**Name : Resham Landge**
**Roll No : 2339**

********** **Assignment No : 1** **********

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

**********

```
#include<iostream>
#include<cstring>
#include<stdlib.h>
#include<stdio.h>
#include<ctype.h>

using namespace std;

class node                              // class to create node
{
        public:
                char data;
                node *next;
};

class stack                             // class to declare different function for stack operation
{
        public:
                node  *top;

                stack()
                {
                  top = NULL;
                }

                int empty();
                void push(char );
                char pop();
                void dis();
                char Top();
};

char stack :: Top()                     // return to element without popping
{
        node *p = top;

        if( empty() == 1 )
                 return -1;
        else
                return p->data;
}

int stack :: empty()                    // function determine whether stack is empty or not
{
```

```
        if( top == NULL )
                return 1;
        else
                return 0 ;
}


void stack :: push(char x)              //  function to insert the element to the stack
{
        node *p;
        p = new node;
        p->data = x;
        p->next = top;
        top = p;
}

char stack :: pop()                     // function to delete the element from stack
{
        char x;
        node *p;

        if( empty() == 1 )
                return -1;
        else
        {
                p = top;
                top = top->next;
                x = p->data;
                delete p;
                return x;
        }
}

void stack :: dis()                     // function to display the stack element
{
        node *p = top;

        while(p! = NULL)
        {
                 cout<<p->data;
                 p = p->next;
        }
}

class convert
{
        public:                         //Declares all members are public
                char in[200],  po[200], pe[200];
                int c;

        void infixtopostfix(char in[],char po[]);   // declare function for infix to postfix conversion
        void infixtoprefix(char in[],char pe[]);    // declare function for infix to prefix conversion
        void prefix(char in[],char pe[]);
        void Epostfix(char po[]);                   // declares function for postfix evaluation
        void Eprefix(char pe[]);                    // declares function for prefix evaluation
```

```
        int evaluate(char,int,int);                 //
        int pri(char);                              // To set the priority of the operator

         void exp()
        {
                    do
                    {
                        cout<<" \n\t 1. Infix To Postfix \n\t 2. Infix To Prefix \n\t 3. Postfix
                                Evalution \n\t 4. Prefix Evaluation \n\t 5. Exit \n";
                        cout<<"\n\t Enter Your choice : ";
                        cin>>c;

                        switch(c)
                        {
                            case 1 :
                                    cout<<"  \n Enter expression for convert to Postfix  -->>   ";
                                    cin>>in;

                                    infixtopostfix(in,po);
                                    cout<<" \n\t Infix :->> "<<in;
                                    cout<<" \n\t Postfix->> "<<po<<"\n\n";
                                    break;

                            case 2 :
                                    cout<<" \n Enter expression for convert to Prefix  -->>  ";
                                    cin>>in;

                                    infixtoprefix(in,pe);
                                    cout<<" \n\t Infix :->> "<<in;
                                    cout<<" \n\t Prefix :->> "<<pe<<"\n\n";
                                    break;

                            case 3 :
                                    cout<<" \n Enter Postfix expression to Evalute  -->>  ";
                                    cin>>po;

                                    Epostfix(po);
                                    break;

                            case 4 :
                                    cout<<" \n Enter Postfix expression to Evalute  -->>  ";
                                    cin>>pe;

                                    Eprefix(pe);
                                    break;

                            case 5 :
                                    break;

                            default :
                                    cout<<" \n\n\t !!!...Invalid Choice…!!! ";
                        }
                    } while (c!=5);
         }
};
```

```
void convert :: Epostfix(char po[])                    // function for postfix evaluation
{
        stack s;                                       //object of stack is being created
        int i, ch1, ch2, n;
        char x;

        for(i =0 ; po[i]!='\0' ;  i++)
        {
                x = po[i];

                if(isalpha(x))                         //push the operand
                {
                    cout<<" \n\t Enter the value of "<<x<<" : ";
                    cin>>n;

                    s.push(n);
                }
                else if(isdigit(x))                    // if digit then push into stack
                    s.push(x-48);
                else
                {
                    ch2 = s.pop();                     //operator pops two operand
                    ch1 = s.pop();
                    n = evaluate(x,ch1,ch2);           // perform operation on the given expression
                    s.push(n);
                }
        }
        n = s.pop();
        cout<<" \n\n\t Value of Expression = "<<n<<"\n\n";
}

void convert :: Eprefix(char pe[])                     //function for prefix evaluation
{
        stack s;                                       //object of stack is being created
        int i, ch1, ch2, n;
        char x;

        for(i=strlen(pe)-1 ; i>=0 ; i--)
        {
                x = pe[i];

                if(isalpha(x))                         //push the operand
                {
                    cout<<" \n\t Enter the value of "<<x<<" : ";
                    cin>>n;

                    s.push(n);
                }
                else if(isdigit(x))
                    s.push(x-48);
                else
                {
                                                       //pops two operands
                    ch1 = s.pop();
                    ch2 = s.pop();
```

