8

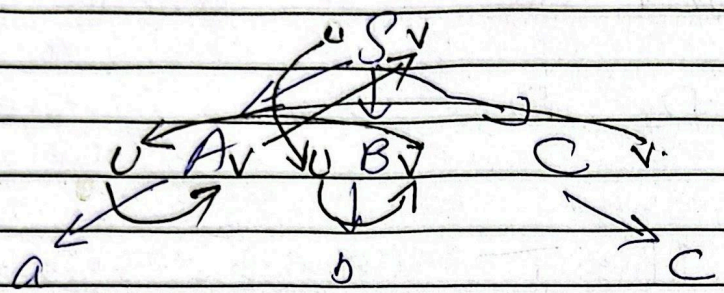
Parse Tree

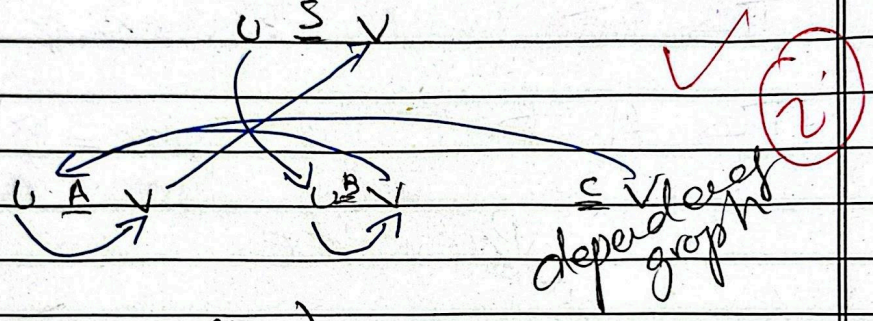


5

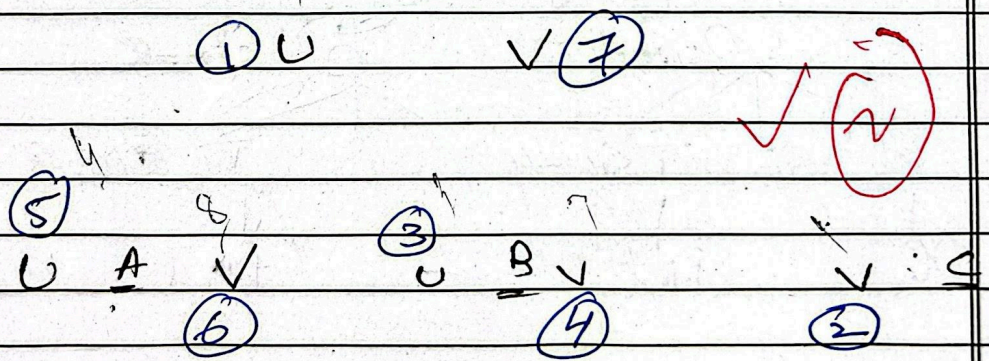
Part-II

(a) Parse Tree





(b)



Order of evaluation by Topological Sort

(c)

1. S.U → 3
2. C.V → 1
3. B.U ⟹ S.U → 3
4. B.V → B.U → 3
5. A.U → B.V + C.V → 3+1 → 4
6. A.V → 2 * A.U ⟹ 8
7. S.V ⟹ A.V ⟹ (8) find value

(d)

