Unit - III

STACKS

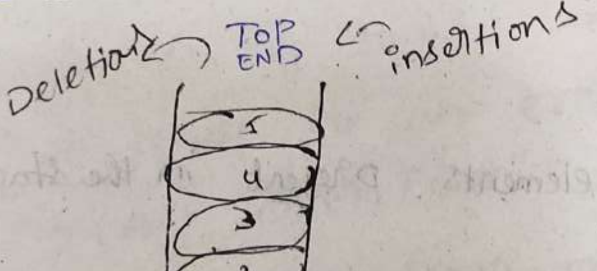
Definition:-

Stack is a linear data structure in which addition of Elements and deletion of elements are done at only one end called "TOP".

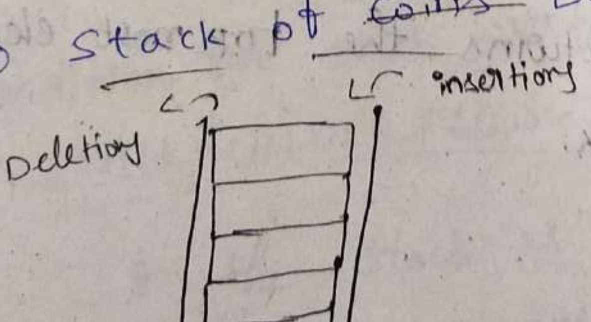
Stack is based on the principle called Last In First Out (LIFO).

Example:-

① Stack of Plates



② Stack of ~~coins~~ Books:

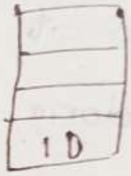


Operations of the Stack:-

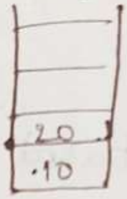
1) Push (item) :-

Adding an element on to the Top of the stack

Ex:- ① Push(10) ⇒



② push (20) ⇒



2) Pop ( ) :-

Deleting an element which is present on the Top of the stack.

3) Display ( ):-

Printing the elements present in the stack

4) Peek( ) :-

Peek() returns the top most element in the stack.

5) isEmpty():-

It is used to find whether the stack is empty or not.

6) isFull():

It is used to find weather the stack is full or not.

-> Stack Created Terms:-

1. top:

top is a variable that which always points to the top most element in the stack.

2. STACK OVER FLOW:-

If the stack is already full and if we try to add one more element, it leads to an error situation called "STACK OVER FLOW".

* STACK OVER FLOW means stack is full.

* Condition for stack overflow => $\boxed{\text{if (top == n-1)}}$

*3. STACK UNDER FLOW:-

If the stack is already empty, and if we try to delete one more element from the from the stack it leads to an error situation called "STACK UNDER FLOW".

* STACK UNDER FLOW => Stack is Empty

* Condition for stack under flow => if (top == -1)

→ Representation of STACK:

There are two ways to represent a stack.

1) STACK using ARRAYS.

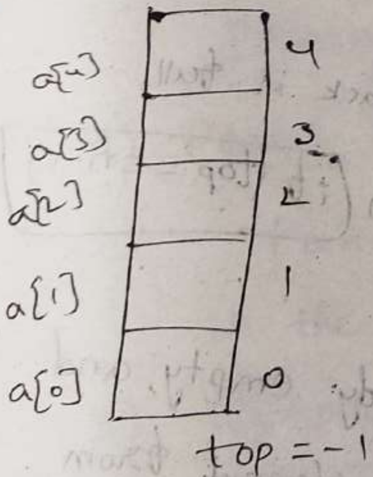
2) STACK using LINKED LIST (pointers)