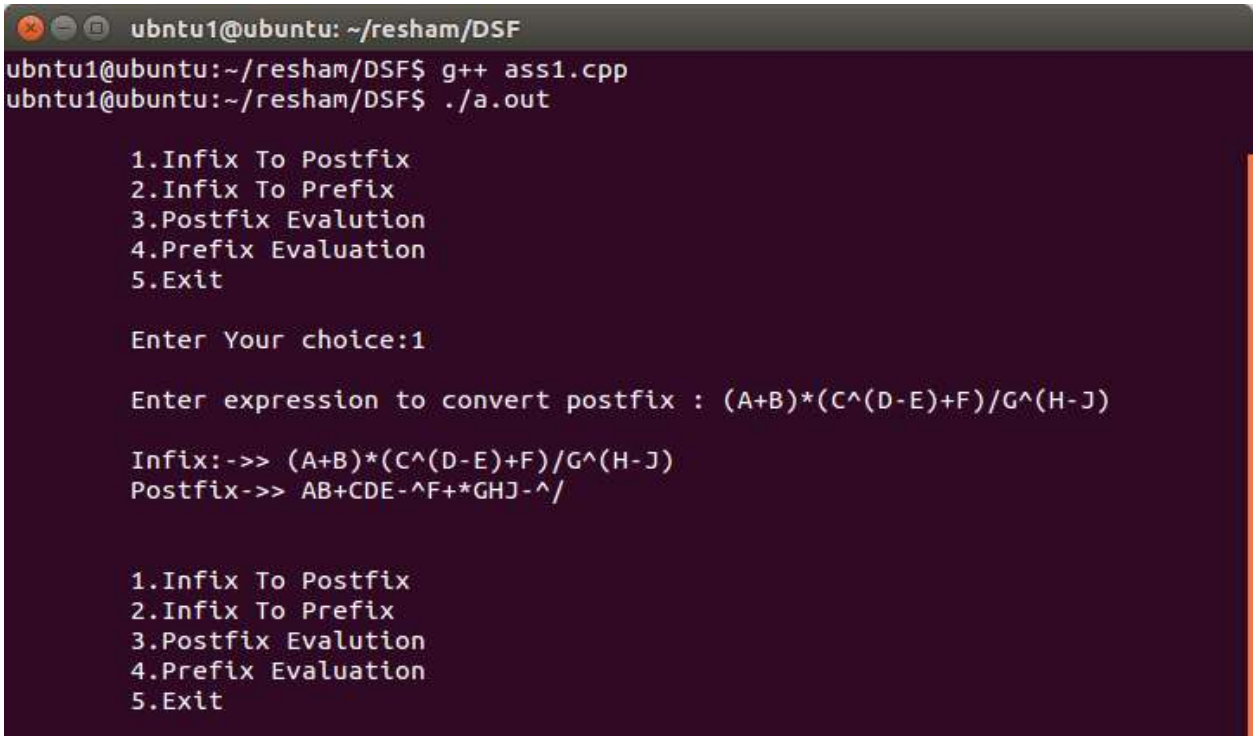
```
                    n = evaluate(x,ch1,ch2);
                    s.push(n);
                }
        }
        n = s.pop();
        cout<<" \n\n\t Value of Expression = "<<n<<"\n\n";
}

int  convert :: evaluate(char x, int ch1, int ch2)
{
        if(x=='+')
                return(ch1+ch2);
        if(x=='-')
                return(ch1-ch2);
        if(x=='*')
                return(ch1*ch2);
        if(x=='/')
                return(ch1/ch2);
        if(x=='%')
                return(ch1%ch2);
        if(x=='^')
        {
          int i,n=1;

          for(i=1 ; i<=ch2 ; i++)
                n=ch1*n;

          return(n);
        }
        if(x=='$')
        {
          int i,n=1;

          for(i=1 ; i<=ch2 ; i++)
                n=ch1*n;

          return(n);
        }
        if(x=='#')
        {
          int i,n=1;

          for(i=1 ; i<=ch2 ; i++)
                n=ch1*n;

          return(n);
        }
}

void convert :: infixtopostfix(char in[],char po[])
{
                stack s;                        //object of stack is being ctreated
                int i=0, k=0, m, n, a=0, b;     //i-index for infix[],k-index for postfix[]
                char ch, x;
                n=strlen(in);
```

