LAB 7 - Binary Trees and Binary Search Trees

Student Details

Name	Pratyush Deo Singh
Registration Number	25BCE5101
Semester	2nd
Subject	Data Structures and Algorithms
Slot	A2
Faculty Name	Dr. Malini A

Programs

Program 1: Binary Tree Traversals (Preorder, Inorder, Postorder)

File Name:

q1_binary_tree_traversals.c

Program Code:

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

struct Node* insert(struct Node* root, int value) {
    if (root == NULL)
        return createNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);
    return root;
}

void preorder(struct Node* root) {
    if (root == NULL)
        return;
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}

void inorder(struct Node* root) {
    if (root == NULL)
        return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

void postorder(struct Node* root) {
    if (root == NULL)
        return;
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}

int main() {
    struct Node* root = NULL;
    int n, value;
    printf("Enter number of nodes: ");
    scanf("%d", &n);

    printf("Enter %d values: ", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        root = insert(root, value);
    }

    printf("\nPreorder Traversal:  ");
    preorder(root);

    printf("\nInorder Traversal:   ");
    inorder(root);

    printf("\nPostorder Traversal: ");
    postorder(root);

    printf("\n");
    return 0;
}
```

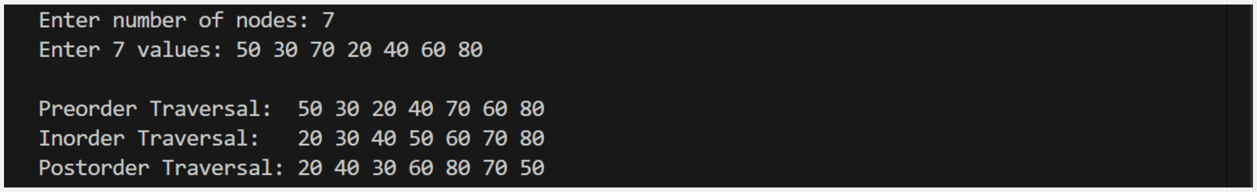
Sample Test Input:

```
Enter number of nodes: 7
Enter 7 values: 50 30 70 20 40 60 80
```

Expected Output:

```
Preorder Traversal:  50 30 20 40 70 60 80
Inorder Traversal:   20 30 40 50 60 70 80
Postorder Traversal: 20 40 30 60 80 70 50
```

Output Screenshot:


```
Enter number of nodes: 7
Enter 7 values: 50 30 70 20 40 60 80

Preorder Traversal:  50 30 20 40 70 60 80
Inorder Traversal:   20 30 40 50 60 70 80
Postorder Traversal: 20 40 30 60 80 70 50
```

Program 2: Expression Tree from Postfix — Print Prefix and Infix

File Name:

q2_expression_tree.c

Program Code:

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

struct Node {
    char data;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(char ch) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = ch;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
```

```
struct Node* stack[100];
int top = -1;

void push(struct Node* node) {
    stack[++top] = node;
}

struct Node* pop() {
    return stack[top--];
}

int isOperator(char ch) {
    return (ch == '+' || ch == '-' || ch == '*' || ch == '/');
}

struct Node* buildExpressionTree(char* postfix) {
    struct Node* node;
    struct Node* left;
    struct Node* right;

    for (int i = 0; postfix[i] != '\0'; i++) {
        char ch = postfix[i];
        if (ch == ' ')
            continue;

        node = createNode(ch);

        if (isOperator(ch)) {
            right = pop();
            left = pop();
            node->right = right;
            node->left = left;
        }

        push(node);
    }

    return pop();
}

void prefixTraversal(struct Node* root) {
    if (root == NULL)
        return;
    printf("%c", root->data);
    prefixTraversal(root->left);
    prefixTraversal(root->right);
}

void infixTraversal(struct Node* root) {
    if (root == NULL)
        return;
    if (isOperator(root->data))
        printf("(");
    infixTraversal(root->left);
    printf("%c", root->data);
    infixTraversal(root->right);
    if (isOperator(root->data))
        printf(")");
}

int main() {
    char postfix[100];
    printf("Enter postfix expression (e.g., ab+cd-*): ");
    scanf("%s", postfix);

    struct Node* root = buildExpressionTree(postfix);

    printf("\nPrefix Expression:  ");
    prefixTraversal(root);

    printf("\nInfix Expression:   ");
    infixTraversal(root);

    printf("\n");
    return 0;
}
```