1) STACK using ARRAYS:-

* Declaration of STACK,

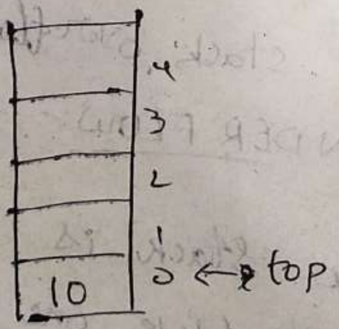
```
int a[5];
top = -1;
```

Step-1: Initially, STACK is EMPTY



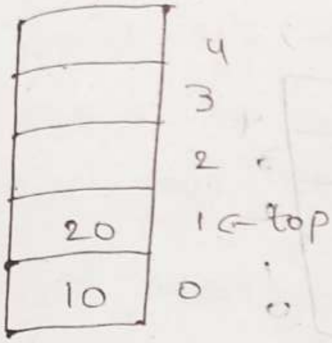
Step-2:- push(10)

```
top++;
```



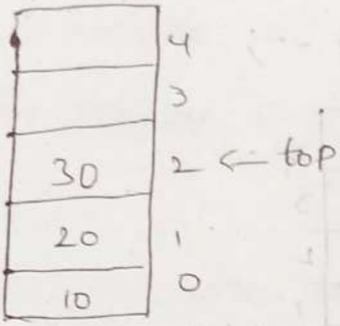
step-3 : Push(20)

top++;



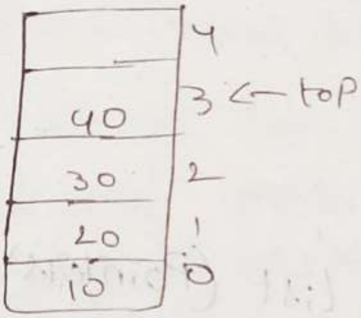
Step-4 :- push(30)

top++;



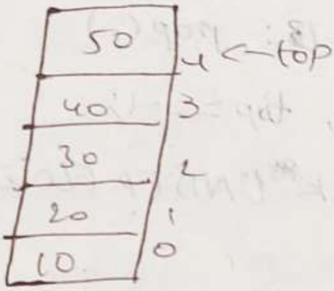
step-5 : push(40).

top++;



Step-6 :- push(50)

top++;



Step-7 :- Push(60)

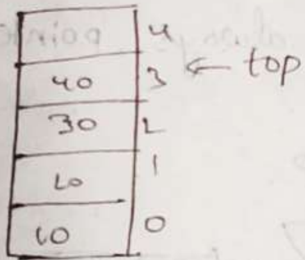
~~top~~

since, (top = = n - 1),

=) STACK OVER FLOW

step-8 :- pop ( )

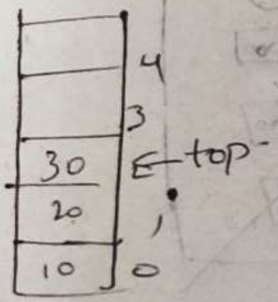
top--;



Deleted element is : 50

step-9 : pop( )

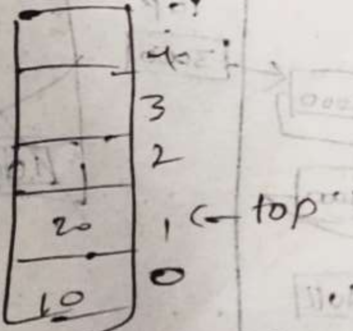
top--;



Deleted ele is: 40

Step-10 : pop ( )

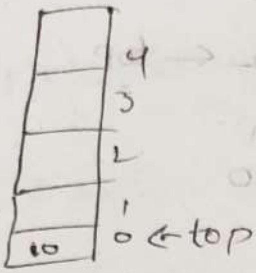
top--;



Deleted ele is: 30

Step-11:- pop( )

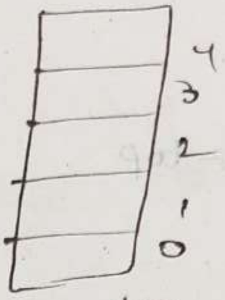
top--;