2

1. S.U → 3
2. B.U → S.U → 3
3. B.V = B.U → 3

2

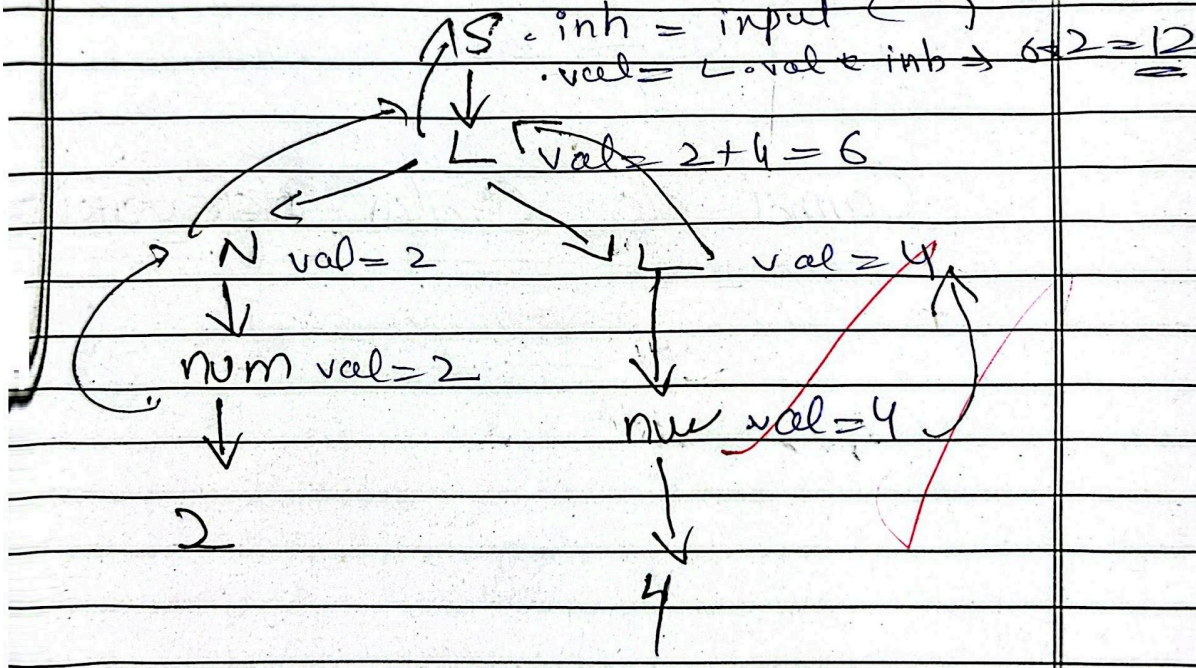
Cannot do Circular dependencies

↓

because of

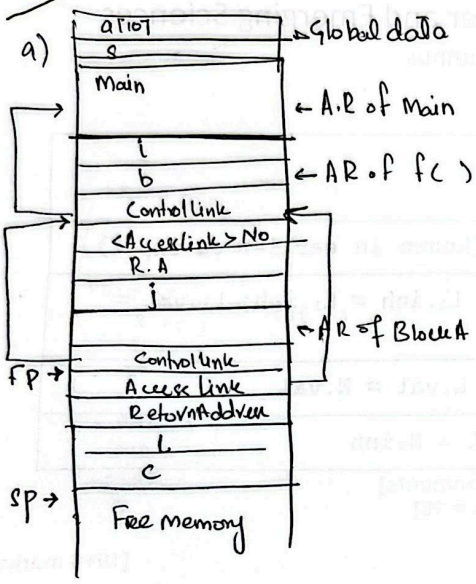
Part - III

Grammar Rule	Semantic Rule
$S \rightarrow L$	$S.val = L.val * S.inh$ S ↳ input
$L1 \rightarrow N, L2$	$L1.val = N1.val + L2.val$
$L \rightarrow N$	$L.val = N.val$
$N \rightarrow num$	$N.val = num.val$

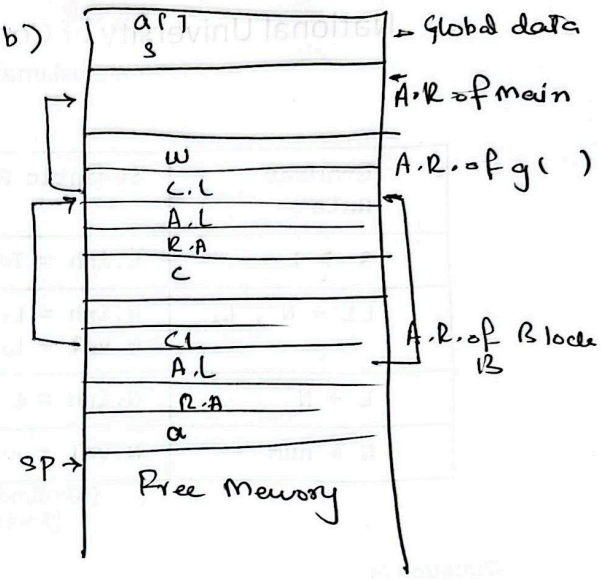


Q NO 6

a)


a[10]
s
Main
t
b
Control Link
<Access link> No
R.A
j
Control Link
Access Link
Return Address
L
c
fp →
sp →
Free memory
← Global data
← A.R of Main
← AR of f( )
← AR of Block A

b)


a[]
s
w
C.L
A.L
R.A
c
C1
A.L
R.A
a
sp →
Free Memory
← Global data
← A.R of main
A.R. of g( )
A.R. of Block B1

ii. Use the following code and give output of the following program using:
   a) Pass by value
   b) Pass by reference
   c) Pass by Value-Result
   d) Pass by Name

```
#include <iostream>
using namespace std;

int i = 0;

void p(int x, int y) {
    x += 1;
    i += 1;
    y += 1;
}
int main() {
    int a[2] = {1, 1};
    p(a[i], a[i]);
    cout << a[0] << \n << a[1] << endl;
    return 0;
}
```

a) 1
1

b) 3
1

c) 2
1

d) 2
2

[Three address Code Generation]
[Total Marks = 15]