Sample Test Input:

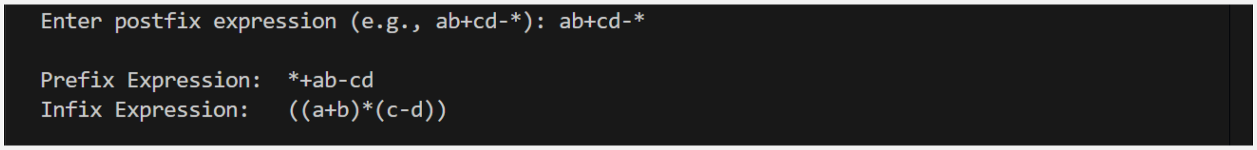
Enter postfix expression: ab+cd-*

Expected Output:

Prefix Expression: *+ab-cd

Infix Expression: ((a+b)*(c-d))

Output Screenshot:



```
Enter postfix expression (e.g., ab+cd-*): ab+cd-*

Prefix Expression:   *+ab-cd
Infix Expression:    ((a+b)*(c-d))
```

Program 3: Menu-Driven Binary Search Tree (Insert, Delete, Search, Kth Min, Kth Max)

File Name:

q3_bst_menu_driven.c

Program Code:

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
```

```
}

struct Node* insert(struct Node* root, int value) {
    if (root == NULL)
        return createNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);
    else
        printf("Value %d already exists in BST.\n", value);
    return root;
}

struct Node* findMin(struct Node* root) {
    while (root->left != NULL)
        root = root->left;
    return root;
}

struct Node* deleteNode(struct Node* root, int value) {
    if (root == NULL) {
        printf("Value %d not found in BST.\n", value);
        return root;
    }
    if (value < root->data)
        root->left = deleteNode(root->left, value);
    else if (value > root->data)
        root->right = deleteNode(root->right, value);
    else {
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            printf("Deleted successfully.\n");
            return temp;
        } else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            printf("Deleted successfully.\n");
            return temp;
        }
        struct Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

void search(struct Node* root, int value) {
    if (root == NULL) {
        printf("Value %d not found in BST.\n", value);
        return;
    }
    if (value == root->data) {
        printf("Value %d found in BST.\n", value);
        return;
    }
    if (value < root->data)
        search(root->left, value);
    else
        search(root->right, value);
}

int kthMinCount = 0;

void kthMin(struct Node* root, int k) {
    if (root == NULL || kthMinCount >= k)
        return;
    kthMin(root->left, k);
    kthMinCount++;
    if (kthMinCount == k) {
        printf("%dth Minimum element: %d\n", k, root->data);
        return;
    }
    kthMin(root->right, k);
}

int kthMaxCount = 0;

void kthMax(struct Node* root, int k) {
    if (root == NULL || kthMaxCount >= k)
        return;
    kthMax(root->right, k);
    kthMaxCount++;
    if (kthMaxCount == k) {
        printf("%dth Maximum element: %d\n", k, root->data);
        return;
    }
    kthMax(root->left, k);
}

void inorder(struct Node* root) {
    if (root == NULL)
        return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

int main() {
    struct Node* root = NULL;
    int choice, value, k;

    while (1) {
        printf("\n===== BST Menu =====\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Search\n");
        printf("4. Kth Minimum\n");
        printf("5. Kth Maximum\n");
        printf("6. Display (Inorder)\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                printf("Inserted %d.\n", value);
                break;

            case 2:
                printf("Enter value to delete: ");
                scanf("%d", &value);
                root = deleteNode(root, value);
                break;
            case 3:
                printf("Enter value to search: ");
                scanf("%d", &value);
                search(root, value);
                break;

            case 4:
                printf("Enter k for Kth Minimum: ");
                scanf("%d", &k);
                kthMinCount = 0;
                kthMin(root, k);
                break;

