Implementation of Binary Search tree

Write a C program to implement a Binary Search Tree and perform the following operations.

(i) Insert

(ii) Delete

(iii) Search

(iv) Display

PROGRAMS:

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

// Define the structure for the nodes of the BST
typedef struct Node {
    int data;
    struct Node *left, *right;
} Node;

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Function to insert a new node with given data
Node* insert(Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }
    return root;
}

// Function to find the minimum value node in the tree
Node* findMin(Node* node) {
    Node* current = node;
    while (current && current->left != NULL) {
        current = current->left;
    }
    return current;
}

// Function to delete a node with given data
Node* deleteNode(Node* root, int data) {
    if (root == NULL) return root;

    if (data < root->data) {
        root->left = deleteNode(root->left, data);
    } else if (data > root->data) {
        root->right = deleteNode(root->right, data);
    } else {
        // Node with only one child or no child
        if (root->left == NULL) {
            Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            Node* temp = root->left;
            free(root);
            return temp;
        }

        // Node with two children: Get the inorder successor (smallest in the right subtree)
        Node* temp = findMin(root->right);

        // Copy the inorder successor's content to this node
        root->data = temp->data;

        // Delete the inorder successor
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

// Function to search for a node with given data
Node* search(Node* root, int data) {
    if (root == NULL || root->data == data) {
        return root;
    }
    if (data < root->data) {
        return search(root->left, data);
    }
    return search(root->right, data);
}

// Function to display the BST in inorder traversal
void inorderTraversal(Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

// Main function to demonstrate the BST operations
int main() {
    Node* root = NULL;
```

```
int choice, data;

while (1) {
    printf("\nBinary Search Tree Operations:\n");
    printf("1. Insert\n");
    printf("2. Delete\n");
    printf("3. Search\n");
    printf("4. Display\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter data to insert: ");
            scanf("%d", &data);
            root = insert(root, data);
            break;
        case 2:
            printf("Enter data to delete: ");
            scanf("%d", &data);
            root = deleteNode(root, data);
            break;
        case 3:
            printf("Enter data to search: ");
            scanf("%d", &data);
            Node* result = search(root, data);
            if (result != NULL) {
                printf("Data %d found in the tree.\n", data);
            } else {
                printf("Data %d not found in the tree.\n", data);
            }
            break;
        case 4:
            printf("Inorder traversal of the BST: ");
            inorderTraversal(root);
            printf("\n");
            break;
          case 5:
            exit(0);
          default:
            printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

OUTPUT:

```
Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 1
Enter data to insert: 50

Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 1
Enter data to insert: 30

Binary Search Tree Operations:
```

```
1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 1

Enter data to insert: 70


Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 1

Enter data to insert: 20


Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 1

Enter data to insert: 40


Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 1

Enter data to insert: 60
```

```
Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 1
Enter data to insert: 80

Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 4
Inorder traversal of the BST: 20 30 40 50 60 70 80

Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 3
Enter data to search: 60
Data 60 found in the tree.

Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. Display
```

```
5. Exit
Enter your choice: 3
Enter data to search: 25
Data 25 not found in the tree.

Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 2
Enter data to delete: 70

Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 4
Inorder traversal of the BST: 20 30 40 50 60 80

Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 5
```