Delcted ele is:20

Step-12:- pop( )

top--;



top=-1

Deleted ele is: 10

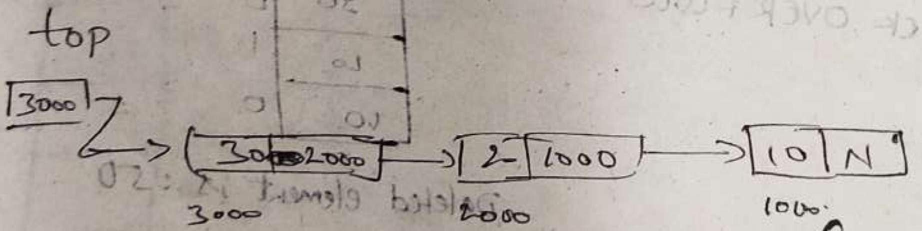
Step -13:- pop( )

Since, top==-1;

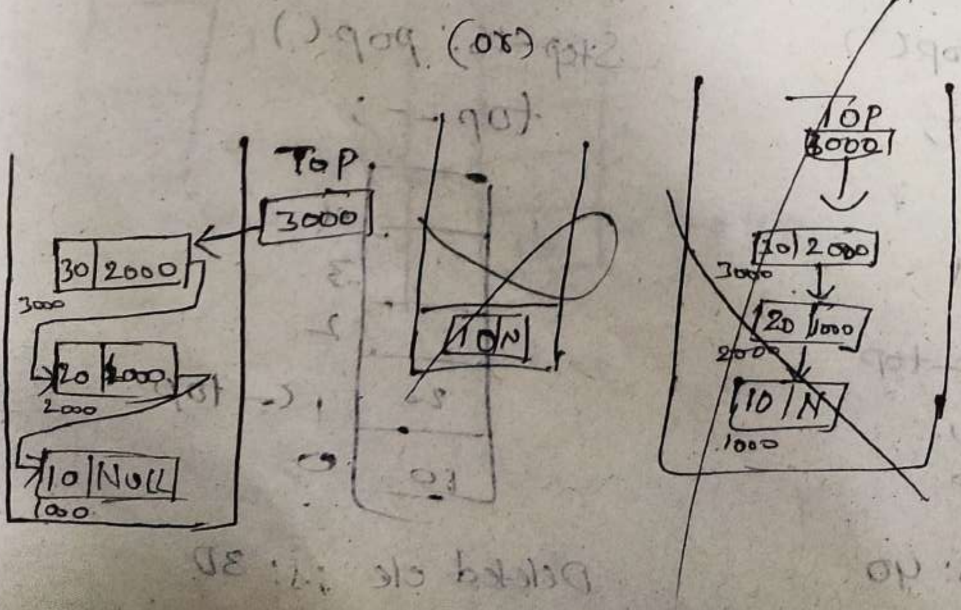
=) STACK UNDERFLOW

2) STACK Using Linked List (Pointers):-

In this representation the topmost element is always pointed by top pointer.



(or)



* Properties of a STACK:-

-> STACK is a linear data structure.

-> STACK is based on the principle "Last In First out" (LIFO).

-> All insertions and deletions are ~~only~~ done at only one end called "Top".

-> only one element can be pushed (or) ~~Popped~~ Popped from a stack at a time.

-> Elements can be removed (in) only in reverse order, in which they are inserted

-> STACK OVER FLOW occurs, ~~ET~~ If we Push any item when the stack is full.

-> STACK UNDER Flow occurs, if we Pop any item when the stack is Empty.

~~Write~~

Write a C PROGRAM TO IMPLEMENT STACK USING ARRAYS. (Lab-week-5)

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

void push (int item);
Void POP [illegible] ();
void display ();

int stack [10]; top=-1, n,, ele, i;

Void main ()
{
    int choice;
    clrscr();
    Printf ("\n Enter the size of the Stack:");
    scanf ("%d", &n);
    ~~while (1)~~
    ~~{~~
        Printf ("\n\t *** MAIN MENU***);
        Printf ("\n 1. PUSH  2. POP  3. DISPLAY
                 4. EXIT");
        Printf ("\n **********");
    while (1)
    {   Printf ("\n Enter your Choice:");
        scanf ("%d", &choice);
        Switch (choice)
        {
                        Printf ("\n Enter the Data:"); topush
            case 1:     ~~Push();~~
                         Scanf ("%d", &ele);
~~Case 2 : Pop();~~
~~break;~~
~~Case 3 :~~

