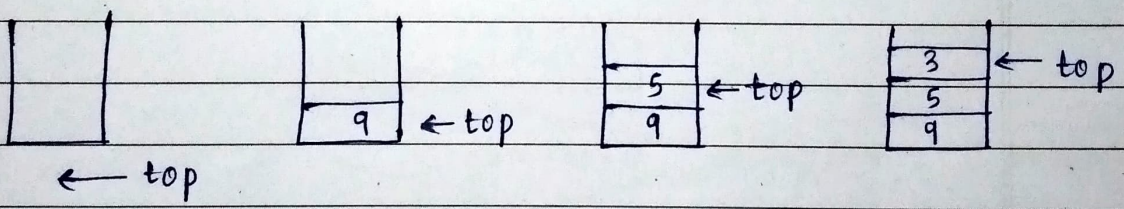
DATE
PAGE NO

* Data structure used:

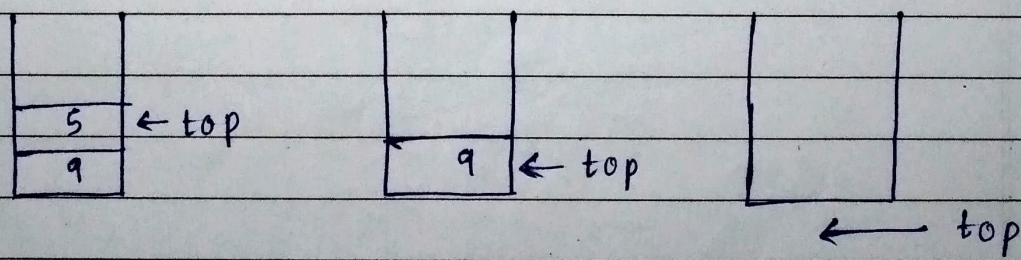
→ Stack :-

- Stack is a LIFO (Last in first out) structure.
- It is an ordered list of the same type of elements.
- A stack is a linear list where all insertions and deletions are permitted only at the end of the list.
- When elements are added to stack it grows at one end.
- Similarly, when elements are deleted from a stack, it shrinks at the same end.

i) Addition of elements :-



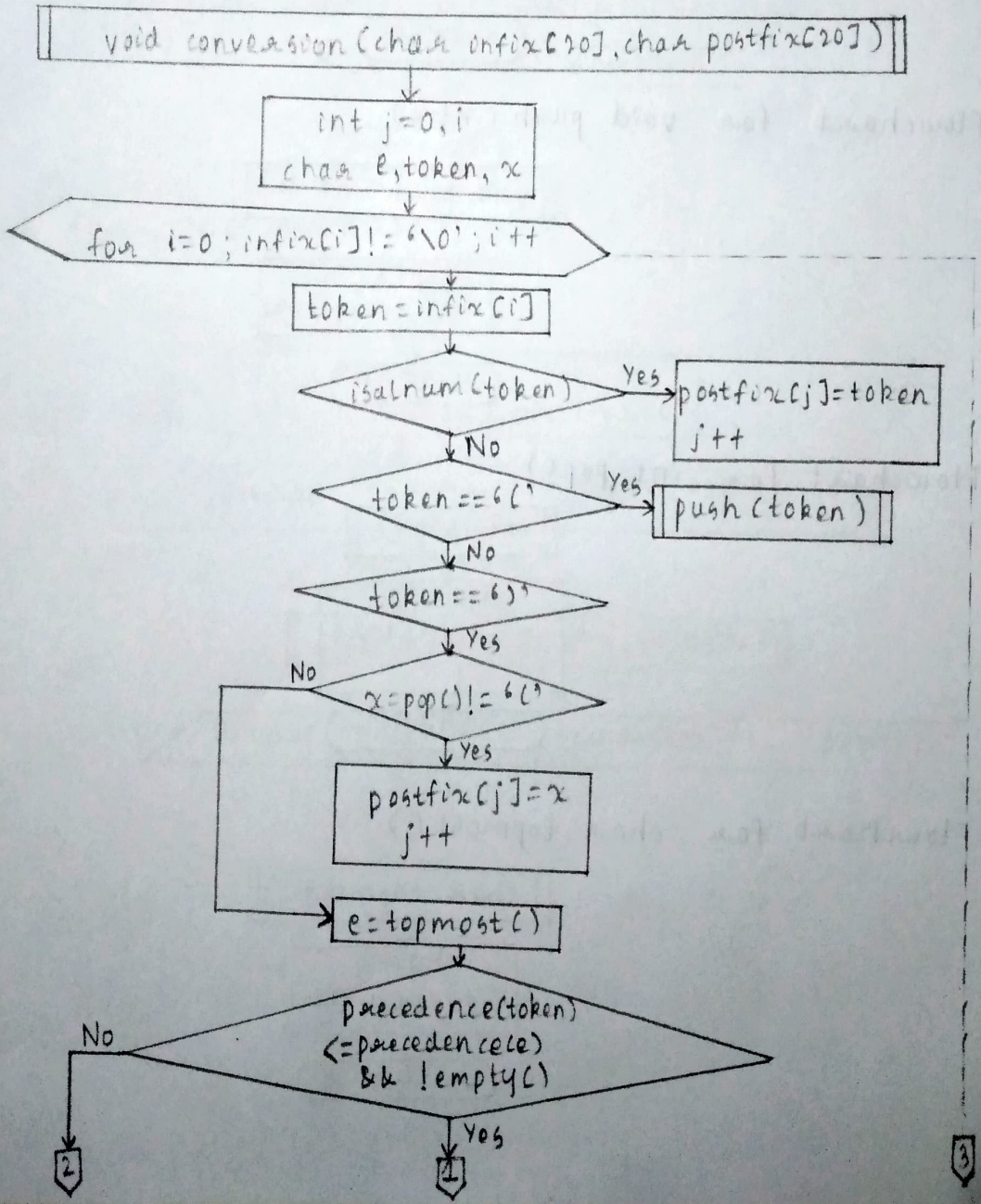
ii) Deletion of elements :-

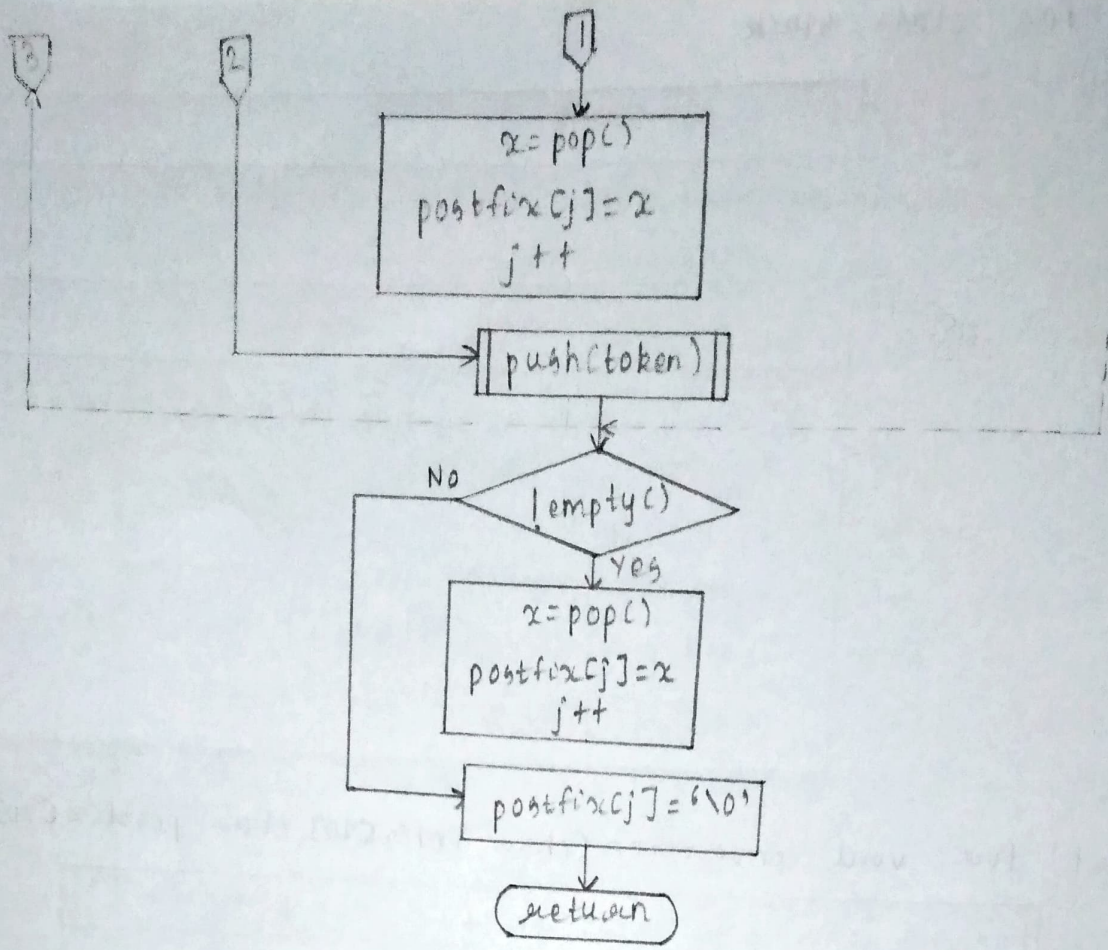


Flowchart for class stack

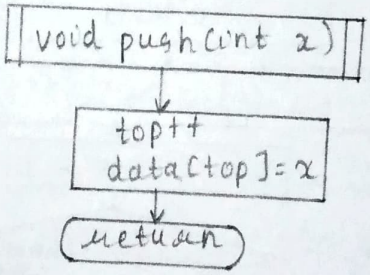
class stack
int top char data[20]
stack(){top=-1;}
void conversion(char infix[20], postfix[20]) void push(int x) int pop() char topmost() int precedence(char x) int empty()

→ Flowchart for void conversion(char infix[20], char postfix[20])

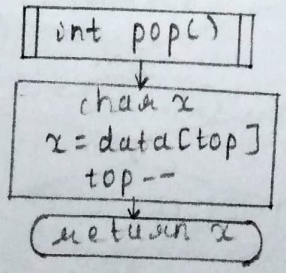




→ Flowchart for void push(int x)



→ Flowchart for int pop()



→ Flowchart for char topmost()

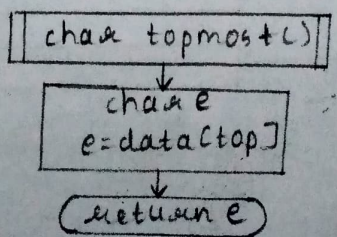
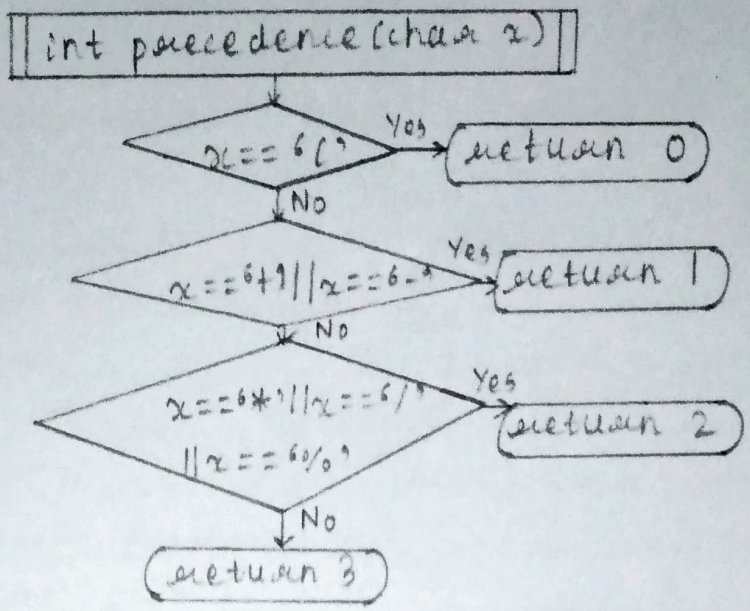
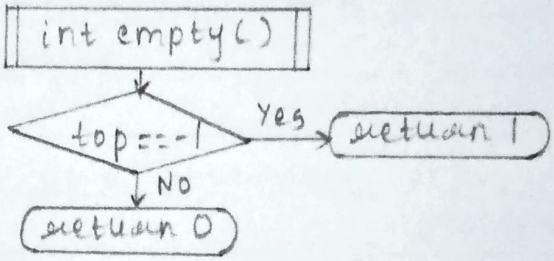


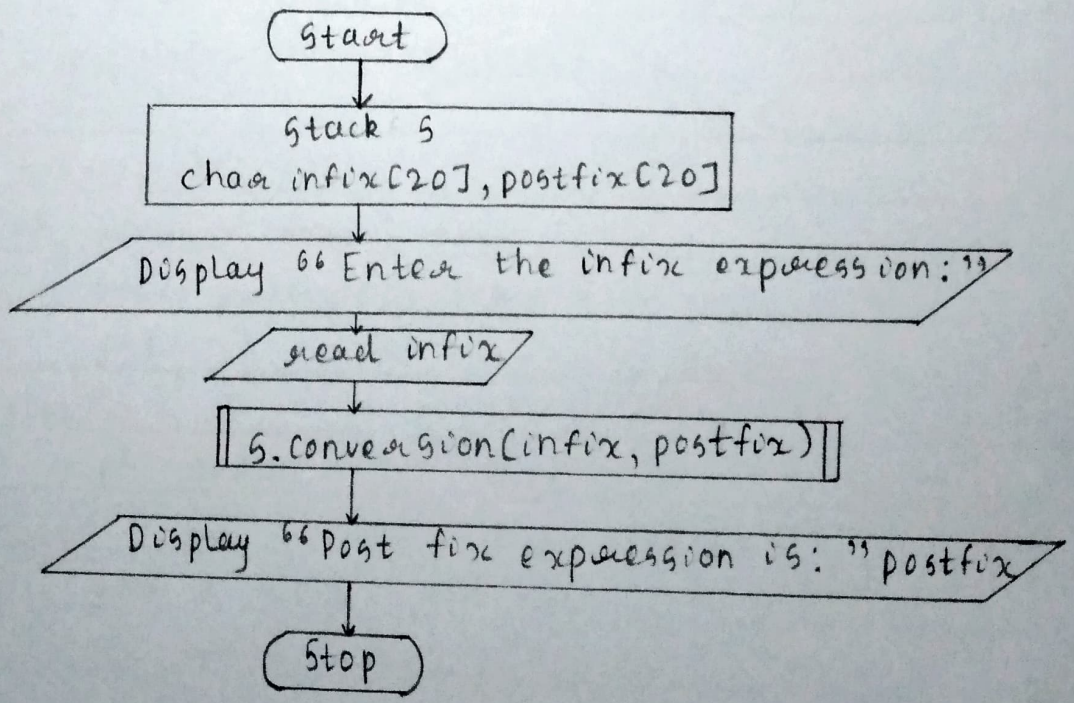
chart for int precedence(char x)



→ Flowchart for int empty()



→ Flowchart for int main()



