Aim ↬ To perform basic operations on Binary Search Tree.

Objective ↬

Write a menu driven program to create a binary search tree of elements entered by the user. Implement the following operations on this binary search tree:

i. Insert a new node.
ii. Search a node having a key value.
iii. Delete a node having a key value.
iv. Display the elements in preorder.
v. Display the elements in inorder.
vi. Display the elements in postorder.

Software Required ↬ Visual Studio Code

Code ↬

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

typedef struct Node {
    int key;
    struct Node *left, *right;
} Node;

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

Node* insertNode(Node* root, int key) {
    if (root == NULL) return createNode(key);
    if (key < root->key) root->left = insertNode(root->left, key);
    else if (key > root->key) root->right = insertNode(root->right, key);
    return root;
}

Node* searchNode(Node* root, int key) {
```

```
    if (root == NULL || root->key == key) return root;
    if (key < root->key) return searchNode(root->left, key);
    return searchNode(root->right, key);
}

Node* findMin(Node* root) {
    while (root && root->left) root = root->left;
    return root;
}

Node* deleteNode(Node* root, int key) {
    if (root == NULL) return root;
    if (key < root->key) root->left = deleteNode(root->left, key);
    else if (key > root->key) root->right = deleteNode(root->right, key);
    else {
        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* temp = findMin(root->right);
        root->key = temp->key;
        root->right = deleteNode(root->right, temp->key);
    }
    return root;
}

void preorder(Node* root) {
    if (root != NULL) {
        printf("%d ", root->key);
        preorder(root->left);
        preorder(root->right);
    }
```

```
}

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

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

int main() {
    Node* root = NULL;
    int choice, key;
    Node* result;

    while (1) {
        printf("\n1. Insert a new node\n");
        printf("2. Search a node\n");
        printf("3. Delete a node\n");
        printf("4. Display in Preorder\n");
        printf("5. Display in Inorder\n");
        printf("6. Display in Postorder\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter key to insert: ");
```

```
            scanf("%d", &key);
            root = insertNode(root, key);
            break;
        case 2:
            printf("Enter key to search: ");
            scanf("%d", &key);
            result = searchNode(root, key);
            if (result) printf("Node with key %d found.\n", key);
            else printf("Node with key %d not found.\n", key);
            break;
        case 3:
            printf("Enter key to delete: ");
            scanf("%d", &key);
            root = deleteNode(root, key);
            printf("Node with key %d deleted if it existed.\n", key);
            break;
        case 4:
            printf("Preorder traversal: ");
            preorder(root);
            printf("\n");
            break;
        case 5:
            printf("Inorder traversal: ");
            inorder(root);
            printf("\n");
            break;
        case 6:
            printf("Postorder traversal: ");
            postorder(root);
            printf("\n");
            break;
        case 7:
            return 0;
        default:
            printf("Invalid choice\n");
    }
}
```

```
    return 0;
}
```

Output ↬

```
1. Insert a new node
2. Search a node
3. Delete a node
4. Display in Preorder
5. Display in Inorder
6. Display in Postorder
7. Exit
Enter your choice: 1
Enter key to insert: 33

1. Insert a new node
2. Search a node
3. Delete a node
4. Display in Preorder
5. Display in Inorder
6. Display in Postorder
7. Exit
Enter your choice: 1
Enter key to insert: 44
```

```
1. Insert a new node
2. Search a node
3. Delete a node
4. Display in Preorder
5. Display in Inorder
6. Display in Postorder
7. Exit
Enter your choice: 2
Enter key to search: 33
Node with key 33 found.

1. Insert a new node
2. Search a node
3. Delete a node
4. Display in Preorder
5. Display in Inorder
6. Display in Postorder
7. Exit
Enter your choice: 4
Preorder traversal: 33 44
```

```
1. Insert a new node
2. Search a node
3. Delete a node
4. Display in Preorder
5. Display in Inorder
6. Display in Postorder
7. Exit
Enter your choice: 5
Inorder traversal: 33 44

1. Insert a new node
2. Search a node
3. Delete a node
4. Display in Preorder
5. Display in Inorder
6. Display in Postorder
7. Exit
Enter your choice: 6
Postorder traversal: 44 33
```

Result ↬

The programs successfully implemented:

- **Queue Operations**: Insertion, deletion, and traversal using arrays.
- **Deque Operations**: Input Restricted, Output Restricted, and Unrestricted types using arrays and linked lists.

Conclusion ↬

The experiment demonstrated effective handling of basic queue and deque operations through both array and linked list implementations, enhancing understanding of data structure management.

Precautions ↬

- Validate all inputs and manage memory properly.
- Address edge cases like empty data structures.
- Implement error handling for common issues like overflow, underflow, and missing nodes.