        Push (ele);
        break;

    case 2 : pop ();
             break;

    case 3 : display();
             break;

    case 4 : exit(1);
             break;
    default : printf ("\n WRONG CHOICE :");
             break;

    }

  }
  getch();
}

void push(int ele)
{
  if (top ==n-1)
  {
    printf ("\n STACK OVER FLOW");

  }
  else
  {
    top++;
    stack[top] = ele;
  } printf ("\n The given data is pushed into the STACK.");
}
```

```
void pop()
{
    if(top==-1)
    {
        printf("\n STACK UNDER FLOW-NO ELEMENTS
                   IN THE STACK");
    }
    else
    {
        printf("\n Deleted element is %d", stack
                                           [top]);
        top--;
    }
}
void display()
{
    if (top==-1)
    {
        printf("\n There are no elements in
                the (lis) STACK to Display:");
    }
    else
    {
        printf("\n The elements in the STACK are:")

        for (i=0 ; i<=top ; i++)
        {
            printf("%d ", stack[i]);
        }
    }
}
```

Output:-

```
Enter the size of the STACK : 3

        * * * MAIN MENU * * *
  1. PUSH   2. POP   3. DISPLAY   4. EXIT
      * * * *  * * *  * * * *

    Enter your choice : 1
  Enter the element to push: 10
  The given data is pushed into the STACK.

  Enter your choice : 1
  Enter the element to push: 20
  The given data is pushed into the STACK.

  Enter your choice : 1
  Enter the element to push: 30
  The given data is pushed into the STACK.

  Enter your choice : 1
  Enter the element to push: 40

  STACK OVER FLOW.

  Enter your choice : 2
  The Deleted element is 30
```

```
Enter your Choice: 2
the Deleted element is 20.
Enter your Choice : 2
The Deleted element is 10.
Enter your Choice : 2
STACK UNDER FLOW - No elements In the
STACK.
Enter your Choice : 4
```

Write a C program to implement STACK Using Linked List. (Lab program)

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

struct node
{
    int data;
    struct node *next;
};

struct node *top, *temp, *new_node;
void push();
void pop();
void display();
[~~void search()~~;]
```

```
void main()
{
    int choice;
    clrscr();
    while(1)
    {
        printf("\n            MAIN MENU ");
        printf("\n 1.PUSH  2.POP  3.DISPLAY
                4. EXIT\n");
        printf("*** *** * ********** ** * *");
        printf("\n Enter your choice:");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: push();
                    break;

            case 2: pop();
                    break;

            case 3: display();
                    break;

            case 4: exit(1);
                    break;
        }

    }
    getch();
}
```

```
Void push()
{
    new_node = (struct node*)malloc (size of (struct node));
    printf ("\n Enter the data:");
    scanf ("%d", &new_node->data);
    new_node -> next = NULL;
      if (top ==NULL)
      {
          top = new_node;
      }
      else
      {
          new_node -> next = top;
           top = new_node;
       }
       getch ();
}

 void pop()
 {
     if (top==NULL)
     {
       Printf ("\n There are no elements in the
                 list");
     }
     else   if (top->next ==NULL)
     {
         Printf("\n Deleted node is %d", top->data
         top = NULL;
         free (top);
```

```
        }
        else
        {
            printf ("\n Deleted node is %d",
                        top -> data);
            top = top ->next

        }
        getch();
    }

void display( )
{
    if (top == NULL)
    {
       printf (" There are no (nodes) elements in the list");

    }
    else
    {
                                            STACK
       printf ("\n Elements in the list are:\n");
       temp = top;
       while (temp != NULL)
        {
            printf("%d\n", temp -> data);
            temp = temp -> next;
        }

    }
}
```

Output:

```
                MAIN MENU

1. PUSH   2. POP   3. DISPLAY   4. EXIT

  ********.***********

Enter your choice : 1

Enter the data : 10

Enter your choice : 1

Enter the data : 20

Enter your choice : 1

Enter the data : 30
```

Reprsentation of Arithmetic Expressions:-

→ There are three Notations to represent Arithmetic Expression.

