```
Name - Shubham Yewalekar
Roll No - 23182
Batch - H9
Topic - Expression conversion using stack

File  - expression.h

#ifndef IMPLEMENT_H_
#define IMPLEMENT_H_

class expression
{
        private:
                char *exp;
                int exp_size;
                int postFix_size;
                char *postFix;
                char *preFix;
        public:
                expression();//constructor

                void setExp();//input infix expression
                void inToPostFix();//infix to postfix conversion
                void inToPreFix();//infix to prefix conversion

                float eval_postExp();//function to evaluate postfix expression
                float eval_preExp();//function to evaluate prefix expression

                int isOperator(char temp);//function to check if the entered character is a
n operator
                int isOperand(char temp);

                int isValid();//validations
                int isValid1();

                int priority(char check);//function to check priority of operators
                int associativity(char temp);//function to check associativity of operators

                float calculate(char temp,float o1,float o2);

                void printExp();
                void printPostFix();
                void printPreFix();

                ~expression();
};

#endif /* IMPLEMENT_H_ */

File - expression.cpp

#include <iostream>
#include "expression.h"
#include "stackll.cpp"
#include<math.h>
#include<string.h>
#include<iomanip>
using namespace std;

expression :: expression()
{
        exp = NULL;
        postFix = NULL;
        exp_size = 0;
        preFix = NULL;
        postFix_size = 0;
}
```

```
void expression :: setExp()
{
        cout<<"Enter the expression : ";
        int size;
        char *temp = new char[50];
        cin>>temp;
        size = strlen(temp);
        exp_size = size;
        exp = new char[exp_size];
        strcpy(exp,temp);
        delete []temp;
        if(isValid()==1)
        {
                if(isValid1()==1)
                {
                        cout<<"Expression Accepted Successfully\n";
                        printExp();
                }
                else
                {
                        cout<<"ERROR : INVALID INFIX EXPRESSION\n";
                        delete []exp;
                }
        }
        else
        {
                cout<<"ERROR : INVALID INFIX EXPRESSION\n";
                delete []exp;
        }
        int i=0;
        int count = exp_size;
        while(i<exp_size)
        {
                if(exp[i]=='(' || exp[i]==')')
                {
                        count--;
                }
                i++;
        }
        postFix_size = count;
        preFix = new char[postFix_size];
        postFix = new char[postFix_size];
}

void expression :: inToPostFix()
{
        cout<<"\nInput               Stack               Output              \n";
        stack_ll<char> stack;
        int counter = 0;
        int loop_pf = 0;
        char token;
        char topToken;
        char tokenOut;
        while(counter<exp_size)
        {
                token = exp[counter];
                cout<<setw(20)<<left<<token;
                if(stack.isEmpty()==0)
                {
                        cout<<setw(20)<<left<<stack.getTop();
                }
                else
                        cout<<setw(20)<<left<<"NONE";
                if(token == '(')
                {
                        stack.push(token);
                }
                else if(token == ')')
                {
                        token = stack.pop();
                        while(token != '(')
                        {
                                postFix[loop_pf] = token;
                                token = stack.pop();
                                loop_pf++;
                        }
                }
                else if(isOperator(token))
                {
                        topToken = stack.getTop();
                        while(stack.isEmpty() == 0 && priority(token)<priority(topToken))
                        {
                                tokenOut = stack.pop();
                                postFix[loop_pf] = tokenOut;
                                loop_pf++;
                                topToken = stack.getTop();
                        }
                        if(priority(token)>priority(topToken))
                        {
                                stack.push(token);
                        }
                        else if(priority(token) == priority(topToken))
                        {
                                        if((associativity(token) == 1) && (associativity(to
pToken)==1))
                                        {
                                                tokenOut = stack.pop();
                                                postFix[loop_pf] = tokenOut;
                                                loop_pf++;
                                                stack.push(token);
                                        }
                                        if((associativity(token) == 2) && (associativity(to
pToken)==2))
                                        {
                                                stack.push(token);
                                        }
                        }
                }
                else
                {
                        postFix[loop_pf] = token;
                        loop_pf++;
                }
                cout<<setw(20)<<left<<postFix<<endl;
                counter++;
        }
        while(stack.isEmpty() != 1)
        {
                cout<<setw(20)<<left<<"NONE";
                cout<<setw(20)<<left<<stack.getTop();
