#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#include <ctype.h>

#include "Stack.h"

#include "ExpressionTree.h"

int main(){

char infix[30];

char \*postfix;

printf("Enter the expression: ");

scanf("%[^\n]%\*c", infix);

do{

postfix = (char \*)malloc(strlen(infix)\*sizeof(char));

if(balanced\_parenthesis(infix)){

printf("Expression is Balanced\n\n");

infixTopostfix(infix,postfix);

printf("Postfix Expression: ");

for(int i=0;i<strlen(postfix);i++)

printf("%c",postfix[i]);

printf("\n\n");

Tree \*ExpT = constructTree(postfix);

printf("Expression Tree Constructed\n\n");

printf("---------------------------------------------------------------------------------\n");

printf("\t\t\t\tTREE TRAVERSAL\n");

printf("Preorder: ");

preorder(ExpT);

printf("\n\nInorder: ");

inorder(ExpT);

printf("\n\nPostorder: ");

postorder(ExpT);

printf("\n\n");

printf("---------------------------------------------------------------------------------\n");

}

else{

printf("Expression is unbalanced!!\n");

}

printf("Enter the expression(to stop enter -1): ");

scanf("%[^\n]%\*c", infix);

}while(strcmp(infix,"-1"));

}

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

**Stack.h**

struct Stack{

int data;

struct Stack \*next;

};

typedef struct Stack Node;

Node \*top = NULL;

void push(char value){

Node \*newNode;

newNode = (Node\*)malloc(sizeof(Node));

newNode->data = value;

if(top == NULL)

newNode->next = NULL;

else

newNode->next = top;

top = newNode;

}

void pop(){

if(top == NULL){

printf("\nStack is Empty!!!\n");

}

else{

Node \*temp = top;

//printf("\nDeleted element: %d", temp->data);

top = temp->next;

free(temp);

}

}

char peek(){

if(top == NULL)

printf("\nStack is Empty!!!\n");

else

return top->data;

}

void display(){

if(top == NULL){

printf("\nStack is Empty!!!\n");

}

else{

Node \*temp = top;

while(temp->next != NULL){

printf("%c--->",temp->data);

temp = temp -> next;

}

printf("%c--->NULL",temp->data);

}

}

void infixTopostfix(char infix[],char \*postfix){

int len = strlen(infix);

int ind=0;

for(int i = 0 ; i < len ; i++){

if(isdigit(infix[i]) || isalpha(infix[i])){

postfix[ind]=infix[i];

ind++;

}

else if(infix[i]=='('){

push(infix[i]);

}

else if(infix[i]=='/' || infix[i]=='\*' || infix[i]=='+' || infix[i]=='-'){

if((top != NULL) && ((top->data=='/' && infix[i]=='/') || (top->data=='\*' && infix[i]=='\*') || (top->data=='/' && infix[i]=='\*') || (top->data=='\*' && infix[i]=='/') || (top->data=='+' && infix[i]=='+') || (top->data=='-' && infix[i]=='-') || (top->data=='-' && infix[i]=='+') || (top->data=='+' && infix[i]=='-') || (top->data=='/' && infix[i]=='+') || (top->data=='/' && infix[i]=='-') || (top->data=='\*' && infix[i]=='+') || (top->data=='\*' && infix[i]=='-'))){

do{

postfix[ind] = top->data;

ind++;

pop();

}while((top != NULL) && (top->data!='(' || isdigit(infix[i]) || isalpha(infix[i])));

push(infix[i]);

}

else if((top != NULL) && ((top->data=='+' && infix[i]=='/') || (top->data=='+' && infix[i]=='\*') || (top->data=='-' && infix[i]=='/') || (top->data=='-' && infix[i]=='\*'))){

push(infix[i]);

}

else{

push(infix[i]);

}

}

else if(infix[i]==')'){

do{

postfix[ind] = top->data;

ind++;

pop();

}while(top != NULL && top->data != '(');

pop();

}

/\*display();

printf("\n");\*/

}

while(top != NULL){

postfix[ind] = top->data;

ind++;

pop();

}

}

bool balanced\_parenthesis(char exp[]){

for(int i=0;i<strlen(exp);i++){

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

push(exp[i]);

}

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

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

pop();

else

return false;

}

}

if(top == NULL)

return true;

else

return false;

}

----------------------------------------------------------------------**ExpressionTree.h**

struct ExpTree{

int data;

struct ExpTree \*left, \*right;

};

typedef struct ExpTree Tree;

struct TreeStack{

Tree \*data;

struct TreeStack \*next;

};

typedef struct TreeStack stack;

stack \*st;

void pushNode(Tree \*value){

stack \*newNode;

newNode = (stack\*)malloc(sizeof(stack));

newNode->data = (Tree \*)malloc(sizeof(Tree));

newNode->data = value;

if(st == NULL)

newNode->next = NULL;

else

newNode->next = st;

st = newNode;

}

void popNode(){

if(st == NULL){

printf("\nStack is Empty!!!\n");

}

else{

stack \*temp = st;

st = temp->next;

free(temp);

}

}

Tree \*topNode(){

if(st == NULL)

printf("\nStack is Empty!!!\n");

else

return st->data;

}

int isOperator(char c){

if (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^')

return 1;

return 0;

}

Tree \*newNode(int v){

Tree \*temp = (Tree\*)malloc(sizeof(Tree));

temp->left = temp->right = NULL;

temp->data = v;

return temp;

}

Tree\* constructTree(char postfix[]){

Tree \*t, \*t1, \*t2;

for (int i=0; i<strlen(postfix); i++){

if (!isOperator(postfix[i])){

t = newNode(postfix[i]);

//printf("%c\n",t->data);

pushNode(t);

//printf("st=%c\n",st->data->data);

}

else{ // operator

t = newNode(postfix[i]);

t1 = topNode(); // Store top

popNode(); // Remove top

t2 = topNode();

popNode();

t->right = t1; // make them children

t->left = t2;

pushNode(t); // Add this subexpression to stack

}

}

t = topNode();

popNode();

return t;

}

void preorder(Tree\* t){

if(t!=NULL){

printf("%c",t->data);

preorder(t->left);

preorder(t->right);

}

}

void inorder(Tree\* t){

if(t!=NULL){

inorder(t->left);

printf("%c",t->data);

inorder(t->right);

}

}

void postorder(Tree\* t){

if(t!=NULL){

postorder(t->left);

postorder(t->right);

printf("%c",t->data);

}

}

----------------------------------------------------------------------**OUTPUT**

Enter the expression: (2+5)\*(3-6)/(7\*8)

Expression is Balanced

Postfix Expression: 25+36-\*78\*/

Expression Tree Constructed

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

TREE TRAVERSAL

Preorder: /\*+25-36\*78

Inorder: 2+5\*3-6/7\*8

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

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

Enter the expression(to stop enter -1): 7-(((3+2)\*(6+1))/(5+6))

Expression is Balanced

Postfix Expression: 732+61+\*56+/-

Expression Tree Constructed

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

TREE TRAVERSAL

Preorder: -7/\*+32+61+56

Inorder: 7-3+2\*6+1/5+6

Postorder: 732+61+\*56+/-

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

Enter the expression(to stop enter -1): ((3+2)\*(2+5))

Expression is Balanced

Postfix Expression: 32+25+\*

Expression Tree Constructed

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

TREE TRAVERSAL

Preorder: \*+32+25

Inorder: 3+2\*2+5

Postorder: 32+25+\*

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

Enter the expression(to stop enter -1): -1