Binary Tree Operations

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

// Define a structure for a binary tree node
typedef struct Node {
    int data;
    struct Node *left;
    struct Node *right;
} Node;

// Function prototypes
Node *createNode(int value);
Node *insertNode(Node *root, int value);
void inorderTraversal(Node *root);
void preorderTraversal(Node *root);
void postorderTraversal(Node *root);
void freeTree(Node *root);

// Main function to demonstrate binary tree operations
int main() {
    Node *root = NULL;
    int choice, value;

    while (1) {
        printf("\nBinary Tree Operations:\n");
        printf("1. Insert Node\n");
        printf("2. In-order Traversal\n");
        printf("3. Pre-order Traversal\n");
        printf("4. Post-order Traversal\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                root = insertNode(root, value);
                break;
            case 2:
                printf("In-order Traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;
            case 3:
                printf("Pre-order Traversal: ");
                preorderTraversal(root);
                printf("\n");
                break;
            case 4:
                printf("Post-order Traversal: ");
                postorderTraversal(root);
                printf("\n");
                break;
            case 5:
                freeTree(root);
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}

// Create a new node
Node *createNode(int value) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Insert a node into the binary tree
Node *insertNode(Node *root, int value) {
```

```
    if (root == NULL) {
        return createNode(value); // If the tree is empty, create a new node
    }

    if (value < root->data) {
        root->left = insertNode(root->left, value); // Insert into the left subtree
    } else {
        root->right = insertNode(root->right, value); // Insert into the right subtree
    }

    return root;
}

// In-order traversal (Left, Root, Right)
void inorderTraversal(Node *root) {
    if (root == NULL) {
        return;
    }
    inorderTraversal(root->left);
    printf("%d ", root->data);
    inorderTraversal(root->right);
}

// Pre-order traversal (Root, Left, Right)
void preorderTraversal(Node *root) {
    if (root == NULL) {
        return;
    }
    printf("%d ", root->data);
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}

// Post-order traversal (Left, Right, Root)
void postorderTraversal(Node *root) {
    if (root == NULL) {
        return;
    }
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    printf("%d ", root->data);
}

// Free all nodes in the tree
void freeTree(Node *root) {
    if (root == NULL) {
        return;
    }
    freeTree(root->left);
    freeTree(root->right);
    free(root);
}
```

OUTPUT:

```
Binary Tree Operations:
1. Insert Node
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 1

Binary Tree Operations:
1. Insert Node
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 2

Binary Tree Operations:
1. Insert Node
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 3

Binary Tree Operations:
1. Insert Node
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 2

Binary Tree Operations:
1. Insert Node
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 3
Pre-order Traversal: 1 2 3 2
```