EXPERIMENT-8

Program :

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

typedef struct TreeNode {
    char data;
    struct TreeNode *left;
    struct TreeNode *right;
} TreeNode;

// Operator precedence
int getPrecedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    if (op == '^') return 3;
    return -1;
}

// Convert infix to postfix
void infixToPostfix(char* infix, char* postfix) {
    char opStack[100];
    int top = -1, k = 0;
    for (int i = 0; infix[i]; i++) {
        if (isalnum(infix[i]))
            postfix[k++] = infix[i];
        else if (infix[i] == '(')
            opStack[++top] = infix[i];
        else if (infix[i] == ')') {
            while (top != -1 && opStack[top] != '(')
                postfix[k++] = opStack[top--];
            top--; // remove '('
```

```
        } else {
            while (top ≠ -1 && getPrecedence(infix[i]) ⩽
getPrecedence(opStack[top]))
                postfix[k++] = opStack[top--];
            opStack[++top] = infix[i];
        }
    }
    while (top ≠ -1)
        postfix[k++] = opStack[top--];
    postfix[k] = '\0';
}

// Create tree node
TreeNode* createNode(char data) {
    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
    newNode→data = data;
    newNode→left = newNode→right = NULL;
    return newNode;
}

// Build expression tree from postfix
TreeNode* buildExpressionTree(char* postfix) {
    TreeNode* nodeStack[100];
    int top = -1;
    TreeNode *node, *rightChild, *leftChild;
    for (int i = 0; postfix[i]; i++) {
        if (isalnum(postfix[i])) {
            node = createNode(postfix[i]);
            nodeStack[++top] = node;
        } else {
            node = createNode(postfix[i]);
            rightChild = nodeStack[top--];
            leftChild = nodeStack[top--];
            node→left = leftChild;
            node→right = rightChild;
            nodeStack[++top] = node;
        }
    }
```

```
    return nodeStack[top];
}

// Print tree using ASCII
void printTree(TreeNode* root, char* prefix, int isLeft) {
    if (root == NULL) return;
    printf("%s", prefix);
    printf("%s", isLeft ? "+--" : "`--");
    printf("%c\n", root->data);

    char newPrefix[100];
    strcpy(newPrefix, prefix);
    strcat(newPrefix, isLeft ? "|   " : "    ");

    if (root->left || root->right) {
        printTree(root->left, newPrefix, 1);
        printTree(root->right, newPrefix, 0);
    }
}

// Print prefix expression
void printPrefix(TreeNode* root) {
    if (root == NULL) return;
    printf("%c", root->data);
    printPrefix(root->left);
    printPrefix(root->right);
}

int main() {
    char infix[100], postfix[100];
    printf("Enter an infix expression: ");
    fgets(infix, sizeof(infix), stdin);
    infix[strcspn(infix, "\n")] = 0; // remove newline

    infixToPostfix(infix, postfix);
    TreeNode* root = buildExpressionTree(postfix);
```

```
    printf("\nInfix Expression: %s", infix);
    printf("\nPostfix Expression: %s", postfix);
    printf("\nPrefix Expression: ");
    printPrefix(root);

    printf("\n\nExpression Tree (with branches):\n");
    printTree(root, "", 0);

    return 0;
}
```

Output :

```
PS C:\Users\there\downloads\Alwin> gcc BinaryTreepostfix.c
PS C:\Users\there\downloads\Alwin> ./a.exe
Enter an infix expression:  (A+B)*C

Infix Expression:  (A+B)*C
Postfix Expression: AB+C*
Prefix Expression:  *+ABC

Expression Tree (with branches):
`--
   `--*
      +--+
      |  +--A
      |  `--B
      `--C
PS C:\Users\there\downloads\Alwin>
```