                token = stack.pop();
                postFix[loop_pf] = token;
                cout<<setw(20)<<left<<postFix<<endl;
                loop_pf++;
        }
}

float expression :: eval_postExp()
{
        stack_ll<float> stack;
        int loop_pf = 0;
        float ans;
        float temp1=-1,temp2=-1;
        while(loop_pf<postFix_size)
        {
                char token = postFix[loop_pf];
                if(isOperand(token))
                {
                        stack.push(token);
                }
                else if(isOperator(token))
                {
                        if(stack.isEmpty()==0)
                        {
                                float o1,o2;
                                float op1 = stack.pop();
                            if((char)op1 == '$')
                            {
                                op1 = stack.pop();
                                temp1 = op1;
                            }
                                float op2 = stack.pop();
                            if((char)op2 == '$')
                                {
                                op2 = stack.pop();
                                temp2 = op2;
                                }
                                if(isalpha((char)op1)!=0 && op1!=temp1)
                                {
                                        cout<<"Enter the data of variable "<<(char)op1<<" :
 ";
                                        cin>>o1;
                                }
                                else
                                {
                                        o1 = op1;
                                }
                                if(isalpha((char)op2)!=0 && op2!=temp2)
                                {
                                        cout<<"Enter the data of variable "<<(char)op2<<" :
 ";
                                        cin>>o2;
                                }
                                else
                                {
                                        o2 = op2;
                                }

                                ans = calculate(token,o2,o1);
                                stack.push(ans);
                                stack.push('$');
                        }
                        else
                                cout<<"Stack is empty";
                }
                loop_pf++;
        }
        float temp3 = stack.pop();
        return (stack.pop());
}

float expression :: eval_preExp()
{
        stack_ll<float> stack;
        int loop_pf = postFix_size-1;
        float ans;
        float temp1=-1,temp2=-1;
        while(loop_pf>=0)
        {
                char token = preFix[loop_pf];
                if(isOperand(token))
                {
                        stack.push(token);
                }
                else if(isOperator(token))
                {
                        if(stack.isEmpty()==0)
                        {
                                float o1,o2;
                                float op1 = stack.pop();
                            if((char)op1 == '$')
                            {
                                op1 = stack.pop();
                                temp1 = op1;
                            }
                                float op2 = stack.pop();
                                if((char)op2 == '$')
                                {
                                op2 = stack.pop();
                                temp2 = op2;
                                }
                                if(isalpha((char)op1)!=0 && op1!=temp1)
                                {
                                        cout<<"Enter the data of variable "<<(char)op1<<" :
 ";
                                        cin>>o1;
                                }
                                else
                                {
                                        o1 = op1;
                                }
                                if(isalpha((char)op2)!=0 && op2!=temp2)
                                {
                                        cout<<"Enter the data of variable "<<(char)op2<<" :
 ";
                                        cin>>o2;
                                }
                                else
                                {
                                        o2 = op2;
                                }

                                ans = calculate(token,o1,o2);
                                stack.push(ans);
                                stack.push('$');
                        }
                        else
                                cout<<"Stack is empty";
                }
                loop_pf--;
        }
        float temp3 = stack.pop();
        return (stack.pop());
}


int expression :: priority(char check)
{
        switch(check)
        {
                case '+' :
                        return 1;
                case '-' :
                        return 1;
                case '*' :
                        return 2;
                case '/' :
                        return 2;
                case '%' :
                        return 2;
                case '^' :
                        return 3;
                default :
                        return 0;
                        break;
        }
}

int expression :: isOperator(char temp)
{
        if(temp == '+' || temp == '-' || temp == '*' || temp == '/' || temp == '^' || temp
== '%')
        {
                return 1;
        }
        else
                return 0;
}

int expression :: isOperand(char temp)
{
        if(((float)temp>=65 && (float)temp<=90) || ((float)temp>=97 && (float)temp<=122))