→ Pseudocode for class stack

1. Declare int top
   char data [20]
2. Create stack()
   initialize top=-1
3. Declare void conversion (char infix [20], postfix[20])
   void push (int x)
   int pop()
   char topmost()
   int precedence (char x)
   int empty

→ Pseudocode for void conversion (char infix[20], char postfix [20])

1. Declare int j=0, i
   char e, token, x
2. for i=0 ; infix [i] != '\0' ; i++) do
   begin
   token = infix [i]
   if isalnum (token) then
   store postfix [j] = token
   j++
   else
   if token == '(' then
   call function push (token)
   else if token == ')' then
   while x=pop() != '(' do
   begin
   postfix [j] = x
   increment j
   end

else

store e = topmost()

while precedence(token) <= precedence(e)

&& !empty() do

begin

x = pop()

store postfix[j] = x

increment j

end

call function push(token)

end

3. while !empty() do

begin

store x = pop()

postfix[j] = x

increment j

end

4. store postfix[j] = '\0'

5. Return

→ Pseudocode for void push(int x)

1. Increment top.
2. store data[top]
3. return

→ Pseudocode for int pop()

1. Declare char x
2. store x = data[top]
3. Decrement top
4. return x

→ Pseudocode for char topmost()

1. Declare char e
2. store e = data[top]
3. return e

→ Pseudocode for int precedence (char x)

1. if x == '(' then
   return 0
2. if x == '+' || x == '-' then
   return 1
3. if x == '*' || x == '/' || x == '%' then
   return 2

else
   return 3

→ Pseudocode for int empty()

1. if top == -1 then
   return 1

else
   return 0

→ Pseudocode for int main()

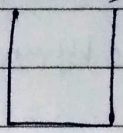
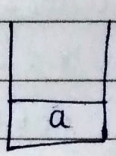
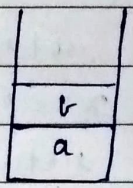
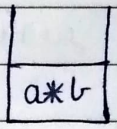
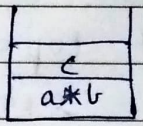
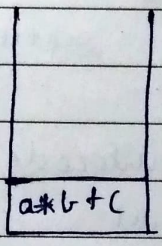
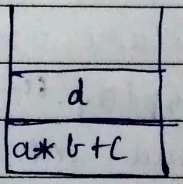
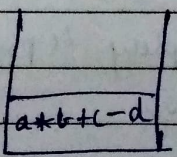
1. start
2. create stack s
3. Declare ~~cha~~ infix [20], postfix [20]
4. Display "Enter ~~valid~~ the infix expression: "
5. Read infix
6. call funct von s.conversion (infix, postfix)
7. Display "post fix expression is: " postfix
8. Stop

DATE

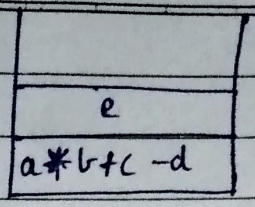
PAGE NO

Q1 Evaluate the following postfix expression and show all steps: ab* c+d-e+

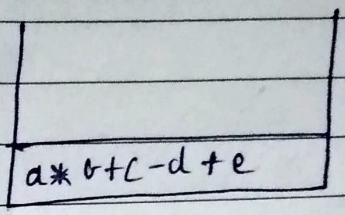
Ans

	Expression	Stacks
1)	ab*c+d-e+	
2)	b*c+d-e+	
3)	*c+d-e+	
4)	c+d-e+	
5)	+d-e+	
6)	d-e+	
7)	-e+	
8)	e+	

9) +



10) -



Q2. Write an algorithm for prefix to postfix.

Ans. 1. Prefix expression is scanned from left to right.

2. when an operand is found, it is pushed onto the stack

3. when an operator is found, two-sub expressions are popped from the stack.

4. op1() = second subexpression
   op2() = first subexpression

5. These 2 subexpressions and operator are merged to create a postfix sub-expression

6. Postfix sub-expression is pushed onto the stack