Binary Expression Trees:

- > Leaf nodes are operands
- > Root and other internal nodes are operators
- > Used for Expression evaluation, conversion.

Procedure for constructing the expression trees:

For Infix expression:

- > Scan the expression from left to right.
- > If an operand, create a node with operand as the info field and push it on to the stack of operands(Each item of the stack is a pointer to node)
- > If an operator, push it on to stack of operators based on precedence and associativity.
- > Whenever an operator is popped out of the operator stack, create a node w.r.t to the operator, pop out the top most two operands from operand stack. Add first popped out(operand 2) as the right child of the new node crated w.r.t to operator popped out of the stack and second popped out (operand 1) from the operand stack as the left chid. Push this new node on to operand stack.

For Postfix expression:

- > Scan the expression from left to right.
- > If an operand, create a node with operand as the info field and push it on to the stack of operands(Each item of the stack is a pointer to node)
- > If an operator, create a node w.r.t to the operator, pop out the top most two operands from operand stack. Add first popped out(operand 2) as the right child of the new node crated w.r.t to operator popped out of the stack

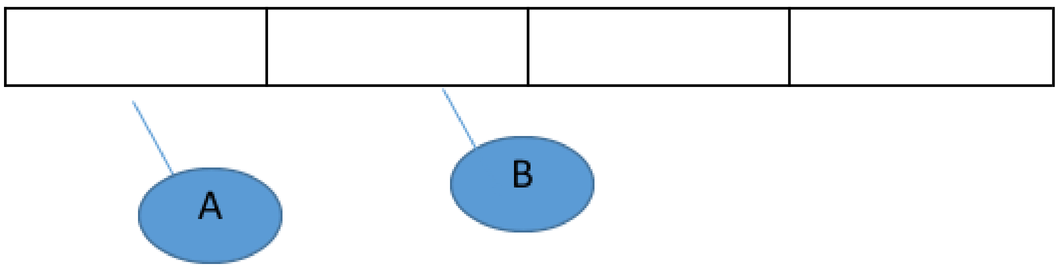
and second popped out (operand 1) from the operand stack as the left chid. Push this new node on to operand stack.

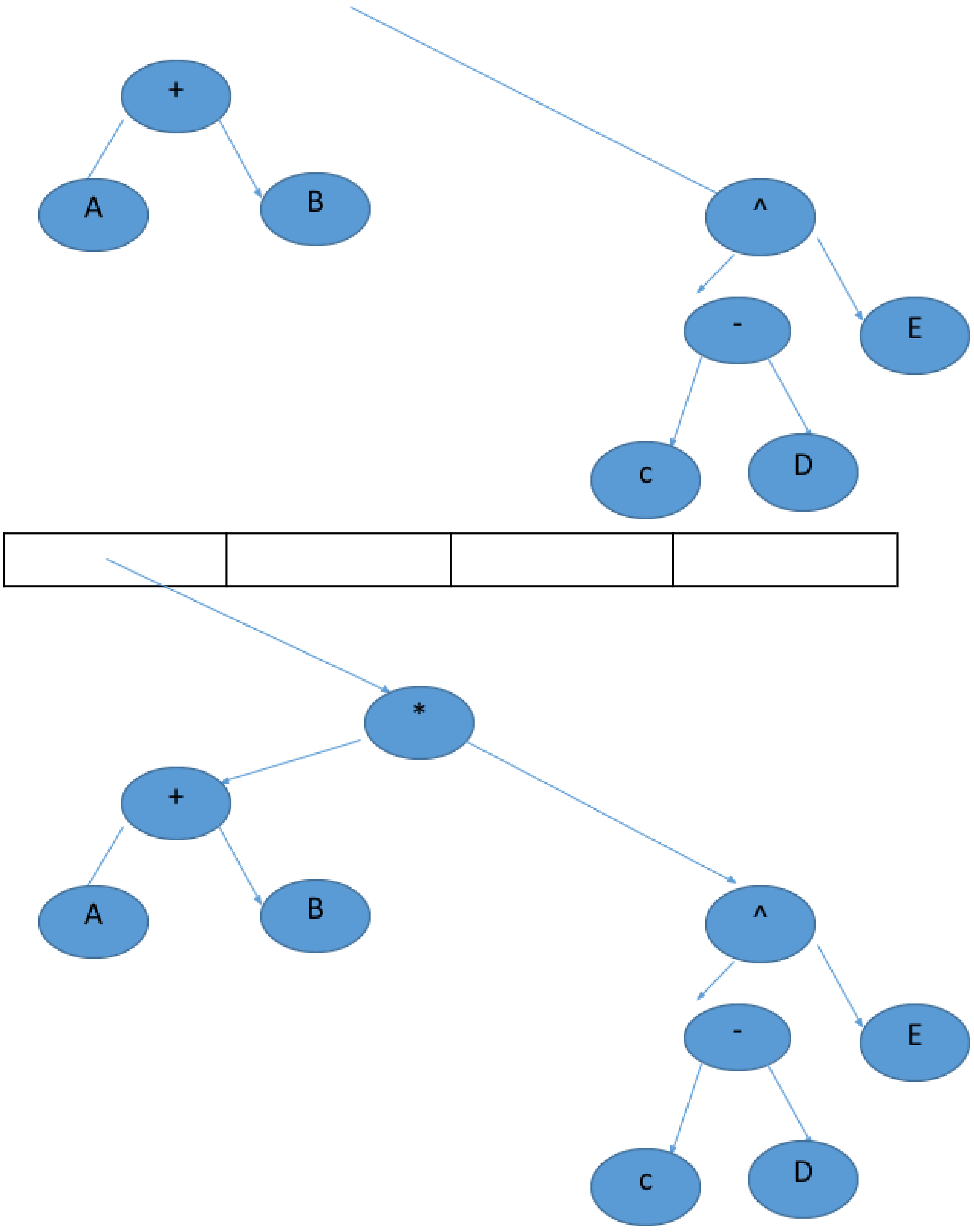
For Prefix expression:

> Scan the expression from right to left
> If an operand, create a node with operand as the info field and push it on to the stack of operands(Each item of the stack is a pointer to node)
> If an operator, create a node w.r.t to the operator, pop out the top most two operands from operand stack. Add first popped out(operand 1) as the left child of the new node crated w.r.t to operator popped out of the stack and second popped out (operand 2) from the operand stack as the right chid. Push this new node on to operand stack.

Expression tree construction for Infix Expression: (A+B)*(C-D)^E

(	(
A	(
+	(+
B	(+
)	
*	*
(	*(
C	*(
-	* (-
D	* (-
)	*
^	*^
E	*^





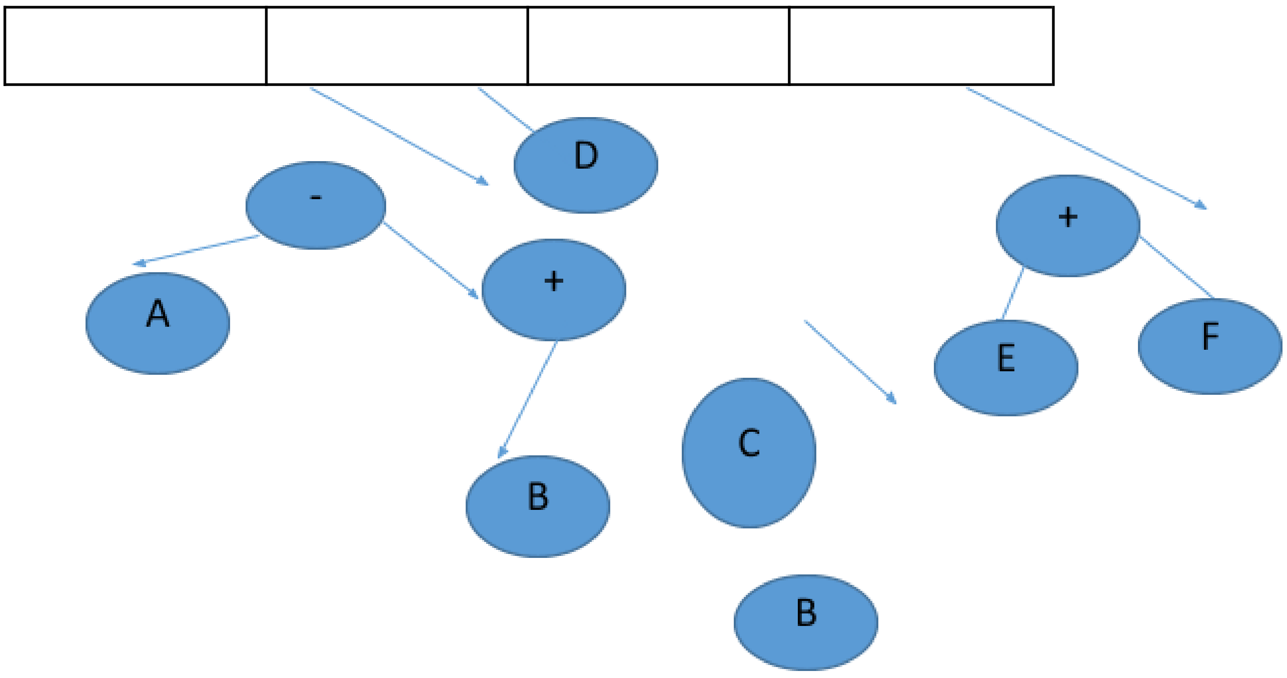
InOrder(A+B) *((C – D) ^ E) **PreOrder: *+AB^-CDE**