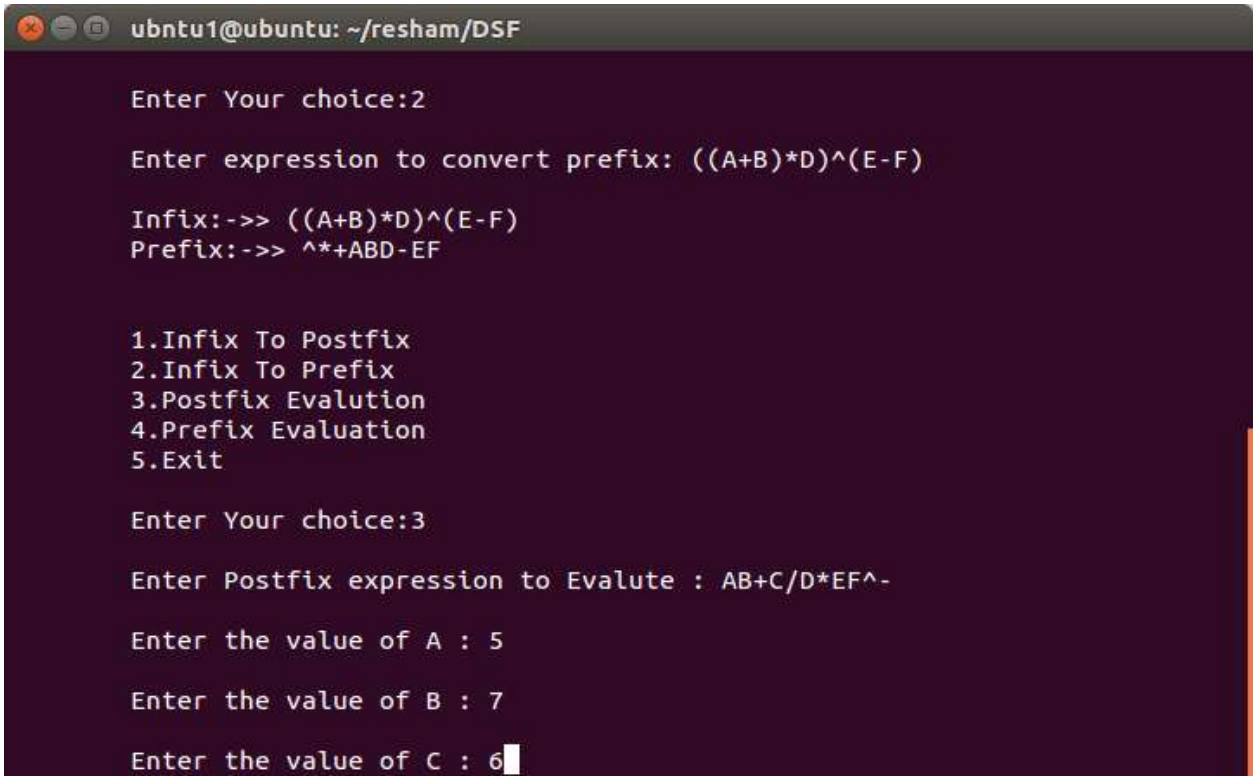
```
		for(i=0 ; in[i]!='\0' ; i++)
		{
			ch = in[i];

			if(isalnum(ch))
				po[k++]=ch;
			else
			{
				if(ch=='(')
					s.push(ch);
				else
				{
					if(ch==')')
					{
						while((x=s.pop())!='(')  //pop from stack till ( occurs
							po[k++]=x;
					}
					else
					{
						while(pri(ch)<=pri(s.Top()) && !s.empty())
						{
							x = s.pop();
							po[k++] = x;
						}
						s.push(ch);
					}
				}
			}
			po[k] = '\0';			//make po[k] as valid string
		}
		while(!s.empty())
		{
			x = s.pop();
			po[k++] = x ;
		}
		po[k++] = '\0';
}
void convert :: infixtoprefix(char in[],char pe[])
{
	stack s;
	int i=0, k=0;
	char ch, temp[200];

							//Reverse the infix expression and store it in temp[]
	for(i=strlen(in)-1 ; k>=0, i>=0 ; i--,k++)
		temp[k] = in[i];

	temp[k] = '\0';
							//reverse the direction of brackets
	for(i=0 ; temp[i]!='\0' ; i++)
	{
		if(temp[i]=='(')
			temp[i]=')';
		else
			if(temp[i]==')')
```