        {
                return 1;
        }
        else
                return 0;
}

int expression :: associativity(char temp)
{
        switch(temp)
        {
                 case '+':
                         return 1;
                 case '-':
                         return 1;
                 case '*':
                         return 1;
                 case '/':
                         return 1;
                 case '%':
                         return 1;
                 case '^':
                         return 2;
                 default:
                         return 0;
        }
}

float expression :: calculate(char temp,float o1,float o2)
{
        switch(temp)
        {
                case '+':
                        return o1 + o2;
                case '-':
                        return o1 - o2;
                case '*':
                        return o1*o2;
                case '/':
                        return o1/o2;
                case '^' :
                        return pow(o1,o2);
                case '%' :
                        return o1/o2;
                default :
                        return 0;
        }
}
```

```
void expression :: inToPreFix()
{
        stack_ll<char> stack;
        int loop = exp_size-1;
        int loop_pf = postFix_size-1;
        char token;
        cout<<"\nInput               Stack               Output              \n";
        while(loop>=0)
        {
                token = exp[loop];
                cout<<setw(20)<<left<<token;

                if(stack.isEmpty()==0)
                {
                                cout<<setw(20)<<left<<stack.getTop();
                }
                else
                        cout<<setw(20)<<left<<"NONE";

                if(isOperand(token))
                {
                        preFix[loop_pf] = token;
                        loop_pf--;
                }
                else if(token == ')')
                {
                        stack.push(token);
                }
                else if(token == '(')
                {
                        char tempToken;
                        tempToken = stack.pop();
                        while(tempToken != ')')
                        {
                                preFix[loop_pf] = tempToken;
                                loop_pf--;
                                tempToken = stack.pop();
                        }
                }
                else if(isOperator(token))
                {
                        char topToken = stack.getTop();
                        if(priority(token)>priority(topToken))
                        {
                                stack.push(token);
                        }
                        else if(priority(token)<priority(topToken))
                        {
                                while(priority(token)<priority(topToken) && stack.isEmpty()
==0)

                                {
                                        char temp = stack.pop();
                                        preFix[loop_pf] = temp;
                                        loop_pf--;
                                        topToken = stack.getTop();
                                }
                                stack.push(token);
                        }
                        else if(priority(token) == priority(topToken))
                        {
                                if(associativity(token)==1 && associativity(topToken)==1)
                                {
                                        stack.push(token);
                                }
                                else if(associativity(token)==2 && associativity(topToken)=
=2)

                                {
                                        char temp = stack.pop();
                                        preFix[loop_pf] = temp;
                                        stack.push(token);
                                        loop_pf--;
                                }
                        }
                }
                cout<<setw(20)<<left<<preFix<<endl;
                loop--;
        }
        while(stack.isEmpty()==0)
        {
                cout<<setw(20)<<left<<"NONE";
                cout<<setw(20)<<left<<stack.getTop();
                token = stack.pop();
                preFix[loop_pf] = token;
                cout<<setw(20)<<left<<preFix<<endl;
                loop_pf--;
        }
}


void expression :: printPostFix()
{
        int count ;
        cout<<"\nPOSTFIX Expression is : ";
        for(count = 0;count<postFix_size;count++)
        {
                cout<<postFix[count];
        }
        cout<<endl;
}

void expression :: printPreFix()
{
        int count ;
        cout<<"\nPREFIX Expression is  : ";
        for(count = 0;count<postFix_size;count++)
        {
                cout<<preFix[count];
        }
        cout<<endl;
}

void expression :: printExp()
{
        int count ;
        cout<<"Entered Expression is : ";
        for(count = 0;count<exp_size;count++)
        {
                cout<<exp[count];
        }
        cout<<endl;
}


int expression :: isValid()
{
        int cnt_openb=0;
        int cnt_closeb=0;
        int cnt_operand=0;
        int cnt_operator =0;
        if(exp == NULL)
        {
                cout<<"Expression not entered\nPlease enter the expression first\n";
                return 0;
        }
        else
        {
                int loop_pf=0;
                while(loop_pf<exp_size)
                {
                        if(exp[loop_pf]==')')
                        {
                                cnt_closeb++;
                        }
                        else if(exp[loop_pf] == '(')
                        {