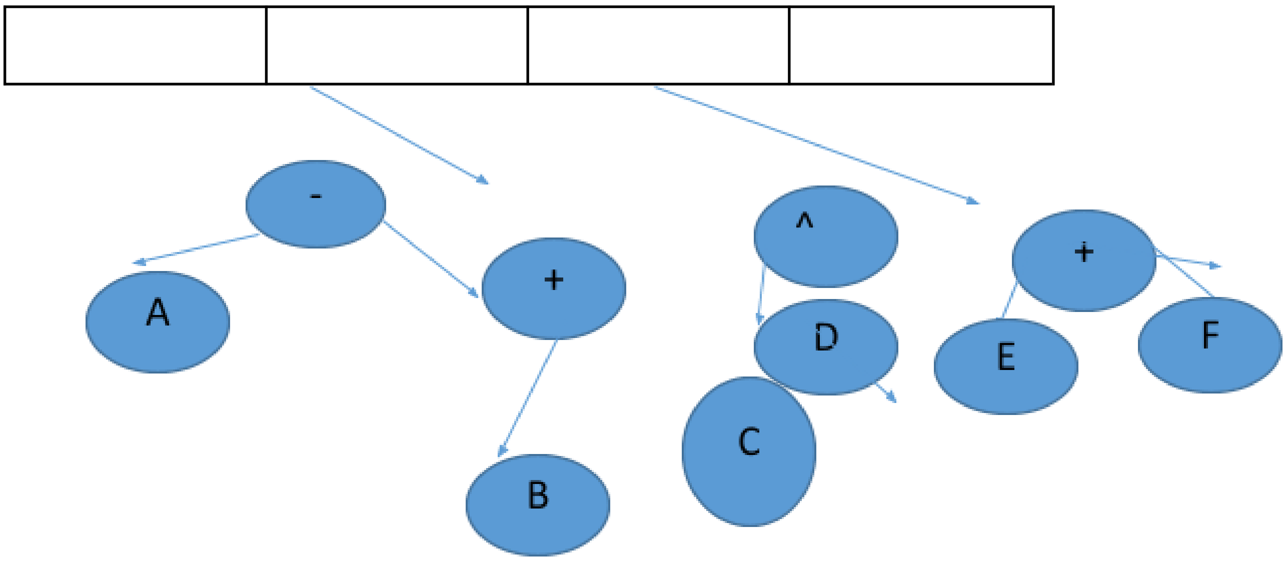
PostOrder: AB+CD-E^*

(A – (B + C)) * D ^ (E + F)

