**Assignment No : 2**

Name: Rasika Mahure

PRN:72018258B

Div : B

Roll No: S1951093

**Aim :-**

Implement stack as an abstract data type using singly linked list and use this ADT for conversion of infix expression to postfix, prefix and evaluation of postfix and prefix expression.

# **//PROGRAM**

#include<iostream> using namespace std; typedef struct node { char data; struct node \*next;

}node; void infix\_postfix(char infix[20],char postfix[20]); void reverse(char a[20],char b[20]); void infix\_prefix(char infix[20],char prefix[20]); int evaluate(int op1,int op2,char op); void evaluate\_postfix(char postfix[20]); int precedence(char x); void evaluate\_prefix(char prefix[20]); int main() { char infix[20],token,postfix [20], prefix[20]; int ch,result; do { cout<<"\n1. Infix to postfix expansion"; cout<<"\n2. infix to prefix expansion"; cout<<"\n3. Evaluate postfix"; cout<<"\n4. evaluate prefix"; cout<<"\n5. exit"; cout<<"\nEnter your choice : "; cin>>ch; switch(ch)

{

case 1:cout<<"\nEnter Infix expression : "; cin>>infix; infix\_postfix(infix , postfix); cout<<"\nPostfix := "<<postfix; break; case 2:cout<<"\n Enter infix expression : "; cin>>infix;

infix\_prefix(infix,prefix); cout<<"\n prefix expression is : "<<prefix; break; case 3:

evaluate\_postfix(postfix); break; case 4:

evaluate\_prefix(prefix); break; } }while(ch!=5); return 0;

} class stack { node

\*top; public: stack() { top = NULL; } int isempty() { if(top==NULL) return 1; return 0; } void push(char x) { node \*p; p=new node(); p->data = x; p->next=top; top = p; } char pop() { node \*p; char x; p = top; x = p-

>data; top =top>next; delete(p); return x; } char topdata() { return top->data;

} };

int precedence(char x) { if(x=='(') { return 0; } if(x=='+' || x=='-') { return 1; } if(x=='\*' || x=='/') { return 2; } return 3;

} int evaluate(int op1 , int op2 , char op)

{ if (op =='+') return op1 + op2 ; if (op =='-') return op1 - op2 ; if (op =='\*') return op1 \* op2 ; if (op =='/') return op1 / op2 ; if (op =='%') return op1 % op2 ; } void evaluate\_postfix(char postfix[20]) { stack s; int i,op1,op2,result; char token; int x; for(i=0;postfix[i]!='\0';i++)

{ token=postfix[i]; if(isalnum(token)) {

cout<<"Enter the value "<<token <<" : "; cin>>x;

s.push(char(x)); } else { op2=s.pop(); op1=s.pop(); result=evaluate(op1,op2,token); s.push(char(result)); } } result=s.pop(); cout<<"result = "<<result;

}

void infix\_postfix(char infix[20],char postfix[20])

{ stack s; int i,j=0; char token,x; for(i=0;infix[i]!='\0';i++) { token = infix[i]; if(isalnum(token)) { postfix[j]=token; j++; } else { if(token=='(')

s.push(token); else if(token==')')

{ while((x=s.pop())!='(') { postfix[j]=x; j++;

}

} else { while(s.isempty()! =1 && precedence(token)< =precedence(s.topd ata() ))

{ postfix[j]=s.pop();

j++; }

s.push(token);

} } } while(s.isempty()!=1) { postfix[j]=s.pop(); j++; } postfix[j]='\0';

}

void infix\_postfix1(char infix[20],char postfix[20])

{ stack s; int i,j=0; char token,x;

for(i=0;infix[i]!='\0';i++)

{ token=infix[i]; if(isalnum(token)) { postfix[j]=token;

j++; } else

{ if(token=='(')

s.push(token); else if(token==')') { while((x=s.pop())!='(') { postfix[j]=x; j++;

}

} else { while(s.isempty()! =1 && precedence(token)< precedence(s.topda ta()) )

{ postfix[j]=s.pop();

j++; }

s.push(token);

} } } while(s.isempty()!=1) { postfix[j]=s.pop(); j++; } postfix[j]='\0'; } void reverse(char a[20],char b[20]) { int i,j=0; for(i=0;a[i]!='\0';i++) { } i--; for(j=0;i>=0;j++,i--) { if(a[i]=='(') b[j]=')'; else if(a[i]==')') b[j]='('; else b[j]=a[i]; } b[j]='\0';

} void infix\_prefix(char infix[20],char prefix[20])

{ char prefix1[20],infix1[20]; reverse(infix,infix1); infix\_postfix1(infix1,prefix1);

reverse(prefix1,prefix);

} void evaluate\_prefix(char prefix[20]) { stack s; int i,op1,op2,result; char token; int x; for(i=0;prefix[i]!='\0';i++) {} i--; for(;i>=0;i--) { token=prefix[i];

if(isalnum(token))

{ cout<<"Enter the value

"<<token<<" : "; cin>>x;

s.push(char(x)); } else { op1=s.pop(); op2=s.pop(); result=evaluate(op1,op2,token);

s.push(char(result)); } } result=s.pop();

cout<<"result="<<result;

}

**//OUTPUT**

1. Infix to postfix expansion
2. infix to prefix expansion
3. Evaluate postfix
4. evaluate prefix 5. exit

Enter your choice : 1

Enter Infix expression : 3+2\*(5-3)+9/3

Postfix := 3253-\*+93/+

1. Infix to postfix expansion
2. infix to prefix expansion
3. Evaluate postfix
4. evaluate prefix
5. exit

Enter your choice : 2

Enter infix expression : 3+2\*(5-3)+9/3

prefix expression is : ++3\*2-53/93 1.

Infix to postfix expansion

1. infix to prefix expansion
2. Evaluate postfix
3. evaluate prefix 5. exit Enter your choice : 3

Enter the value 3 : 3

Enter the value 2 : 2

Enter the value 5 : 5

Enter the value 3 : 3

Enter the value 9 : 9 Enter the value 3 : 3 result = 10

1. Infix to postfix expansion
2. infix to prefix expansion
3. Evaluate postfix
4. evaluate prefix 5. exit

Enter your choice : 4

Enter the value 3 : 3

Enter the value 9 : 9

Enter the value 3 : 3

Enter the value 5 : 5 Enter the value 2 : 2 Enter the value 3 : 3 result=10

1. Infix to postfix expansion
2. infix to prefix expansion
3. Evaluate postfix
4. evaluate prefix
5. exit

Enter your choice : 5