**EXPT 9**

**Develop and execute a program in C using suitable data structures to create a binary tree for an expression. The tree traversals in some proper method should result in conversion of original expression into prefix, infix and postfix forms. Display the original expression along with the three different forms also.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Define the structure for a binary tree node

typedef struct Node {

char data;

struct Node\* left;

struct Node\* right;

} Node;

// Function to create a new binary tree node

Node\* createNode(char data) {

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

if (!newNode) {

printf("Memory error\n");

return NULL;

}

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;

}

// Function to insert a node into the binary tree

Node\* insertNode(Node\* root, char data) {

if (root == NULL) {

root = createNode(data);

} else if (data <= root->data) {

root->left = insertNode(root->left, data);

} else {

root->right = insertNode(root->right, data);

}

return root;

}

// Function to perform inorder traversal (infix form)

void inorderTraversal(Node\* root) {

if (root) {

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

}

// Function to perform preorder traversal (prefix form)

void preorderTraversal(Node\* root) {

if (root) {

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

preorderTraversal(root->left);

preorderTraversal(root->right);

}

}

// Function to perform postorder traversal (postfix form)

void postorderTraversal(Node\* root) {

if (root) {

postorderTraversal(root->left);

postorderTraversal(root->right);

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

}

}

int main() {

char expression[100];

// Get the expression from the user

printf("Enter an expression: ");

fgets(expression, sizeof(expression), stdin);

expression[strcspn(expression, "\n")] = 0; // Remove the newline character

Node\* root = NULL;

// Create the binary tree

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

root = insertNode(root, expression[i]);

}

// Display the original expression

printf("Original Expression: %s\n", expression);

// Perform tree traversals to get prefix, infix, and postfix forms

printf("Prefix Form: ");

preorderTraversal(root);

printf("\n");

printf("Infix Form: ");

inorderTraversal(root);

printf("\n");

printf("Postfix Form: ");

postorderTraversal(root);

printf("\n");

return 0;

}