Question 7: **[5+8+2 marks]**

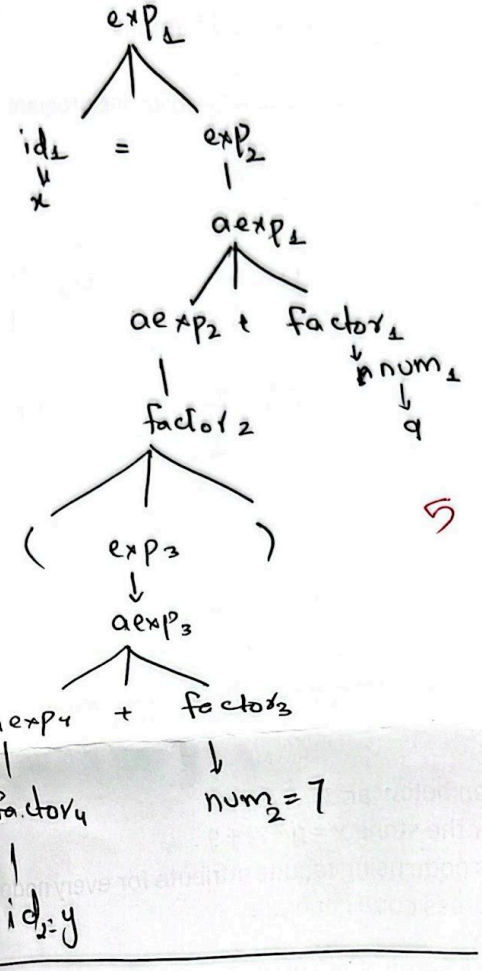
Use the attribute grammar given below and

a. construct parse tree for the string ***x = (y+7) + 9***
b. compute three address code using tacode attribute for every node (non-terminal) in the tree.
c. write the final three address code generate

Grammar Rule	Semantic Rules
$exp_1 \rightarrow id = exp_2$	exp_1.name = exp_2.name exp_1.tacode = exp_2.tacode ++ id.strval || ":=" || exp_2.name
exp → aexp	exp.name = aexp.name exp.tacode = aexp.tacode
$aexp_1 \rightarrow aexp_2$ + factor	$aexp_1$.name = newtemp() $aexp_1$.tacode = $aexp_2$.tacode ++ factor.tacode ++ $aexp_1$.name || ":=" || $aexp_2$.name || "+" || factor.name
aexp → factor	aexp.name = factor.name aexp.tacode = factor.tacode
factor → (exp)	factor.name = exp.name factor.tacode = exp.tacode
factor → num	factor.name = num.strval factor.tacode = " "
factor → id	factor.name = id.strval factor.tacode = " "

Q No 7

a) $x = (y + 7) + 9$



5

b)

$factor_4.name = id_2.strval = y$

$factor_4.tacode = " "$

8

$aexp_4.name = factor_4.name = y$

$aexp_4.tacode = factor_4.tacode = " "$

$factor_3.name = num_2.strval = 7$

$factor_3.tacode = " "$

$aexp_3.name = newtemp() = t1$

$aexp_3.tacode = aexp_4.tacode \; ++ \; factor_3.tacode \; ++ \; "t1 = y + 7"$

$exp_3.name = t1$

$exp_3.tacode \Rightarrow t1 = y + 7$

$factor_2.name = t1$

$factor_2.tacode \Rightarrow t1 = y + 7$

$aexp_2.name = t1$

$aexp_2.tacode \Rightarrow t1 = y + 7$

$factor_1.name = num_1.strval = 9$

$factor_1.tacode = " "$

$aexp_1.name = new\ temp() = t2$

$aexp_1.tacode = aexp_2.tacode\ ++\ factor_1.tacode\ ++$

$aexp_1.name = aexp_2.name + factor_1.name$

$\Rightarrow t1 = y + 7$

$t2 = t1 + 9$

$exp_2.name = t2$

$exp_2.tacode \Rightarrow t1 = y + 7$

$t2 = t1 + 9$

$exp_1.name = exp_2.name = t2$

$exp_1.tacode = exp_2.code\ ++$

$id_1.strval = exp_2.name$

$\Rightarrow t1 = y + 7$

$t2 = t1 + 9$

$x = t2$

---

c)

$$t1 = y + 7$$
$$t2 = t1 + 9$$
$$x = t2$$

2

National University of Computer and Emerging Sciences

Islamabad Campus

[Code Optimization]
[Total Marks = 9]

Question 8: **[3+3+3 marks]**

Consider the following code and

a. Apply constant folding and constant propagation and give the result.
b. Apply algebraic simplification to the result of (a) and give the result.
c. Apply strength reduction to the result of (b) and give the result.

```
int foo(int n) {
    int x, y, z, a, b, c;
    x = 23;
    y = 42;
    z = 69;

    a = x;
    a = a + 4 * (x + y) * n;
    c = n;

    if (a > 10) {
        b = a * 1 + x * (a + 0 + a);
        c = x + z;
    }

    a = b * 3;
}
```

a) a=23, a = 23+4*(23+42)*n = 283*n

b) b = a*1 + 23*(a+0+a) ⇒ b = a*1 + 23*(a+a)

c = 23+69 = 92

a = b*3 ⇒ a = b+b+b || (b<<1)

[Definition of Language using CFG]
[Total Marks = 8]

Question 9: **[8 marks]**

In C++, functions are essential constructs that encapsulate logic and enable code reuse. Your task is to write a **Context-Free Grammar (CFG)** that captures the syntax of C++ function definitions and function calls.

Your grammar must support the following features:

1. Function Definitions

Each function definition must include:

- A return type (e.g., int, void, double, char)
- A function name (identifier)
- A parameter list (which may be empty)
- A body enclosed in {} with a list of statements (you may use a placeholder like <stmt_list>)

2. Function Calls

A function call consists of a function name followed by arguments in parentheses:

- Examples: greet(), add(3, 4), compute(x + 2, y)

Arguments can be:

- Literal values (e.g., numbers like 3, 5.6)

- Identifiers (e.g., x, y)
- Expressions (e.g., a + b, x * (y + 2))
- Nested function calls (e.g., add(square(x), 4))

3. Parameter Lists
- A parameter list can contain zero or more parameters
- Each parameter includes a type and an identifier
- Example: int a, double val

4. Function Nesting
- Function calls can appear inside the body of another function, including inside expressions or return statements.

Q9 ----------GOOD LUCK----------

FN_DEF → FN_HEADER FN_BODY

FN_HDR → Rettype id (F_PARAMLIST)

Rettype → ε | int | void | char | double | ....

F_PARAMLIST → ε | PARAM_LIST

PARAM_LIST → PARAM, PARAM_LIST | PARAM

PARAM → Rettype id

FN_BODY → { stmt_list }

stmt_list → stmt stmt_list | ε

stmt → FN_CALL | others

FN_CALL → Id (arg_list)

arg_list → ε | args

args → expr , args | expr

expr → id | num | FN_CALL

Part A)

[Total Marks = 10]

Question 2:

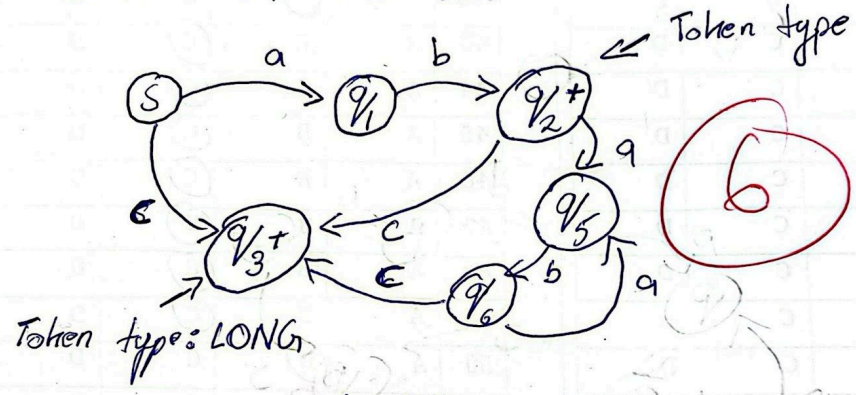
Consider a language over the alphabet {a, b, c} with two token types defined by regular expressions as follows:

1. Token type SHORT with regular expression ab
2. Token type LONG with regular expression (ab)*c

a) Draw one DFA that recognizes tokens of both types. Mark each accepting state with a double outline and annotate it with the corresponding token type. **(6 points)**