                                cnt_openb++;
                        }

                        if(isOperand(exp[loop_pf])==1)
                        {
                                cnt_operand++;
                        }
                        else if(isOperator(exp[loop_pf])==1)
                        {
                                cnt_operator++;
                        }

                        if(isOperand(exp[loop_pf])||isOperator(exp[loop_pf]) || exp[loop_pf
] == '(' || exp[loop_pf] == ')')
                        {
                                loop_pf++;
                        }
                        else
                                break;
                }
                if((loop_pf == exp_size) && (cnt_openb == cnt_closeb) && (cnt_operand == (c
nt_operator+1)))
                {
                        return 1;
                }
                else
                        return 0;
        }
}

int expression :: isValid1()
{
        int loop_pf = 1;
        while(loop_pf<exp_size)
        {
                if(isOperand(exp[loop_pf-1]) && isOperand(exp[loop_pf]))
                {
                        return 0;
                }
                if(isOperator(exp[loop_pf-1]) && isOperator(exp[loop_pf]))
                {
                        return 0;
                }
                loop_pf++;
        }
        return 1;
}

expression :: ~expression()
{
        delete []exp;
        delete []postFix;
        delete []preFix;
}

File - stack.h

#include "linkedL.cpp"
```

```
#ifndef STACKLL_H_
#define STACKLL_H_

template<class T>
class stack_ll
{
        private:
                linkedL<T> L;
                Node<T> *top;
        public:
                stack_ll();
                void push(T dat);//function to insert data at top of stack
                T pop();//function to get data from top of the stack
                T getTop();//function to check the top of the stack
                int isEmpty();
};

#endif /* STACKLL_H_ */

File - stack.cpp

#include "stackll.h"
#include "linkedL.h"
#include<iostream>
using namespace std;

template<class T>
stack_ll<T>::stack_ll()
{
        top = NULL;

}

template<class T>
void stack_ll<T>::push(T dat)
{
        L.insert(1,dat);
        top = L.getHead();
}

template<class T>
T stack_ll<T>::pop()
{
        T temp = top->data;
        L.delete_node(top->data);
        top = L.getHead();
        return temp;
}

template<class T>
T stack_ll<T>::getTop()
{
        if(isEmpty()==0)
        {
                return top->data;
        }
}

template<class T>
int stack_ll<T>::isEmpty()
{
        if(top == NULL)
        {
                return 1;
        }
        else
                return 0;
}
```

File - linkedL.h

```
#ifndef LINKEDL_H_
#define LINKEDL_H_

template<class T>
class Node
{
	public:
		T data;
		Node *next;
};

template<class T>
class linkedL
{
	private:
		Node<T> *head;
		int n_o_n;
	public:
		linkedL();
		void create();//function to create a linked list having required no of node
s
		void display();//function to display all the nodes present in the linked li
st
		void modify(T key);//function to search the node by data stored in it and m
odify that data
		int search(T key);//function to search the node based on data stored in the
 node
		void insert(int pos,T dat);//function to insert new node in the linked list
		void display_rev(Node<T> *temp,int check);//function to display linked list
 in reverse order
		void delete_node(T key);//function to delete
		void revert_list();
		Node<T>* getHead();
};

#endif /* LINKEDL_H_ */
```

File - linkedL.cpp

```
#include "linkedL.h"
#include<iostream>
using namespace std;

template<class T>
linkedL<T> :: linkedL()
{
	head = NULL;
	n_o_n = 0;
}

template<class T>
void linkedL<T> :: create()
{
	int flag = 1;
	Node<T> *temp;
	do
	{
		Node<T> *node1 = new Node<T>;
		cout<<"Enter the data of the node: ";
		cin>>node1->data;
		n_o_n++;
		if(head==NULL)
		{
			head = node1;
			node1->next = NULL;
```

```
                        temp = head;
                }
                else
                {
                        temp->next = node1;
                        node1->next = NULL;
                        temp = temp->next;
                }
                cout<<"Do you want to add another node (Y-1/N-0) : ";
                cin>>flag;
        }while(flag==1);
}

template<class T>
void linkedL<T> :: display()
{
        Node<T> *temp = head;
        cout<<"Linked List Created is as follows : \n";
        while(temp!=NULL)
        {
                cout<<temp->data<<"|"<<temp->next;
                if(temp->next!=NULL)
                {
                        cout<<"-->";
                }
                temp = temp->next;
        }
        cout<<endl;
}

template<class T>
void linkedL<T> :: modify(T key)
{
        Node<T>* temp = head;
        while(temp->data!=key && temp != NULL)
        {
                        temp = temp->next;
        }
        if(temp==NULL)
        {
                cout<<"Node not found.PLease enter valid node data."<<endl;
        }
        else
        {
                cout<<"Enter the modified data to be stored in the node : ";
                cin>>temp->data;
        }
}

template<class T>
int linkedL<T> :: search(T key)
{
        int count = 0;
        Node<T> *temp = head;
        while(temp->data!=key && temp!=NULL)
        {
                temp = temp->next;
                count++;
        }
        if(temp == NULL)
        {
                return -1;
        }
        else
        {
                return count+1;
        }
}
```

```
template<class T>
void linkedL<T> :: insert(int pos,T dat)
{
        Node<T> *node1 = new Node<T>;
        node1->data = dat;
        if(head == NULL)
        {