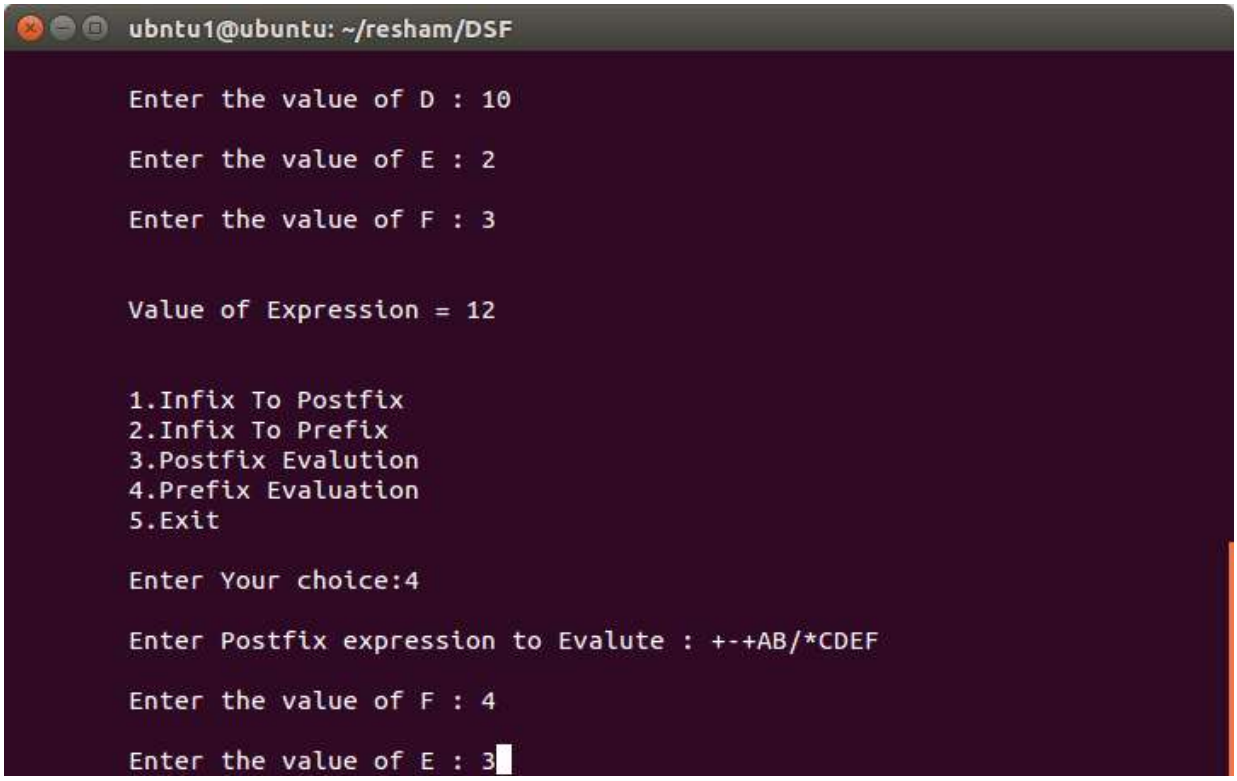
```
                                        temp[i]='(';
        }
                                                //convert from infix to postfix
        prefix(temp,pe);
                                                //reverse the final expression
        for(i=0, k=strlen(pe)-1 ; i<k ; i++, k--)
        {
                ch = pe[i];
                pe[i] = pe[k];
                pe[k] = ch;
        }
}

void convert :: prefix(char in[],char pe[])
{
        stack s;
        int i=0, k=0, m, n, a=0, b;
        char ch, x;
        n=strlen(in);                           //the length of string stored in variable n

        for(i=0 ; in[i]!='\0' ; i++)
        {
                ch = in[i];

                if(isalnum(ch))
                        pe[k++] = ch;
                else
                {
                        if(ch=='(')
                                s.push(ch);
                        else
                        {
                                if(ch==')')
                                {
                                        while((x=s.pop())!='(')   //pop the operand till ( occurs
                                                pe[k++] = x;
                                }
                                else
                                {
                                        while(pri(ch)<pri(s.Top()) && !s.empty())
                                        {
                                                x = s.pop();
                                                pe[k++] = x;
                                        }
                                        s.push(ch);
                                }
                        }
                }
                pe[k]='\0';
        }

        while(!s.empty())
        {
                x = s.pop();
                pe[k++] = x;
```