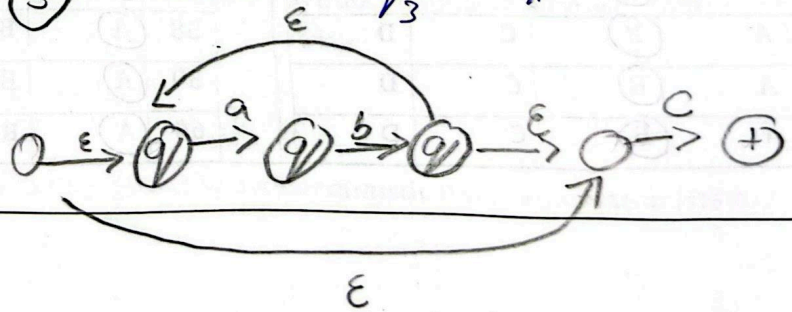
(You do not need to draw edges for invalid inputs: we assume that input characters that do not have an edge lead to a non-accepting invalid token state.)

↳ A combined DFA will be:



↳ Dry run on sample tokens:

- ab → it gives token: SHORT at q_2 final/accept state.
- c → gives token: LONG " q_3 accept state
- abc → " " LONG " b q_1 q_2 " "
- ababc → " S " LONG " q_1 q_3 " "



[Total Marks = 10]

Question 2:

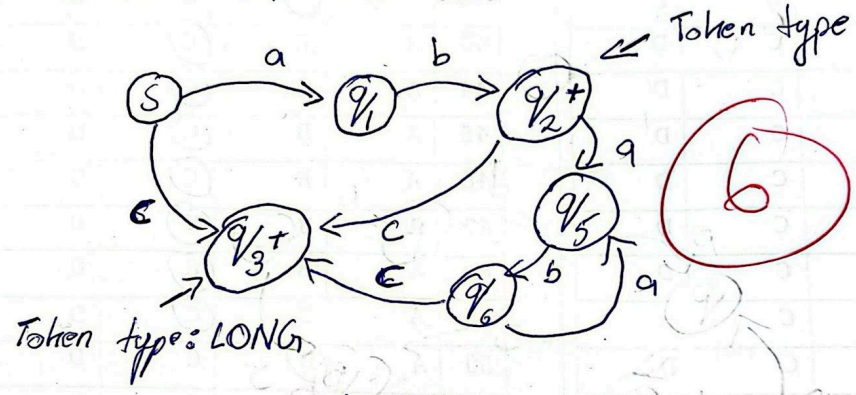
Consider a language over the alphabet {a, b, c} with two token types defined by regular expressions as follows:

1. Token type SHORT with regular expression ab
2. Token type LONG with regular expression (ab)*c

a) Draw one DFA that recognizes tokens of both types. Mark each accepting state with a double outline and annotate it with the corresponding token type. **(6 points)**

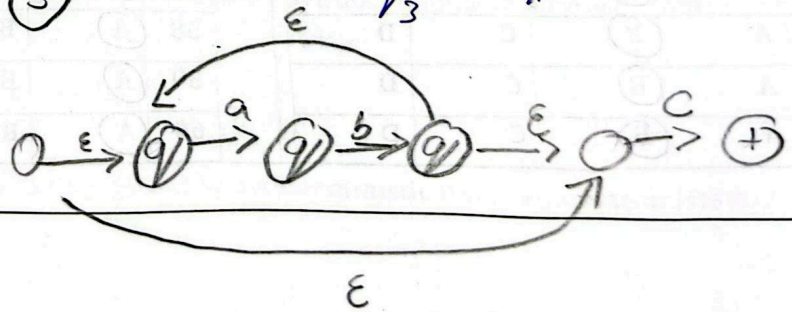
(You do not need to draw edges for invalid inputs: we assume that input characters that do not have an edge lead to a non-accepting invalid token state.)

↳ A combined DFA will be:



↳ Dry run on sample tokens:

- ab → it gives token: SHORT at q_2 final/accept state.
- c → gives token: LONG " q_3 accept state
- abc → " " LONG " b q_1 q_2 " "
- ababc → " S " LONG " q_1 q_3 " "



National University of Computer and Emerging Sciences 0042

Islamabad Campus

b) For each of the following inputs, state whether it is fully scanned successfully. If yes, what are the resulting tokens and their token types? **(3 points)**

Input String	Scanned	Resulting Tokens	Token Type
ababcab	Yes	ababc ; ab	LONG ; SHORT
cbabab	No		
ababab	Yes	ab ; ab ; ab	LONG ; LONG ; LONG

c) Provide name of one error recovery technique used by compilers in order to deal with lexical errors? **(1 point)** Panic Mode Recovery

ROUGH WORK