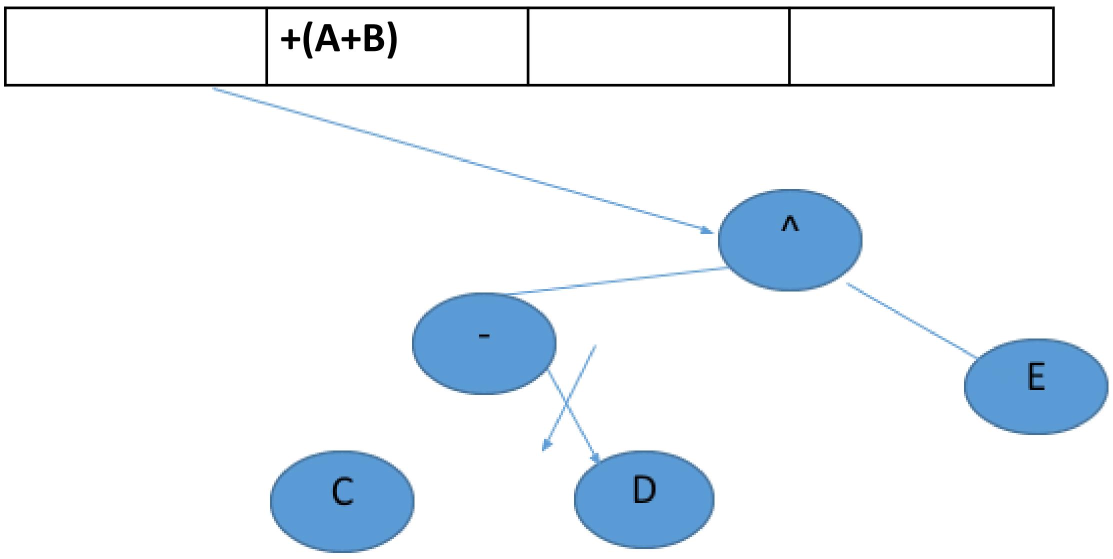
> ABC+- DEF+^*
> *-A+BC^D+EF
>

> ABC+- DEF+^*





***+AB^-CDE**



(2)

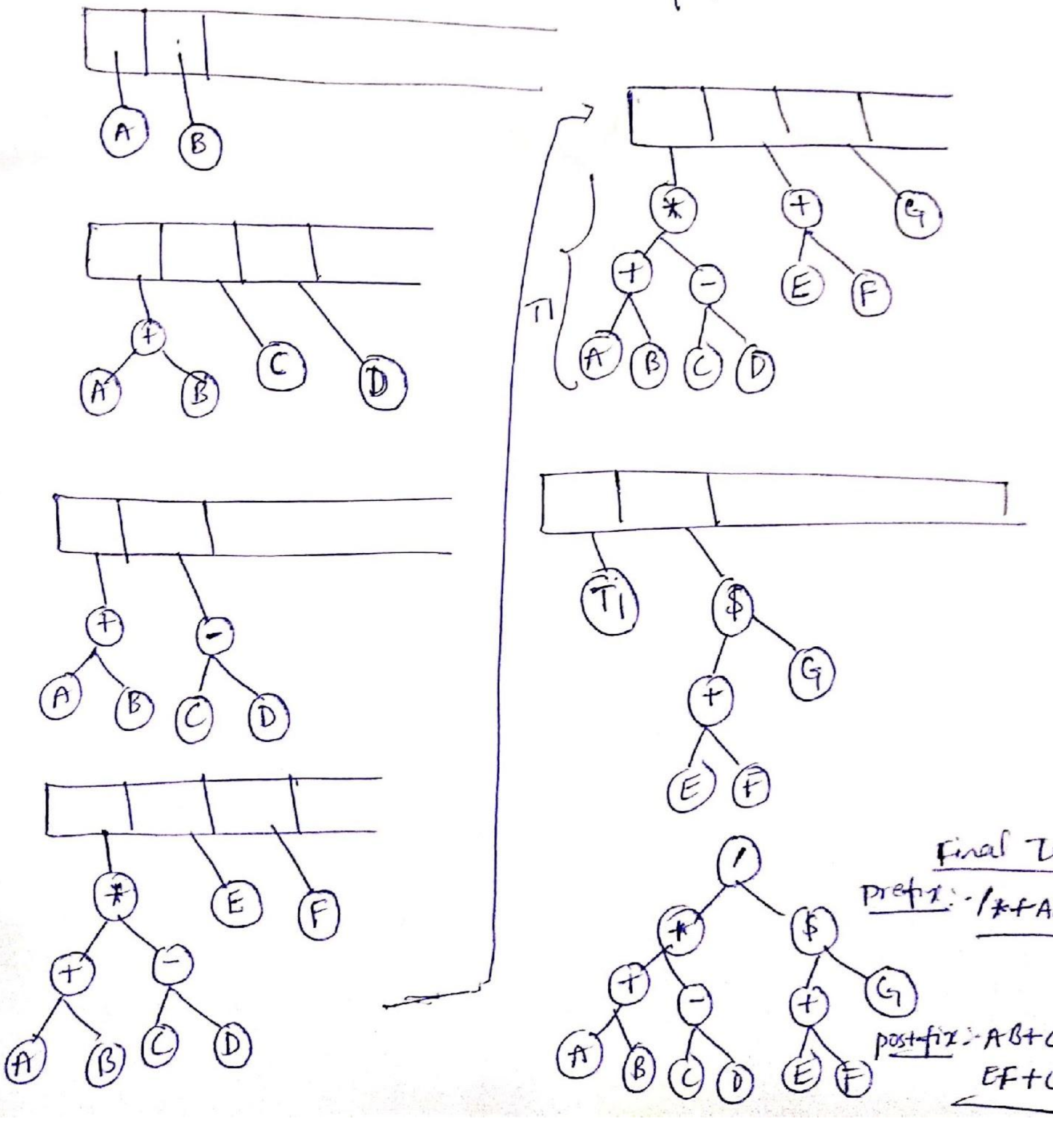
Construct the expression tree for the infix expression $(A+B)*(C-D)/(E+F)\$G$ and write the other two forms of the expression by traversing in the required order.

