```
// Including necessary header files
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

// Structure for a node in the expression tree
struct TreeNode {
    char data;
    struct TreeNode* left;
    struct TreeNode* right;
};

// Function to create a new node
struct TreeNode* createNode(char value) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    if (newNode != NULL) {
        newNode->data = value;
        newNode->left = NULL;
        newNode->right = NULL;
    }
    return newNode;
}

// Function to check if a character is an operator
int isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/');
}

// Function to build an expression tree from a postfix expression
struct TreeNode* buildExpressionTree(char postfix[]) {
    struct TreeNode* stack[100];
```

```
    int top = -1;

    for (int i = 0; postfix[i] != '\0'; i++) {
        struct TreeNode* newNode = createNode(postfix[i]);

        if (isdigit(postfix[i])) {
            stack[++top] = newNode;
        } else if (isOperator(postfix[i])) {
            newNode->right = stack[top--];
            newNode->left = stack[top--];
            stack[++top] = newNode;
        }
    }

    return stack[top];
}

// Function to evaluate the expression tree
int evaluateExpressionTree(struct TreeNode* root) {
    if (root->data == '+') {
        return evaluateExpressionTree(root->left) + evaluateExpressionTree(root->right);
    } else if (root->data == '-') {
        return evaluateExpressionTree(root->left) - evaluateExpressionTree(root->right);
    } else if (root->data == '*') {
        return evaluateExpressionTree(root->left) * evaluateExpressionTree(root->right);
    } else if (root->data == '/') {
        return evaluateExpressionTree(root->left) / evaluateExpressionTree(root->right);
    } else {
        return root->data - '0'; // Convert character to integer
    }
}
```

```
// Function to perform in-order traversal of the expression tree
void inOrderTraversal(struct TreeNode* root) {
    if (root != NULL) {
        inOrderTraversal(root->left);
        printf("%c ", root->data);
        inOrderTraversal(root->right);
    }
}

// Function to free the memory allocated for the expression tree
void freeExpressionTree(struct TreeNode* root) {
    if (root != NULL) {
        freeExpressionTree(root->left);
        freeExpressionTree(root->right);
        free(root);
    }
}

int main() {
    char postfixExpression[100];

    // Input postfix expression
    printf("Enter a postfix expression: ");
    scanf("%s", postfixExpression);

    // Build the expression tree
    struct TreeNode* root = buildExpressionTree(postfixExpression);

    // Display the in-order traversal of the expression tree
    printf("In-order Traversal of the Expression Tree: ");
```

```
    inOrderTraversal(root);
    printf("\n");

    // Evaluate and display the result
    int result = evaluateExpressionTree(root);
    printf("Result: %d\n", result);

    // Free allocated memory
    freeExpressionTree(root);

    return 0;
}
```

```
Enter a postfix expression: 23+5*
In-order Traversal of the Expression Tree: 2 + 3 * 5
Result: 25
Enter a postfix expression: 92/3*4+
In-order Traversal of the Expression Tree: 9 / 2 * 3 + 4
Result: 16
Enter a postfix expression: 82/3*
In-order Traversal of the Expression Tree: 8 / 2 * 3
Result: 12
Enter a postfix expression: 63/4*5+
In-order Traversal of the Expression Tree: 6 / 3 * 4 + 5
Result: 13
```