SSN COLLEGE OF ENGINEERING

(Autonomous)

Affiliated to Anna University

DEPARTMENT OF CSE

UCS308 Data Structures Lab

Assignment 7

Expression Tree

Register Number : 185001131

Name : Sai Charan B

Class : CSE – B

Tree.h

```
#include<stdio.h>

#include<stdlib.h>
```

```
#include<string.h>
#include "stack.h"

typedef struct et
{
      char value;
      struct et *left, *right;
}et;

typedef struct str
{
      et *x;
      struct str *next;
}Node;
Node *Top=NULL;

void treepush(et* value)
{
      Node *new;
      new=malloc(sizeof(Node));
      new->x=value;
      if(Top==NULL)
            new->next=NULL;
      else
```

```
            new->next=Top;
      Top=new;
}

et* treePop()
{
      et *ele=NULL;
      if(Top==NULL)
            printf("Stack Empty\n");
      else
      {
            Node *temp=Top;
            Top=temp->next;
            ele=temp->x;
            free(temp);
      }
      return ele;
}

void preorder(et* t)
{
      if(t!=NULL)
      {
            printf("%c",t->value);
```

```
            preorder(t->left);
            preorder(t->right);
      }
}

void inorder(et* t)
{
      if(t!=NULL)
      {
            inorder(t->left);
            printf("%c",t->value);
            inorder(t->right);
      }
}

void postorder(et* t)
{
      if(t!=NULL)
      {
            postorder(t->left);
            postorder(t->right);
            printf("%c",t->value);
      }
}
```

```
int isOperator(char c)
{
	if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^')
		return 1;
	return 0;
}

et* newNode(int v)
{
	et* temp = (et*)malloc(sizeof(et));
	temp->left = temp->right = NULL;
	temp->value = v;
	return temp;
}

et* constructTree(char postfix[])
{
	et *t, *t1, *t2;
	for (int i=0; i<strlen(postfix); i++)
	{
		if (!isOperator(postfix[i]))
		{
			t = newNode(postfix[i]);
```

```
                treepush(t);
            }
            else // operator
            {
                t = newNode(postfix[i]);
                t1 = treePop();
                t2 = treePop();
                t->right = t1;
                t->left = t2;
                treepush(t);
            }
        }
        t = treePop();
        return t;
}
```

Stack.h

```
#include<stdio.h>
#include<stdlib.h>
```

```
#include<string.h>
#include<ctype.h>
#define bool int


struct sNode
{
char data;
struct sNode *next;
};


void pushi(struct sNode** top_ref, float new_data);


int popi(struct sNode** top_ref);


bool isMatchingPair(char character1, char character2)
{
if (character1 == '(' && character2 == ')')
     return 1;
else if (character1 == '{' && character2 == '}')
     return 1;
```

```
else if (character1 == '[' && character2 == ']')
      return 1;
else
    return 0;
}


bool areParenthesisBalanced(char exp[])
{
int i = 0;


struct sNode *stack = NULL;


while (exp[i])
{

      if (exp[i] == '{' || exp[i] == '(' || exp[i] == '[')
      pushi(&stack, exp[i]);


      if (exp[i] == '}' || exp[i] == ')' || exp[i] == ']')
      {
```

```
      if (stack == NULL)
      return 0;



      else if ( !isMatchingPair(popi(&stack), exp[i]) )
      return 0;
      }
      i++;
}



if (stack == NULL)
      return 1;
else
      return 0;
}



void pushi(struct sNode** top_ref, float new_data)
{
      struct sNode* new_node = (struct sNode*) malloc(sizeof(struct
sNode));
```

```
      new_node->data = new_data;
      new_node->next = (*top_ref);
      (*top_ref) = new_node;
}


int popi(struct sNode** top_ref)
{
      char res;
      struct sNode *top;
      top = *top_ref;
      res = top->data;
      *top_ref = top->next;
      free(top);
      return res;
}

struct Stack
{
int top;
unsigned capacity;
float* array;
};
```

```
struct Stack* createStack( unsigned capacity )

{

     struct Stack* stack = (struct Stack*) malloc(sizeof(struct
Stack));



     if (!stack)

     return NULL;



     stack->top = -1;

     stack->capacity = capacity;



     stack->array = (float*) malloc(stack->capacity *
sizeof(float));



     if (!stack->array)

     return NULL;

     return stack;

}



int isEmpty(struct Stack* stack)

{

     return stack->top == -1 ;

}
```

```
char peek(struct Stack* stack)
{
      return stack->array[stack->top];
}

float pop(struct Stack* stack)
{
      if (!isEmpty(stack))
      return stack->array[stack->top--] ;
      return '$';
}

void push(struct Stack* stack, char op)
{
      stack->array[++stack->top] = op;
}

int isOperand(char ch)
{
      return (ch >= '1' && ch <= '9');
}

int Prec(char ch)
{
```

```
        switch (ch)
        {
        case '+':
        case '-':
        return 1;

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

        case '^':
        return 3;
        }
        return -1;
}

int infixToPostfix(char* exp)
{
        int i, k;

        struct Stack* stack = createStack(strlen(exp));
        if(!stack)
        return -1;
        for (i = 0, k = -1; exp[i]; ++i)
```

```
	{


	if (isOperand(exp[i]))
		exp[++k] = exp[i];



	else if (exp[i] == '(')
		push(stack, exp[i]);



	else if (exp[i] == ')')
	{
		while (!isEmpty(stack) && peek(stack) != '(')
			exp[++k] = pop(stack);
		if (!isEmpty(stack) && peek(stack) != '(')
			return -1;
		else
			pop(stack);
	}
	else
	{
		while (!isEmpty(stack) && Prec(exp[i]) <=
Prec(peek(stack)))
			exp[++k] = pop(stack);
```

```
            push(stack, exp[i]);
      }

      }
      while (!isEmpty(stack))
      exp[++k] = pop(stack );

      exp[++k] = '\0';
      return 0;
}
```

Main.c

```
#include "tree.h”

int main()
{
      int ch;
      do
      {
            char exp[100];
            printf("\nEnter Expression : ");
```

```
            scanf("%s",exp);
            infixToPostfix(exp);
            et* r = constructTree(exp);
            printf("Preorder : ");
            preorder(r);
            printf("\nInorder : ");
            inorder(r);
            printf("\nPostorder : ");
            postorder(r);
            printf("\n\n");
            printf("Do you want to continue? (1.Yes, 2.No)");
            scanf("%d",&ch);
      }while(ch!=2);
      return 0;
}
```

Output:

```
Enter Expression : (2+5)*(3-6)/(7/8)
Preorder : /*+25-36/78
Inorder : 2+5*3-6/7/8
```

```
Postorder : 25+36-*78//

Do you want to continue? (1.Yes, 2.No)1

Enter Expression : 7-(((3+2)*(6+1))/(5+6))
Preorder : -7/*+32+61+56
Inorder : 7-3+2*6+1/5+6
Postorder : 732+61+*56+/-

Do you want to continue? (1.Yes, 2.No)1

Enter Expression : ((3+2)*(2+5))
Preorder : *+32+25
Inorder : 3+2*2+5
Postorder : 32+25+*

Do you want to continue? (1.Yes, 2.No)2
```