1) Infix Expression (or) Notation
2) Prefix Expression (or) Notation
3) postfix Expression (or) Notation

1. Infix Expression (or) Notation :-

Definition:-

In this notation, operator comes in between the operands.

Ex:- a+b,
a-b
a*b

2. Prefix Expression (or) Notation:-

Definition:-

In this notation, operator comes before the operands.

Ex:- +ab
-ab
*ab

3. Postfix Expression (or) Notation:-

Definition:-

In this notation, operator comes after the operands.

Ex:- ab+,
ab-
ab*

Applications of STACK :-

1) Evaluation of postfix expression

2) Converting the infix expression to postfix expression.

3) ~~(Conve)~~ Recursion

4)

5)

6)

1. Evaluation of postfix expression :-

Algorithm for evaluation of Postfix Expression:-

Step - 1 : Read the postfix expression from left to right, character by character. ~~Until the last ch~~ perform the below steps for each character.

Step - 2 : If the input character is operand, Push it on to the stack.

Step - 3 : If the input character is operator, then perform the below three steps one by one.

(i) perform pop () operation for 2 times

(ii) Apply the operator on operands

(iii) Push the result on to the STACK.

step-4 : After reading all the characters, perform Pop() operation and print the result on the screen.

EXAMPLE1:-

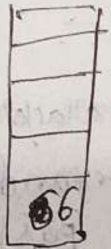
Evaluate the post fix expression

6 8 + 9 2 - /

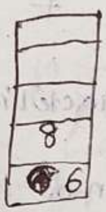
17 4

4

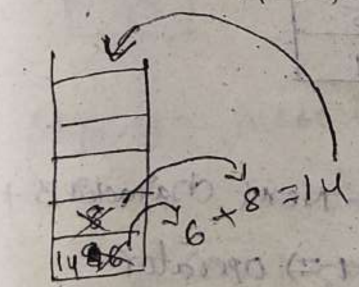
Step-1 :- 6 -> operand
Push(6)



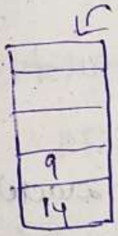
Step 2 : Next character is
8 => operand
Push(8)



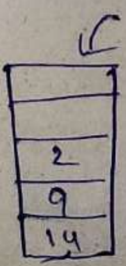
Step-3 : Next character is +
+ => operator



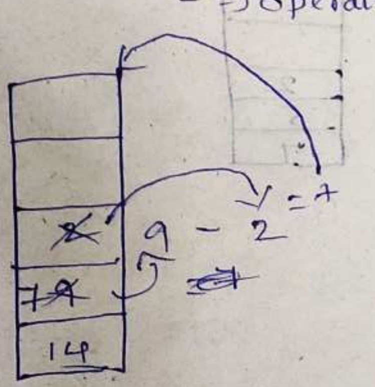
Step-4 :- Next character is
9 => operand
push(9)



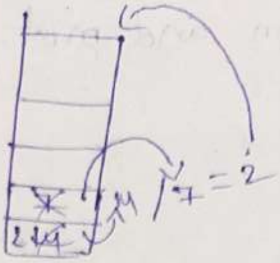
Step-5 : Next character is 2
2 => operand
push(2)



Step-6 : Next character is -
- => operator



Step-7 :- Next character is / => operator



=> perform pop() and print the result

o/p :- Evaluation of the above post fix exp is '2'

Example 2 :-

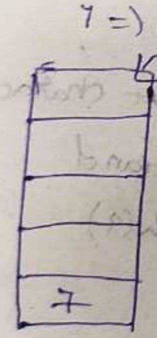
Evaluate the following post fix expression.