                head = node1;
                n_o_n++;
        }
        else
        {
                if(pos==1)
                {
                        node1->next = head;
                        head = node1;
                }
                else if(pos>n_o_n)
                {
                        Node<T> *temp = head;
                        while(temp->next!=NULL)
                        {
                                temp = temp->next;
                        }
                        temp->next = node1;
                        node1->next = NULL;
                }
                else
                {
                        Node<T> *temp = head;
                        int i;
                        for(i=0;i<pos-2;i++)
                        {
                                temp = temp->next;
                        }
                        node1->next = temp->next;
                        temp->next = node1;
                }
        }
}

template<class T>
void linkedL<T> :: display_rev(Node<T> *temp,int check)
{
        if(check == 0)
        {
                temp = head;
                display_rev(temp->next,1);
        }
        else
        {
                if(temp->next!=NULL)
                {
                        display_rev(temp->next,1);
                }
        }
        cout<<temp->data<<endl;
}

template<class T>
void linkedL<T> :: delete_node(T key)
{
        Node<T> *temp = head;
        Node<T> *prev = NULL;
        while(temp!=NULL && temp->data!=key)
        {
                prev = temp;
                temp = temp->next;
        }
        if(temp==NULL)
        {
                cout<<"Node not found."<<endl;
        }
        else if(temp == head)
        {
                head = temp->next;
                delete temp;
        }
        else if(temp->next==NULL)
        {
                prev->next = NULL;
                delete temp;
        }
        else
        {
                prev->next = temp->next;
                delete temp;
        }
}

template<class T>
void linkedL<T> :: revert_list()
{
        Node<T> *prev = NULL;
        Node<T> *temp = head;
        Node<T> *next = temp->next;
        while(temp!=NULL)
        {
                temp->next = prev;
                prev = temp;
                temp = next;
                if(next!=NULL)
                {
                        next = temp->next;
                }
        }
        head = prev;
}

template<class T>
Node<T>* linkedL<T> :: getHead()
{
        return head;
}

File - Implement.cpp

#include <iostream>
#include "expression.h"
using namespace std;

int main()
{
        expression e;
        int ch,flag,flag1=0;
        float ans;
        do
        {
                cout<<"\n\n********************\n";
                cout<<"WELCOME TO THE MENU ! \n";
                cout<<"********************\n";

                cout<<"Which operation do you want to perform :\n               1)Input Inf
ix Expression      - Press 1\n          2)Infix to Postfix Conversion - Press 2\n
3)Infix to Prefix Conversion  - Press 3\n               4)Evaluate PostFix           - Pre
```

```
ss 4\n          5)Evaluate PreFix             - Press 4\n";
                cout<<"Choice : ";
                cin>>ch;
                switch(ch)
                {
                        case 1 :
                                cout<<"\n<<<<<<<< INPUT INFIX EXPRESSION >>>>>>>>\n";
                                e.setExp();
                                cout<<"\n----------------------------------------\n";
                                break;
                        case 2 :
                                cout<<"\n<<<<<<<< INFIX TO POSTFIX EXPRESSION >>>>>>>>\n";
                                e.inToPostFix();
                                e.printPostFix();
                                flag1 = 1;
                                cout<<"\n--------------------------------------------\n";
                                break;
                        case 3 :
                                cout<<"\n<<<<<<<< INFIX TO PREFIX EXPRESSION >>>>>>>>\n";
                                e.inToPreFix();
                                e.printPreFix();
                                cout<<"\n--------------------------------------------\n";
                                break;
                        case 4 :
                                cout<<"\n<<<<<<<< EVALUATE EXPRESSION >>>>>>>>\n\n";
                                ans = e.eval_postExp();
                                cout<<"Answer : "<<ans;
                                cout<<"\n\n--------------------------------------\n";
                                break;
                        case 5 :
                                cout<<"\n<<<<<<<< EVALUATE EXPRESSION >>>>>>>>\n\n";
                                ans = e.eval_preExp();
                                cout<<"Answer : "<<ans;
                                cout<<"\n\n--------------------------------------\n";
                                break;
                        default :
                                cout<<"Invalid Input\n";
                                break;
                }
                cout<<"\nDo you want to continue (Yes - Press 1 / No - Press 0) : ";
                cin>>flag;
        }while(flag==1);
        return 0;

}

OUTPUT -


Test Case 1: ((a+b)+c*(d/e))+f

*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 1

<<<<<<<< INPUT INFIX EXPRESSION >>>>>>>>
Enter the expression : ((a+b)+c*(d/e))+f
Expression Accepted Successfully
Entered Expression is : ((a+b)+c*(d/e))+f

----------------------------------------
```

```
Do you want to continue (Yes - Press 1 / No - Press 0) : 1


********************
WELCOME TO THE MENU !
