- **Header File:**

```
/*
 * StackAdt.h
 *
 *  Created on: Nov 4, 2020
 *      Author: Megha Sonavane(23355)
 */

#ifndef STACKADT_H_
#define STACKADT_H_
//structure for stack as linked list
template<typename T>
struct Node{
	T symbol;
	Node<T>*next;
};
//class declaration
template<class T>
class StackAdt {
	Node<T>*top;
public:
	StackAdt();
	bool isEmpty();
	void push(T);
	T pop();
	T peep();
	void display();
	virtual ~StackAdt();
```

```
};

#endif /* STACKADT_H_ */
```

- **Implementation of header file:**

```
/*
 * StackAdt.cpp
 *
 *  Created on: Nov 4, 2020
 *      Author: Megha Sonavane(23355)
 */
#include<iostream>
#include "StackAdt.h"

using namespace std;
template<typename T>
StackAdt<T>::StackAdt() {
	top=NULL;
}
//-----------------------------------definition of isEmpty-----------------------------------------
template<typename T>
bool StackAdt<T>::isEmpty(){
	if(top==NULL)
		return true;
	return false;
}
//---------------------------------definition of push method-------------------------------------------
template<class T>
void StackAdt<T>::push(T symbol)
{
```

```
        Node<T>*ptr=new Node<T>;
        ptr->symbol=symbol;
        ptr->next=NULL;
        //if it is first node
        if(top==NULL)
        {
                top=ptr;
        }
        else{
                ptr->next=top;
                top=ptr;
        }
}
//----------------------------------definition of pop method-----------------------------------------------------
template<class T>
T StackAdt<T>::pop(){
        T c=top->symbol;
        top=top->next;
        return c;
}
//---------------------------------definition of peep method-----------------------------------------------------
template<class T>
T StackAdt<T>::peep(){
        return top->symbol;
}
//---------------------------------definition of display---------------------------------------------------------
template<class T>
void StackAdt<T>::display(){
        Node<T>*temp;
```

```
	temp=top;
	while(temp!=NULL){
		cout<<temp->symbol;
		temp=temp->next;
	}

}
template<class T>
StackAdt<T>::~StackAdt() {
	// TODO Auto-generated destructor stub
}
```