prefix :- $\boxed{/*+AB-CD\$+EFG}$

postfix :- $\boxed{AB+CD-*EF+G\$/}$

o/p	operator stack				
(	(	(	*(	(	/(
A	(	C	*(	E	/(
+	(+	-	*(-	+	/(+
B		D		F	/(+
)	Empty	)	*	)	/
*	*	/	/	$	/$
				G	

Operand Stack



Final Tree

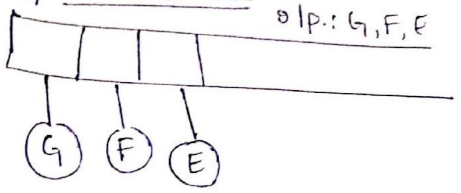
Prefix :- /*+AB-CD$+EFG

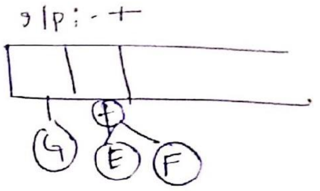
postfix :- AB+CD-*EF+G$/

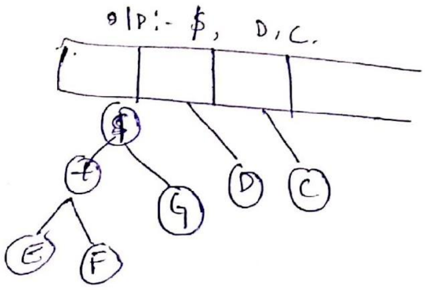
Construct the expression (5) tree for the prefix expression /*+AB-CD$+EFG and write the other two forms by traversing in the required order.

