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

typedef struct tnode
{
    char data;
    struct tnode *left, *right;
} Tnode;

typedef struct
{
    Tnode *a[20];
    int tos;
} Stack;

typedef struct
{
    Tnode *root;
} Tree;

void push(Stack *s, Tnode *p)
{
    // Push a node onto the stack
    s->a[++s->tos] = p;
}

Tnode *pop(Stack *s)
{
    // Pop a node from the stack
    return s->a[s->tos--];
}

Tnode *createTree(char exp[])
{
    Stack s;
    s.tos = -1;

    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);
}

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(")");
        }
    }
}

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
```

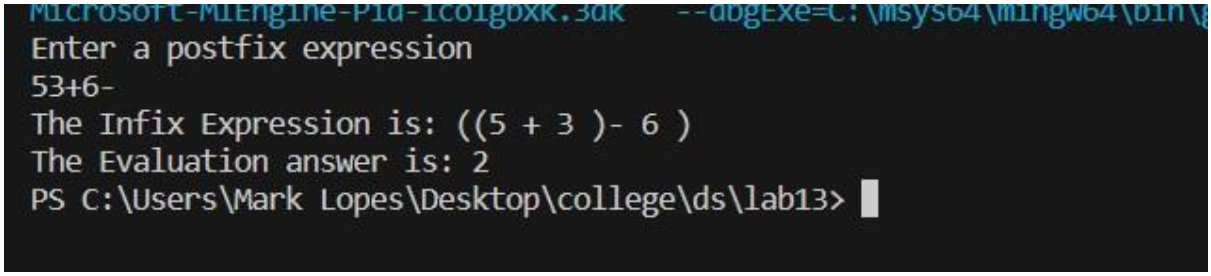
```
        return (rt->data - '0');
    }
    else
    {
        // If the node is an operator
        t1 = expeval(rt->left);  // Evaluate the left subtree
        t2 = expeval(rt->right); // Evaluate the right subtree
        switch (rt->data)
        {
        case '+':
            return (t1 + t2);
        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);
    return 0;
}
```


Microsoft-MIEngine-Pid-1co1gbxk.3dk --dbgExe=C:\msys64\mingw64\bin\g
Enter a postfix expression
53+6-
The Infix Expression is: ((5 + 3 )- 6 )
The Evaluation answer is: 2
PS C:\Users\Mark Lopes\Desktop\college\ds\lab13>