********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 2

<<<<<<<< INFIX TO POSTFIX EXPRESSION >>>>>>>>

Input               Stack               Output
(                   NONE
(                   (
a                   (                   a
+                   (                   a
b                   +                   ab
)                   +                   ab+
+                   (                   ab+
c                   +                   ab+c
*                   +                   ab+c
(                   *                   ab+c
d                   (                   ab+cd
/                   (                   ab+cd
e                   /                   ab+cde
)                   /                   ab+cde/
)                   *                   ab+cde/*+
+                   NONE                ab+cde/*+
f                   +                   ab+cde/*+f
NONE                +                   ab+cde/*+f+

POSTFIX Expression is : ab+cde/*+f+

--------------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


********************
WELCOME TO THE MENU !
********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 3

<<<<<<<< INFIX TO PREFIX EXPRESSION >>>>>>>>

Input               Stack               Output
f                   NONE
+                   NONE
)                   +
)                   )
e                   )
/                   )
d                   /
(                   /
*                   )
c                   *
```

```
+                    *
)                    +
b                    )
+                    )
a                    +
(                    +
(                    +
NONE                 +                   +++ab*c/def

PREFIX Expression is  : +++ab*c/def

--------------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 4

<<<<<<<< EVALUATE EXPRESSION >>>>>>>>

Enter the data of variable b : 1
Enter the data of variable a : 2
Enter the data of variable e : 2
Enter the data of variable d : 1
Enter the data of variable c : 2
Enter the data of variable f : 1
Answer : 5

-------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 5

<<<<<<<< EVALUATE EXPRESSION >>>>>>>>

Enter the data of variable d : 1
Enter the data of variable e : 2
Enter the data of variable c : 2
Enter the data of variable a : 2
Enter the data of variable b : 1
Enter the data of variable f : 1
Answer : 5

-------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 0

Test Case 2 : a*(b+c)-d/e
```

```
*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 1

<<<<<<<< INPUT INFIX EXPRESSION >>>>>>>>
Enter the expression : a*(b+c)-d/e
Expression Accepted Successfully
Entered Expression is : a*(b+c)-d/e

----------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 2

<<<<<<<< INFIX TO POSTFIX EXPRESSION >>>>>>>>

Input               Stack               Output
a                   NONE                a
*                   NONE                a
(                   *                   a
b                   (                   ab
+                   (                   ab
c                   +                   abc
)                   +                   abc+
-                   *                   abc+*
d                   -                   abc+*d
/                   -                   abc+*d
e                   /                   abc+*de
NONE                /                   abc+*de/
NONE                -                   abc+*de/-

POSTFIX Expression is : abc+*de/-

--------------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 3
```

```
<<<<<<<< INFIX TO PREFIX EXPRESSION >>>>>>>>

Input               Stack               Output
e                   NONE
/                   NONE
d                   /
-                   /
)                   -
c                   )
+                   )
b                   +
(                   +
*                   -
a                   *
NONE                *
NONE                -                   -*a+bc/de

PREFIX Expression is  : -*a+bc/de

---------------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 4

<<<<<<<< EVALUATE EXPRESSION >>>>>>>>

Enter the data of variable c : 2
Enter the data of variable b : 1
Enter the data of variable a : 2
Enter the data of variable e : 2
Enter the data of variable d : 1
Answer : 5.5

--------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 5

<<<<<<<< EVALUATE EXPRESSION >>>>>>>>

Enter the data of variable d : 1
Enter the data of variable e : 2
Enter the data of variable b : 1
Enter the data of variable c : 2
Enter the data of variable a : 2
Answer : 5.5
```

```
-------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 0

Test Case 3 : ((a+b)*(c+d)/(e-f))+g