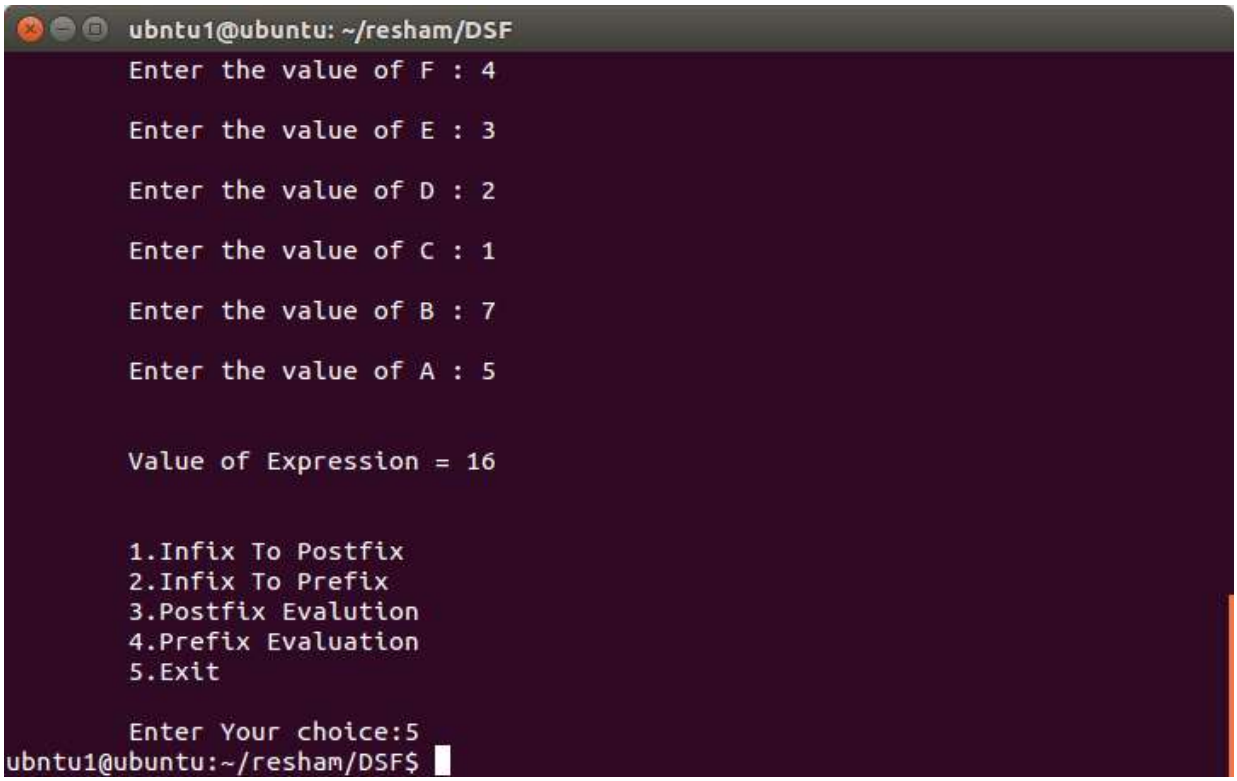
```
        }
        pe[k] = '\0';
}

int convert :: pri(char x)
{
        switch(x)
        {
                case '$' :
                case '#' :
                case '^' : return 3;

                case '%' :
                case '*'  :
                case '/'  : return 2;

                case '+' :
                case '-'  : return 1;

                default : return 0;
        }
}

int main()
{
        convert c;
        c.exp();
}
```