7 5 2 + * 4 1 5 - / -

Step 1 :- First character is 7

7 => operand

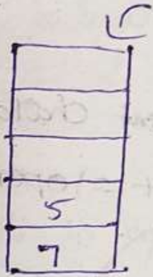
push(7)



Step-2 :- Next character is 5

5 => operand

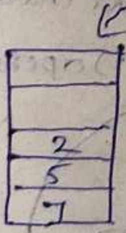
push(5)



Step-3 :- Next character is 2

2 => operand

push(2)



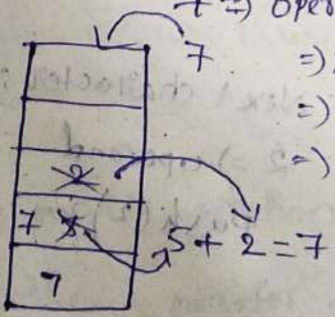
Step-4 :- Next character is +

+ => operator

=> pop() 2 times

=> Apply operator

=> push result



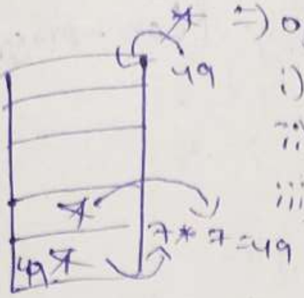
Step-5 :- Next character is *   Step-6:- Next ch is 4

* => operator

i) pop() 2 times

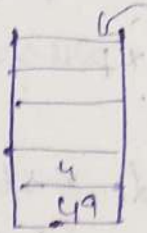
ii) Apply operator

iii) push result



4 => operand

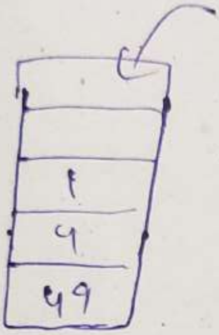
push(4)



Step 7:- Next ch is 1

1 => operand

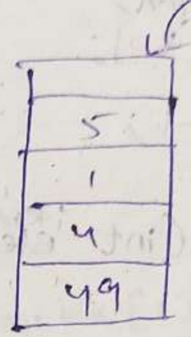
push(1)



Step 8:- Next ch is 5

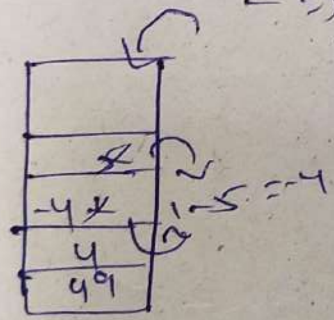
5 => operand

push(5).



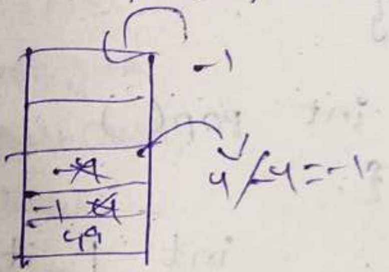
Step 9:- Next ch is -

- => operator.