*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 1

<<<<<<<< INPUT INFIX EXPRESSION >>>>>>>>
Enter the expression : ((a+b)*(c+d)/(e-f))+g
Expression Accepted Successfully
Entered Expression is : ((a+b)*(c+d)/(e-f))+g

----------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 2

<<<<<<<< INFIX TO POSTFIX EXPRESSION >>>>>>>>

Input               Stack               Output
(                   NONE
(                   (
a                   (                   a
+                   (                   a
b                   +                   ab
)                   +                   ab+
*                   (                   ab+
(                   *                   ab+
c                   (                   ab+c
+                   (                   ab+c
d                   +                   ab+cd
)                   +                   ab+cd+
/                   *                   ab+cd+*
(                   /                   ab+cd+*
e                   (                   ab+cd+*e
-                   (                   ab+cd+*e
f                   -                   ab+cd+*ef
)                   -                   ab+cd+*ef-
)                   /                   ab+cd+*ef-/
+                   NONE                ab+cd+*ef-/
g                   +                   ab+cd+*ef-/g
NONE                +                   ab+cd+*ef-/g+

POSTFIX Expression is : ab+cd+*ef-/g+

--------------------------------------------
```

```
Do you want to continue (Yes - Press 1 / No - Press 0) : 1


********************
WELCOME TO THE MENU !
********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 3

<<<<<<<< INFIX TO PREFIX EXPRESSION >>>>>>>>

Input               Stack               Output
g                   NONE
+                   NONE
)                   +
)                   )
f                   )
-                   )
e                   -
(                   -
/                   )
)                   /
d                   )
+                   )
c                   +
(                   +
*                   /
)                   *
b                   )
+                   )
a                   +
(                   +
(                   *
NONE                +                   +/*+ab+cd-efg

PREFIX Expression is  : +/*+ab+cd-efg

--------------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


********************
WELCOME TO THE MENU !
********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 4

<<<<<<<< EVALUATE EXPRESSION >>>>>>>>

Enter the data of variable b : 1
Enter the data of variable a : 2
Enter the data of variable d : 1
Enter the data of variable c : 2
Enter the data of variable f : 1
Enter the data of variable e : 2
Enter the data of variable g : 1
```

```
Answer : 10

-------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 5

<<<<<<<< EVALUATE EXPRESSION >>>>>>>>

Enter the data of variable e : 2
Enter the data of variable f : 1
Enter the data of variable c : 2
Enter the data of variable d : 1
Enter the data of variable a : 2
Enter the data of variable b : 1
Enter the data of variable g : 1
Answer : 10

-------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 0

Test Case 4: a+b^c^d-(e*f)-g

*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 1

<<<<<<<< INPUT INFIX EXPRESSION >>>>>>>>
Enter the expression : a+b^c^d-(e*f)-g
Expression Accepted Successfully
Entered Expression is : a+b^c^d-(e*f)-g

----------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 2

<<<<<<<< INFIX TO POSTFIX EXPRESSION >>>>>>>>
```

```
Input                   Stack                   Output
a                       NONE                    a
+                       NONE                    a
b                       +                       ab
^                       +                       ab
c                       ^                       abc
^                       ^                       abc
d                       ^                       abcd
-                       ^                       abcd^^+
(                       -                       abcd^^+
e                       (                       abcd^^+e
*                       (                       abcd^^+e
f                       *                       abcd^^+ef
)                       *                       abcd^^+ef*
-                       -                       abcd^^+ef*-
g                       -                       abcd^^+ef*-g
NONE                    -                       abcd^^+ef*-g-

POSTFIX Expression is : abcd^^+ef*-g-

--------------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 3

<<<<<<<< INFIX TO PREFIX EXPRESSION >>>>>>>>