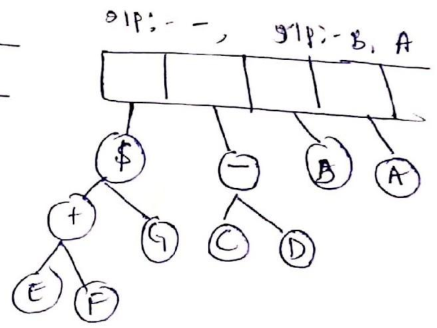
/*+AB-CD$+EFG

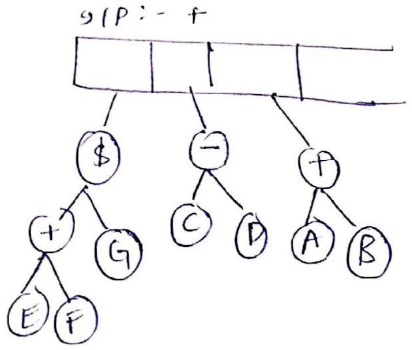
operand stack

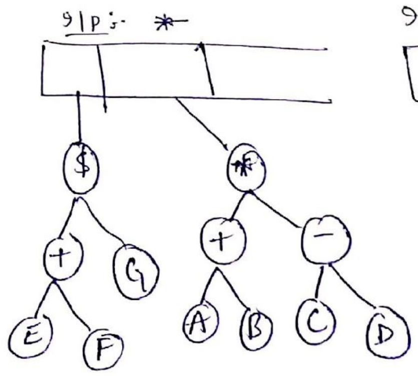


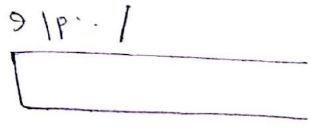


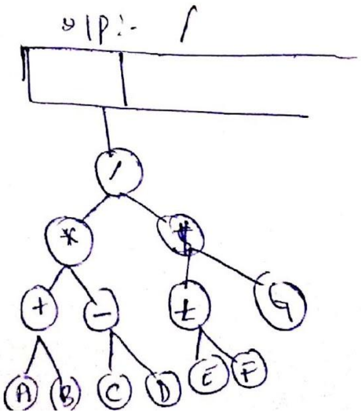












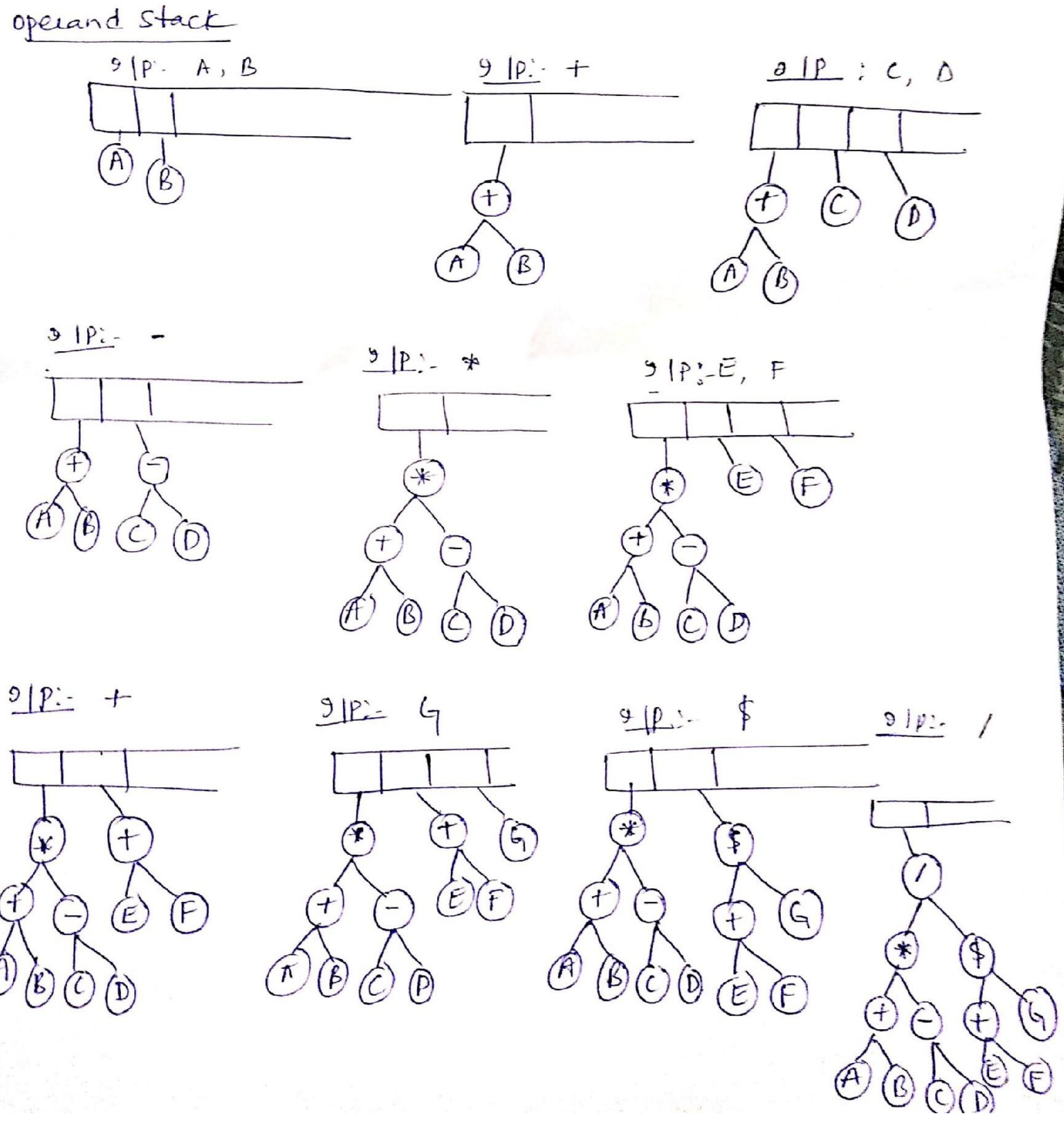
Infix:- ((A+B) * (C-D))/((E+F) $ G

postfix:- AB+CD-*EF+G$/

(4)

Construct the expression tree for the postfix expression AB+ CD- *EF+G$/ and write the other two forms of the expression by traversing in the required order.

AB + CD - * EF + G $ /


operand stack
o/p:- A, B
o/p:- +
o/p : C, D
o/p:- -
o/p:- *
o/p:- E, F
o/p:- +
o/p:- G
o/p:- $
o/p:- /
A
B
+
C
D
-
*
E
F
G
$
/