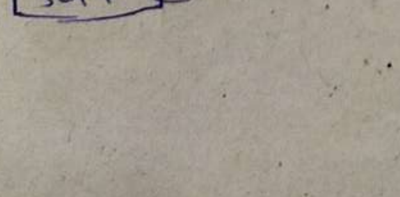
Step-10 :- Next ch is /

/ => operator



Step-11 :- Next Ch is -

- => operator



=> perform pop() and print the result

o/p => Evaluation of the above post fix is 50

* write a C program for evaluation of Postfix expression.(using STACK) [Lab-program]

```
→   #include <stdio.h>
    #include <conio.h>
    #include <ctype.h>
    #include <math.h>
    # include <string.h>
    int stack[20];
    int top = -1;

                    item
    void push (int  ele)
    {

        top ++;
        stack[top] = item;
    }

     int pop( )
     {
         int p_item;
         p_item = stack [top];
         top --;
         return p_item;
     }

    void main ( )
    {
     char postfix [20];
     int i,j,k,x,n,y,res;
     clrscr();
     printf("\n Enter the postfix expression:");
     get.s (postfix);
```

```
for (i=0 ; postfix [i] != "\0" ; i++)
{
    if (isalpha (post fix [i]))
    {
        printf ("\n Enter the value of %c:", postfix
                                              [i]);
        scanf ("%d", &n);
        Push (n);
    }

    else
    {
        switch (postfix [i])
        {
            case '+': x = pop ( );
                      y = pop ( );
                      res = y+x ;
                      push(res);
                      break;

            case '-':- x = pop();
                       y = pop ();
                       res = y -x;
                       push (res);
                       break;

            case '*':- x =pop () ;
                       y=pop ();
                       res = y * x;
                       push (res);
                       break;

            case '/':  x = pop ();
                       y = pop();
                       res = y /x;
```

```
                              break;
                    }
              }
      }
      printf("\n Result of postfix expression is %c",
                                    pop());
      getch();
  }
```

~~Converting E~~

Application 2

Converting Infinix Expression to post fix expression (using STACK):-

Algorithm:-

Step -1:

Add the left paranthesis '(' at the beginning and right paranthesis at the end of the expression.

Step-2 :-

Initialise the STACK to be EMPTY.

Step -3 :-

Scan the expression from left to right character by character and perform the below steps.

Step -4: If the input character

i)	Left Paranthesis '('	Push it on to the STACK
ii)	operand (A/B/C/D...)	Add it to the POST FIX Expression.
iii)	Operator + - * / ^	a) If top of stack contains Left Paranthesis, Push the operator on to the stack b) If the input operator (Senior) has high precedence than the operator (Junior) on the TOP of the STACK, Push the operator (Senior) on to the STACK.

		c) If the input operator has lower OR equal precedence than the operator on the TOP of the STACK, then (i) POP the operator on the top of the stack and add it to POSTFIX EXPRESSION. (ii) Add the input operator on to the top of the STACK.
iv	Right Paranthesis ')'	a) Pop all the operators from the STACK and add them to the POSTFIX Expression. b) Pop the left Paranthesis and discard it.

Step-5 :- Print the resultant POSTFIX Expression on the Monitor.

Example:-

Convert the following infix expression into postfix expression.

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

Sol:- Add left & right paranthesis at the beginning and End of the infix expression

$$\Rightarrow (A * B / (C - D) + E * (F - G))$$

i/p character	STACK	Postfix Expression
--	EMPTY	--
(	(	—
A	(	A
*	(*	A
B	(*	AB
/	(/	AB*
(	(/(	AB* e
C	(/(	AB*C
-	(/(-	AB*C
D	(/(-	AB*CD
)	(/	AB* CD-
t	(+	AB*CD-/
E	(+	AB*CD-/E
*	(+*	AB*CD-/E
(	(+*(	AB*CD-/E
F	(+*(	AB*CD-/EF
-	(+*(-	AB*CD-/EF
G	(+*(-	AB*CD-/EFG
)	(+*	AB*CD-/EFG-
)	EMPTY	AB*CD-/EFG-**+

The resultant postfix expression is .

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

Example - 2 :-

Convert the following infix expression to postfix expression

a / b - c + (d * e) - a * c

(a / b - c + (d * e) - a * c)

i/p character	STACK	postfix expression
-	Empty	- - -
(	(	- - -
a	(	a
/	(/	a
b	(/	ab
-	(-	ab/
c	(-	ab/c
+	(+	ab/c-
(	(+(	ab/c-
d	(+(	ab/c-d
*	(+(*	ab/c-d
e	(+(*	ab/c-de
)	(+	ab/c-de*
-	(-	ab/c-de*+
a	(-	ab/c-de*+a
*	(-*	ab/c-de*+a

C	(-*	ab/c -de* + ac
)	EMPTY	ab/c - de * + ac*-

The resultant post fix expression is
ab/c - de * + a c * -

Example-3:-

Convert the following infix expression into Postfix expression.