Input                   Stack                   Output
g                       NONE
-                       NONE
)                       -
f                       )
*                       )
e                       *
(                       *
-                       -
d                       -
^                       -
c                       ^
^                       ^
b                       ^
+                       ^
a                       +
NONE                    +
NONE                    -
NONE                    -                       --+a^b^cd*efg

PREFIX Expression is  : --+a^b^cd*efg

--------------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
```

```
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 4

<<<<<<<< EVALUATE EXPRESSION >>>>>>>>

Enter the data of variable d : 1
Enter the data of variable c : 2
Enter the data of variable b : 1
Enter the data of variable a : 2
Enter the data of variable f : 1
Enter the data of variable e : 2
Enter the data of variable g : 1
Answer : 0

-------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 5

<<<<<<<< EVALUATE EXPRESSION >>>>>>>>

Enter the data of variable e : 2
Enter the data of variable f : 1
Enter the data of variable c : 2
Enter the data of variable d : 1
Enter the data of variable b : 1
Enter the data of variable a : 2
Enter the data of variable g : 1
Answer : 0

-------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 0

Test Case 5 : b-c-d-f*a*e

*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 1

<<<<<<<< INPUT INFIX EXPRESSION >>>>>>>>
Enter the expression : b-c-d-f*a*e
Expression Accepted Successfully
Entered Expression is : b-c-d-f*a*e
```

```
----------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 2

<<<<<<<< INFIX TO POSTFIX EXPRESSION >>>>>>>>

Input               Stack               Output
b                   NONE                b
-                   NONE                b
c                   -                   bc
-                   -                   bc-
d                   -                   bc-d
-                   -                   bc-d-
f                   -                   bc-d-f
*                   -                   bc-d-f
a                   *                   bc-d-fa
*                   *                   bc-d-fa*
e                   *                   bc-d-fa*e
NONE                *                   bc-d-fa*e*
NONE                -                   bc-d-fa*e*-

POSTFIX Expression is : bc-d-fa*e*-

--------------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 3

<<<<<<<< INFIX TO PREFIX EXPRESSION >>>>>>>>

Input               Stack               Output
e                   NONE
*                   NONE
a                   *
*                   *
f                   *
-                   *
d                   -
-                   -
c                   -
-                   -
b                   -
NONE                -
NONE                -
NONE                -                   ---bcd**fae

PREFIX Expression is  : ---bcd**fae

--------------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 4

<<<<<<<< EVALUATE EXPRESSION >>>>>>>>

Enter the data of variable c : 2
Enter the data of variable b : 1
Enter the data of variable d : 1
Enter the data of variable a : 2
Enter the data of variable f : 1
Enter the data of variable e : 2
Answer : -6

-------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 1


*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 5

<<<<<<<< EVALUATE EXPRESSION >>>>>>>>

Enter the data of variable f : 1
Enter the data of variable a : 2
Enter the data of variable e : 2
Enter the data of variable b : 1
Enter the data of variable c : 2
Enter the data of variable d : 1
Answer : -6

-------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 0

Test Case 6 : (a+b)-)c*d

*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 1

<<<<<<<< INPUT INFIX EXPRESSION >>>>>>>>
Enter the expression : (a+b)-)c*d
ERROR : INVALID INFIX EXPRESSION

----------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 0

Test Case 7 : (a+b+)*c/d/f-

*********************
WELCOME TO THE MENU !
*********************
Which operation do you want to perform :
                1)Input Infix Expression      - Press 1
                2)Infix to Postfix Conversion - Press 2
                3)Infix to Prefix Conversion  - Press 3
                4)Evaluate PostFix            - Press 4
                5)Evaluate PreFix             - Press 4
Choice : 1

<<<<<<<< INPUT INFIX EXPRESSION >>>>>>>>
Enter the expression : (a+b+)*c/d/f-
ERROR : INVALID INFIX EXPRESSION

----------------------------------------

Do you want to continue (Yes - Press 1 / No - Press 0) : 0
```