#include <stdio.h>

#include <stdlib.h>

// binary search tree

struct node

{

    int data;

    struct node \*left;

    struct node \*right;

};

struct node \*createNode(int item)

{

    struct node \*temp = (struct node \*)malloc(sizeof(struct node));

    temp->data = item;

    temp->left = temp->right = NULL;

    return temp;

}

struct node \*insert(struct 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;

}

struct node \*search(struct node \*root, int data)

{

    if (root == NULL || root->data == data)

        return root;

    if (root->data < data)

        return search(root->right, data);

    return search(root->left, data);

}

void preorderTraversal(struct node \*root)

{

    if (root == NULL)

        return;

    printf("%d ", root->data);

    preorderTraversal(root->left);

    preorderTraversal(root->right);

}

void postorderTraversal(struct node \*root)

{

    if (root == NULL)

        return;

    postorderTraversal(root->left);

    postorderTraversal(root->right);

    printf("%d ", root->data);

}

void inorderTraversal(struct node \*root)

{

    if (root == NULL)

        return;

    inorderTraversal(root->left);

    printf("%d ", root->data);

    inorderTraversal(root->right);

}

struct node \*deleteNode(struct node \*root, int k)

{

    if (root == NULL)

        return root;

    if (k < root->data)

    {

        root->left = deleteNode(root->left, k);

    }

    else if (k > root->data)

    {

        root->right = deleteNode(root->right, k);

    }

    else

    {

        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;

        }

        struct node \*succParent = root;

        struct node \*succ = root->right;

        while (succ->left != NULL)

        {

            succParent = succ;

            succ = succ->left;

        }

        if (succParent != root)

            succParent->left = succ->right;

        else

            succParent->right = succ->right;

        root->data = succ->data;

        free(succ);

    }

    return root;

}

int main()

{

    int ch, item;

    struct node \*root = NULL;

    do

    {

        printf("1. Insert\n");

        printf("2. Preorder\n");

        printf("3. Postorder\n");

        printf("4. Inorder\n");

        printf("5. Search\n");

        printf("6. Delete\n");

        printf("7. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            printf("Enter value to be inserted: ");

            scanf("%d", &item);

            root = insert(root, item);

            break;

        case 2:

            printf("Preorder Traversal: ");

            preorderTraversal(root);

            printf("\n");

            break;

        case 3:

            printf("Postorder Traversal: ");

            postorderTraversal(root);

            printf("\n");

            break;

        case 4:

            printf("Inorder Traversal: ");

            inorderTraversal(root);

            printf("\n");

            break;

        case 5:

            printf("Enter element to be searched: ");

            scanf("%d", &item);

            if (search(root, item) == NULL)

            {

                printf("Not Found!\n");

            }

            else

            {

                printf("Found!\n");

            }

            break;

        case 6:

            printf("Enter element to be deleted: ");

            scanf("%d", &item);

            root = deleteNode(root, item);

            break;

        case 7:

            printf("Exiting the program.\n");

            break;

        default:

            printf("Invalid choice. Please try again.\n");

            break;

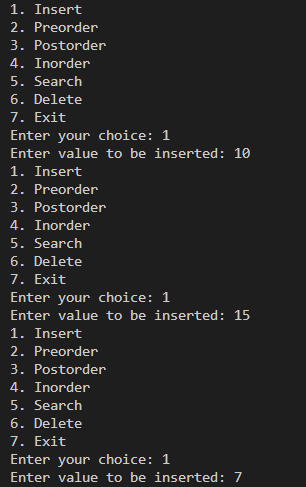
        }

    } while (ch != 7);

    return 0;

}

Output:



A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated