**DS LAB 13: Expression Tree**

Name: Vivian Vijay Ludrick

Branch: SE Comps A Batch C

Roll No: 9914

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

*// Define a structure for a binary tree node*

typedef struct tnode

{

char data; *// Data (operator or operand) in the node*

struct tnode \*left; *// Pointer to the left child node*

struct tnode \*right; *// Pointer to the right child node*

} Tnode;

*// Define a structure for a stack of binary tree nodes*

typedef struct

{

Tnode \*a[20]; *// Array to store nodes*

int tos; *// Top of stack pointer*

} Stack;

*// Define a structure for a tree, which will hold the root of the expression tree*

typedef struct

{

Tnode \*root; *// Pointer to the root node of the expression tree*

} Tree;

*// Function to push a node onto the stack*

void **push**(Stack \**s*, Tnode \**p*)

{

*s*->a[++*s*->tos] = *p*;

}

*// Function to pop a node from the stack*

Tnode \***pop**(Stack \**s*)

{

return *s*->a[*s*->tos--];

}

*// Function to create an expression tree from a postfix expression*

Tnode \***createTree**(char *exp*[])

{

Stack s;

s.tos = -1; *// Initialize the stack top*

int i;

Tnode \*x, \*y;

for (i = 0; *exp*[i] != '\0'; i++)

{

Tnode \*p;

p = (Tnode \*)**malloc**(sizeof(Tnode));

p->data = *exp*[i];

p->left = p->right = **NULL**;

if (!**isalnum**(*exp*[i]))

{

*// If the character is an operator, pop the top two nodes from the stack*

x = **pop**(&s);

y = **pop**(&s);

*// Make the popped nodes the left and right children of the new operator node*

p->left = y;

p->right = x;

}

*// Push the new node onto the stack*

**push**(&s, p);

}

*// The root of the expression tree is the last node left on the stack*

return **pop**(&s);

}

*// Function to perform an in-order traversal and print the fully parenthesized infix expression*

void **inorderfullp**(Tnode \**rt*)

{

if (*rt* != **NULL**)

{

if (*rt*->left != **NULL** && *rt*->right != **NULL**)

{

*// If the node has both left and right children, print an opening parenthesis*

**printf**("(");

}

*// Recursively traverse the left subtree*

**inorderfullp**(*rt*->left);

*// Print the operator or operand character*

**printf**("%c ", *rt*->data);

*// Recursively traverse the right subtree*

**inorderfullp**(*rt*->right);

if (*rt*->left != **NULL** && *rt*->right != **NULL**)

{

*// If the node had both children, print a closing parenthesis*

**printf**(")");

}

}

}

*// Function to evaluate the expression tree*

int **expeval**(Tnode \**rt*)

{

int t1, t2;

if (*rt*->left == **NULL** && *rt*->right == **NULL**)

{

*// If the node is a leaf (operand), convert the character to an integer*

return (*rt*->data - '0');

}

else

{

*// If the node is an operator*

t1 = **expeval**(*rt*->left); *// Evaluate the left subtree*

t2 = **expeval**(*rt*->right); *// Evaluate the right subtree*

switch (*rt*->data)

{

case '+':

return (t1 + t2);

case '-':

return (t1 - t2);

case '\*':

return (t1 \* t2);

case '/':

if (t2 == 0)

{

**printf**("Division by zero\n");

**exit**(1);

}

return (t1 / t2);

}

}

}

int **main**()

{

char str[20];

int ans;

Tree t1;

t1.root = **NULL**;

**printf**("Enter a postfix expression\n");

**scanf**("%s", str);

t1.root = **createTree**(str); *// Create the expression tree and return the root pointer*

**printf**("The Infix Expression is: ");

**inorderfullp**(t1.root); *// Print the fully parenthesized infix expression*

**printf**("\n");

ans = **expeval**(t1.root); *// Evaluate the expression*

**printf**("The Evaluation answer is: %d\n", ans); *// Print the result*

return 0;

}

OUTPUT:

