Data Structures

Experiment no. 8

Develop code to implement different operations on BST using following operations – create, delete, display (traversal).

Q. WAP in C to implement different operations on BST.

Code:

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

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

// Function to create a new node
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;
}

// Function to insert a node in BST
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;
}

// Function to find minimum value node in BST
struct Node* minValueNode(struct Node* node) {
   struct Node* current = node;
```

```
    while (current && current->left != NULL)
        current = current->left;
    return current;
}

// Function to delete a node from BST
struct Node* deleteNode(struct Node* root, int value) {
    if (root == NULL)
        return root;

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

        // Node with two children
        struct Node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

// Functions for different traversals
void inorderTraversal(struct Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

void preorderTraversal(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
```

```
      preorderTraversal(root->right);
   }
}

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

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

   while(1) {
      printf("\n1. Insert\n");
      printf("2. Delete\n");
      printf("3. Inorder Traversal\n");
      printf("4. Preorder Traversal\n");
      printf("5. Postorder Traversal\n");
      printf("6. 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);
            break;

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

         case 3:
            printf("Inorder Traversal: ");
            inorderTraversal(root);
            printf("\n");
            break;

         case 4:
            printf("Preorder Traversal: ");
            preorderTraversal(root);
```

```
            printf("\n");
            break;

        case 5:
            printf("Postorder Traversal: ");
            postorderTraversal(root);
            printf("\n");
            break;

        case 6:
            exit(0);

        default:
            printf("Invalid choice!\n");
      }
    }
    return 0;
}
```

Output:

```
1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice: 1
Enter value to insert: 50

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice: 1
Enter value to insert: 30

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice: 1
Enter value to insert: 20
```

```
1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice: 1
Enter value to insert: 40

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice: 1
Enter value to insert: 70

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice: 1
Enter value to insert: 60

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice: 1
Enter value to insert: 80

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice: 3
Inorder Traversal: 20 30 40 50 60 70 80

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
```

```
Enter your choice: 4
Preorder Traversal: 50 30 20 40 70 60 80

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice: 5
Postorder Traversal: 20 40 30 60 80 70 50

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice: 2
Enter value to delete: 30

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice: 3
Inorder Traversal: 20 40 50 60 70 80

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice: 4
Preorder Traversal: 50 40 20 70 60 80

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice: 5
Postorder Traversal: 20 40 60 80 70 50

1. Insert
2. Delete
3. Inorder Traversal
```

```
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice: 6
```