Name:- Brendan Lucas , Roll No:- 8953

Expression Tree implication Code:-

Source Code:-

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <math.h>
#define total 50
typedef struct node
 {
        char data;
        struct node *left,*right;
 }node;

 struct prog
{
        node *arr[total];
        int top;
}stack;

struct eval
{
        int arr[total];
        int top;
}numstack;

typedef struct
 {
node *start;
 }bt;

int isfull()
{
        return !(stack.top<total);
}

int isempty()
{
        return stack.top==-1;
}
void push(node *p)
{
        if(isfull())
        {
                //printf("--------------------------------\n");
                //printf("Stack overflow\n");
                //printf("--------------------------------\n");

        }
```

```
	else
	{
		stack.arr[++stack.top]=p;
		//printf("--------------------------------\n");
		//printf("pushed successfully\n");
		//printf("--------------------------------\n");
	}
}

node* pop()
{
	if(isempty())
	{
		//printf("--------------------------------\n");
		//printf("Stack Underflow\n");
		//printf("--------------------------------\n");
		return NULL;
	}
	else
	{
		//printf("--------------------------------\n");
		//printf("Successfully popped\n");
		//printf("--------------------------------\n");
		return stack.arr[stack.top--];
	}
}
void preorder(node* n)
 {
	node *p=n;
	if(p==NULL)
	{
		printf("Tree is Empty\n");
		return;
	}
	printf("%c ",p->data);
	if(p->left)
	{
		preorder(p->left);
	}
	if(p->right)
	{
		preorder(p->right);
	}
 }
 void inorder(node* n)
 {
	node *p=n;
	if(p==NULL)
	{
		printf("Tree is Empty\n");
		return;
	}
	if(p->left)
```

```
        {
                printf("(");
                inorder(p->left);
        }
        printf("%c ",p->data);
        if(p->right)
        {
                inorder(p->right);
                printf(")");
        }
}
 void postorder(node* n)
{
        node *p=n;
        if(p==NULL)
        {
                printf("Tree is Empty\n");
                return;
        }
        if(p->left)
        {
                postorder(p->left);
        }
        if(p->right)
        {
                postorder(p->right);
        }
        printf("%c ",p->data);
}

int evaluate(char *a)
{
        int i=0,c,b,s;
        numstack.top=-1;
        while(a[i]!='\0')
        {
                if(isdigit(a[i]))
                {
                        numstack.arr[++numstack.top]=a[i]-48;
                }
                else
                {
                        c=numstack.arr[numstack.top--];
                        b=numstack.arr[numstack.top--];
                        switch(a[i])
                        {
                                case '+':{s=b+c;break;}
                                case '-':{s=b-c;break;}
                                case '*':{s=b*c;break;}
                                case '/':{s=b/c;break;}
                                case '%':{s=b%c;break;}
                                case '^':{s=(int)powf(b, c);break;}
                        }
```

```
                        numstack.arr[++numstack.top]=s;
                }
                i++;
        }
        return numstack.arr[0];
 }
int main(void)
{
        node *q;
        char a[total];
        int i=0;
        bt tree;
        stack.top=-1;
        numstack.top=-1;
        printf("Expression Tree program\n\n");
        printf("Enter the postfix Expression:-\n");
        gets(a);
        while(a[i]!='\0')
        {
                q=(node*)malloc(sizeof(node));
                q->left=q->right=NULL;
                q->data=a[i];
                if(!isalnum(a[i]))
                {
                        q->right=pop();
                        q->left=pop();
                }

                push(q);
                i++;
        }
        tree.start=pop();
        printf("\n");
        printf("Infix order :-  ");
        inorder(tree.start);
        printf("\n");
        printf("Prefix order :-  ");
        preorder(tree.start);
        printf("\n");
        printf("Postfix order :-  ");
        postorder(tree.start);
        printf("\n");
        i=evaluate(a);
        printf("The value of the expression is %d",i);
}
```

Output:-

Expression Tree program

Enter the postfix Expression:-
7654321^%/*-+

Infix order :- (7 + (6 - (5 * (4 / (3 % (2 ^ 1))))))
Prefix order :- + 7 - 6 * 5 / 4 % 3 ^ 2 1
Postfix order :- 7 6 5 4 3 2 1 ^ % / * - +
The value of the expression is -7