***Output :***


ubntu1@ubuntu: ~/resham/DSF
ubntu1@ubuntu:~/resham/DSF$ g++ ass1.cpp
ubntu1@ubuntu:~/resham/DSF$ ./a.out
1.Infix To Postfix
2.Infix To Prefix
3.Postfix Evalution
4.Prefix Evaluation
5.Exit
Enter Your choice:1
Enter expression to convert postfix : (A+B)*(C^(D-E)+F)/G^(H-J)
Infix:->> (A+B)*(C^(D-E)+F)/G^(H-J)
Postfix->> AB+CDE-^F+*GHJ-^/
1.Infix To Postfix
2.Infix To Prefix
3.Postfix Evalution
4.Prefix Evaluation
5.Exit


ubntu1@ubuntu: ~/resham/DSF
Enter Your choice:2
Enter expression to convert prefix: ((A+B)*D)^(E-F)
Infix:->> ((A+B)*D)^(E-F)
Prefix:->> ^*+ABD-EF
1.Infix To Postfix
2.Infix To Prefix
3.Postfix Evalution
4.Prefix Evaluation
5.Exit
Enter Your choice:3
Enter Postfix expression to Evalute : AB+C/D*EF^-
Enter the value of A : 5
Enter the value of B : 7
Enter the value of C : 6


ubntu1@ubuntu: ~/resham/DSF
Enter the value of D : 10
Enter the value of E : 2
Enter the value of F : 3
Value of Expression = 12
1.Infix To Postfix
2.Infix To Prefix
3.Postfix Evalution
4.Prefix Evaluation
5.Exit
Enter Your choice:4
Enter Postfix expression to Evalute : +-+AB/*CDEF
Enter the value of F : 4
Enter the value of E : 3


ubntu1@ubuntu: ~/resham/DSF
Enter the value of F : 4
Enter the value of E : 3
Enter the value of D : 2
Enter the value of C : 1
Enter the value of B : 7
Enter the value of A : 5
Value of Expression = 16
1.Infix To Postfix
2.Infix To Prefix
3.Postfix Evalution
4.Prefix Evaluation
5.Exit
Enter Your choice:5
ubntu1@ubuntu:~/resham/DSF$