A+(B*C -(D/E ∧ F) * H)

~~Add~~ the ~~left para~~ '(' at the beginning and ')' at the end Then, (A+(B*C-(D/E∧F)*H))

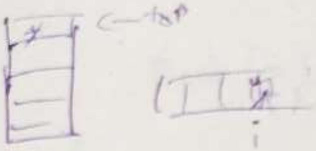
Input character	STACK	POST fix Expression
--	EMPTY	- - - -
(	(	- - -
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
C	(+(*	ABC
-	(+(-	ABC*
(	(+(-(	ABC*
D	(+(-(	ABC*D
/	(+(-(/	ABC*D
E	(+(-(/	ABC*DE
∧	(+(-(/∧	ABC*DE
F	(+(-(/∧	ABC*DEF
)	(+(-	ABC*DEF∧/
*	(+(-*	ABC*DEF∧/

H	(+(-*	ABC*DEF^/ H
)	(+	ABC*DEF^/H*-
)	EMPTY	ABC*DEF^/H*-+

write a C program for converting Infix expression to postfix expression (Lab program)

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
char stack[20], infix[20], postfix[20];
int top = -1;
voidpush(char ch)
{
     top ++;
     stack [top] =ch;
 }
char pop ()
 {
        char ch;
        ch = stack[top];
        top--;
        return ch;
  }
  int precedence (char ch)
  {
        switch (ch)
        {
             case '(': return 0;

             case '+':
             case '-': return 1;
             case '*':
                case '/':
                case '%': return 2;
                case '^': return 3;

        }

    }

    void main()
    {
        int  i, j = 0;
        clrscr();
        [~~printf~~("\n Enter infix your" expression:");
        ~~Scanf~~("%]
        Printf("\n Enter your infix expression
                (Add '(' at beginning and ')' at
                 end):");
        scanf("%s", infix);
        (~~Push~~ )
        for(i=0 ; infix[i]!='\0' ; i++)
        {
            if (isalpha (infix[i]))
            {
                postfix[j] = infix[i];
                j++;
            }
            else
            {
```

```
switch (infix[i])
{
        case '(' : push (infix[i]);
                   break;
        case ')' : while (stack[top]!= '(')
                   {
                        Postfix[j] = Pop();
                        j++;
                   }
                   Pop();
                   break;
          default : if (stack[top] == '(')
                        push(infix[i]);

else
{
    while (precedence (stack[top]) >= precedence (infix[i])
                                                      )
    {
        postfix[j]=pop();
        j++;
    }
    Push (infix[i]);

}
break;
}
}
Postfix [j]= '\0';
Printf ("\n The resultant. Postfix Expression is: %s",
          Postfix);
getch();
```



Reversing the list using stack:

Write a C program to reverse the given array using stack. (Lab program)

```
  #include <stdio.h>
  # include <conio.h>

  int stack [20], top = -1;
void Push (int ele)
  {
     top ++;
     stack [top] = ele;
  }
int   Pop( )
  {
     int r_ele;
     r_ele = stack [top];
     top--;
     return r_ele;
  }
 void main()
 {
   int a[20], n, i;
   clrscr();
   Printf("\n Enter no.of elements in the list:");
   scanf("%d", &n);
   Printf("\n Enter the elements in the
                list:\n");
for ( i=0 ; i<n ; i++)
{
    scanf ("%d", &a[i]);
}
for (i=0 ; i<n ; i++)
{
    push (a[i]);
}
for (i=0 ; i<n ; i++)
{
    a [i] = pop ();
}
printf("\n Reversed list is:\n");
for (i=0 ; i<n ; i++)
{
    printf ("%d ", a[i]);
}
getch();
}
```

Input:

```
Enter no.of elements in the list : 5
Enter the elements in the list:
10 20 30 40 50
```

Output:

```
Reversed List is:
50 40 30 20 10
```