            case 5:
                printf("Enter k for Kth Maximum: ");
                scanf("%d", &k);
                kthMaxCount = 0;
                kthMax(root, k);
                break;

            case 6:
                printf("Inorder (sorted): ");
                inorder(root);
                printf("\n");
                break;

            case 7:
                printf("Exiting...\n");
                exit(0);

            default:
                printf("Invalid choice. Try again.\n");
        }
    }

    return 0;
}
```

Sample Test Input:

Choice: 1 → Insert: 50

Choice: 1 → Insert: 30

Choice: 1 → Insert: 70

Choice: 1 → Insert: 20

Choice: 1 → Insert: 40

Choice: 3 → Search: 40

Choice: 4 → Kth Min: 2

Choice: 5 → Kth Max: 2

Choice: 2 → Delete: 30

Choice: 6 → Display

Choice: 7 → Exit

Expected Output:

Inserted 50.

Inserted 30.

Inserted 70.

Inserted 20.

Inserted 40.

Value 40 found in BST.

2th Minimum element: 30

2th Maximum element: 50

Deleted successfully.

Inorder (sorted): 20 40 50 70

Output:

```
===== BST Menu =====
1. Insert
2. Delete
3. Search
4. Kth Minimum
5. Kth Maximum
```

```
6. Display (Inorder)
7. Exit
Enter your choice: 1
Enter value to insert: 50
Inserted 50.

===== BST Menu =====
1. Insert
2. Delete
3. Search
4. Kth Minimum
5. Kth Maximum
6. Display (Inorder)
7. Exit
Enter your choice: 1
Enter value to insert: 30
Inserted 30.

===== BST Menu =====
1. Insert
2. Delete
3. Search
4. Kth Minimum
5. Kth Maximum
6. Display (Inorder)
7. Exit
Enter your choice: 1
Enter value to insert: 70
Inserted 70.

===== BST Menu =====
1. Insert
2. Delete
3. Search
4. Kth Minimum
5. Kth Maximum
6. Display (Inorder)
7. Exit
Enter your choice: 1
Enter value to insert: 20
Inserted 20.

===== BST Menu =====
1. Insert
2. Delete
3. Search
4. Kth Minimum
5. Kth Maximum
6. Display (Inorder)
7. Exit
Enter your choice: 1
Enter value to insert: 40
Inserted 40.

===== BST Menu =====
1. Insert
2. Delete
3. Search
4. Kth Minimum
5. Kth Maximum
6. Display (Inorder)
7. Exit
Enter your choice: 6
Inorder (sorted): 20 30 40 50 70

===== BST Menu =====
1. Insert
2. Delete
3. Search
4. Kth Minimum
5. Kth Maximum
6. Display (Inorder)
7. Exit
Enter your choice: 3
Enter value to search: 40
Value 40 found in BST.

===== BST Menu =====
1. Insert
2. Delete
3. Search
4. Kth Minimum
5. Kth Maximum
6. Display (Inorder)
7. Exit
Enter your choice: 4
Enter k for Kth Minimum: 2
2th Minimum element: 30

===== BST Menu =====
1. Insert
2. Delete
3. Search
4. Kth Minimum
5. Kth Maximum
6. Display (Inorder)
7. Exit
Enter your choice: 5
Enter k for Kth Maximum: 2
2th Maximum element: 50

===== BST Menu =====
1. Insert
2. Delete
3. Search
4. Kth Minimum
5. Kth Maximum
6. Display (Inorder)
7. Exit
Enter your choice: 2
Enter value to delete: 30
Deleted successfully.

===== BST Menu =====
1. Insert
2. Delete
3. Search
4. Kth Minimum
5. Kth Maximum
6. Display (Inorder)
7. Exit
Enter your choice: 6
Inorder (sorted): 20 40 50 70

===== BST Menu =====
1. Insert
2. Delete
3. Search
4. Kth Minimum
```

```
5. Kth Maximum
6. Display (Inorder)
7. Exit
Enter your choice: 7
Exiting...
```