- **Main Implementation file:**

```
//============================================================================
// Name        : MockTest.cpp
// Author      : Megha Sonavane (23355)
// Date                  : Nov 4,2020
// Description : Conversion of infix to potfix and evaluation of postfix expression
//============================================================================

#include <iostream>
#include<cmath>
#include"StackAdt.h"
#include"StackAdt.cpp"
using namespace std;
//class declaration
class Expression{

public:
        bool isOperator(char);
        string toPostfix(string); //converting infix to postfix
        double evaluate_postfix(string);  //evaluating postfix
        double evaluate(double,double,char); //calculate result
        int precedence(char); //to check precedence of operator
};
//----------------------------definition of isOperator method--------------------------
bool Expression::isOperator(char c){
        if(c=='+'||c=='-'||c=='*'||c=='/'||c=='^')
                return true;
        return false;
```

```
}
//--------------------------------definition to check precedence of operator----------------
int Expression::precedence(char c)
{
	if(c=='^')
		return 3;
	else if(c=='*'||c=='/')
		return 2;
	else if(c=='+'||c=='-')
		return 1;
	return -1;
}
//-------------------------------------definition of method to convert expression into postfix------------------
string Expression::toPostfix(string infix){
	StackAdt<char>s;
	string postfix="";
	int len=infix.length();
	cout<<"-----------------------------------------------------------------------------------------------------"<<endl;
	cout<<"\tConversion:"<<endl<<"Scan"<<"\t"<<"Stack"<<"\t"<<"Expression"<<endl;
	for(int i=0;i<len;i++)
	{
		//-----------1.If it is operand----------------------
		if(isalpha(infix[i]))
			postfix+=infix[i];
		//---------2.If it is (--------------------------
		else if(infix[i]=='(')
			s.push(infix[i]);
		//--------3.If it is )-----------------------------
		else if(infix[i]==')')
```

```
{
        while((s.peep()!='(')&&(!s.isEmpty()))
        {
                postfix+=s.pop();
        }
        if(s.peep()=='('
                s.pop();
}
//-----------4.If it is operator
else if(isOperator(infix[i]))
{
        //----4.1.If stack is empty or contains ( at top---
        if((s.isEmpty())||(s.peep()=='('))
                s.push(infix[i]);
        //---4.2.If precedence of operator in expression is greater than that of operator in stack---
        else if(precedence(infix[i])>precedence(s.peep()))
                s.push(infix[i]);
        //---4.3. If the precedence of operator in expression is smaller than that of operator in stack---
        else{
                while((!s.isEmpty())&&( precedence(infix[i])<=precedence(s.peep())))
                {
                        postfix+=s.pop();
                }
                s.push(infix[i]);
        }
}
//------------else the expression is invalid----------------------
else{
        cout<<"*****Invalid Expression****";
```

```
                exit(1);
            }
            //-----------------display symbol scanned, current status of stack and expression----------------------
            cout<<infix[i]<<"\t";
            s.display();
            cout<<"\t"<<postfix<<endl;

        }
        while(!s.isEmpty())
            postfix+=s.pop();
        return postfix;
}

//-------------------------------------------definition of evaluate method-----------------------------------------
double Expression::evaluate(double a, double b,char op){
        switch(op){
        case '+':
            return (a+b);
            break;
        case '-':
            return (a-b);
            break;
        case '*':
            return (a*b);
            break;
        case '/':
            return (a/b);
            break;
        case '^':
```

```
                return (pow(a,b));
                break;
        default:
                cout<<"****Invalid values***";
                return 0;
        }
}
//-----------------------------------------------definition of expression evaluation-----------------------------------------------
double Expression::evaluate_postfix(string exp)
{
        double result;
        StackAdt<double>s;
        double op1,op2,val;
        int len=exp.length();
        for(int i=0;i<len;i++)
        {
                if(isalpha(exp[i])) //input the values of operands
                {
                        cout<<"Enter value of "<<exp[i]<<":";
                        cin>>val;
                        s.push(val); //push values to stack
                }
                else{
                        op1=s.pop(); //pop values from stack for calculation
                        op2=s.pop();
                        result=evaluate(op2,op1,exp[i]); //calculate result in evaluate method
                        s.push(result); //push result of operation to stack
                }
        }
```

```
        return result;
}

//-----------------------------------------------driver method-----------------------------------------------------
int main() {
        Expression e;
        string infix,postfix;
        double result;
        int ch;
        cout<<"\tEnter infix expression:";
        cin>>infix;
        do{
                cout<<"-------------------------------------------------------------------------------"<<endl;
                cout<<"\t1:To postfix"<<endl<<"\t2:To evaluate"<<endl<<"\t3:To enter new
expression"<<endl<<"\t4:Exit"<<endl<<"\tEnter choice:";
                cin>>ch;
                switch(ch){
                case 1:
                        //conversion from infix to postfix
                        postfix=e.toPostfix(infix);
                        cout<<"-----------------------------------------------------------"<<endl;
                        cout<<"\tPostfix Conversion:"<<postfix<<endl;
                        break;
                case 2:
                        //evaluating postfix expression
                        postfix=e.toPostfix(infix);  //converting infix to postfix
                        result=e.evaluate_postfix(postfix);
                        cout<<"-----------------------------------------------------------"<<endl;
                        cout<<"\tResult:"<<result<<endl;
```

```
                break;
            case 3:
                cout<<"\tEnter expression:";
                cin>>infix;
                break;
            case 4:
                cout<<"Thank you..."<<endl;
                break;
            default:
                cout<<"Invalid choice...";
            }
            //cout<<"--------------------------------------------------------------------------------------------"<<endl;
        }while(ch!=4);

        return 0;
}
```

- **Output:**

```
Enter infix expression:(a+b)*(c^d)
--------------------------------------------------------------------------------
1:To postfix
2:To evaluate
3:To enter new expression
4:Exit
Enter choice:1
--------------------------------------------------------------------------------------------------
Conversion:
Scan  Stack  Expression
(     (
a     (      a
+     +(     a
b     +(     ab
)            ab+
*     *      ab+
(     (*     ab+
c     (*     ab+c
^     ^(*    ab+c
d     ^(*    ab+cd
)     *      ab+cd^
------------------------------------------------------------
Postfix Conversion:ab+cd^*
--------------------------------------------------------------------------------
1:To postfix
2:To evaluate
3:To enter new expression
```

```
        4:Exit
        Enter choice:2
-----------------------------------------------------------------------------------------
        Conversion:
Scan    Stack   Expression
(       (
a       (       a
+       +(      a
b       +(      ab
)               ab+
*       *       ab+
(       (*      ab+
c       (*      ab+c
^       ^(*     ab+c
d       ^(*     ab+cd
)       *       ab+cd^
Enter value of a:12
Enter value of b:4
Enter value of c:3
Enter value of d:2
-----------------------------------------------------
        Result:144
-------------------------------------------------------------------------------
        1:To postfix
        2:To evaluate
        3:To enter new expression
        4:Exit
        Enter choice:4
Thank you...
```