ASSIGNMENT 3

1) Write a C program to create a binary search tree using recursive function and display that.

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

// Define the structure of a node in the binary search tree
struct node {
  int data;
  struct node *left;
  struct node *right;
};

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

// Function to insert a node into the binary search tree
struct node* insert(struct node* root, int data) {
  if (root == NULL) {
    return createNode(data);
  } else {
    if (data <= root->data) {
      root->left = insert(root->left, data);
    } else {
      root->right = insert(root->right, data);
    }
    return root;
  }
}

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

int main() {
  struct node* root = NULL;
```

```c
    // Insert some elements into the binary search tree
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);

    // Display the binary search tree
    printf("Binary Search Tree (inorder traversal): ");
    inorderTraversal(root);
    printf("\n");

    return 0;
}
```

2) Write a C program to create a binary search tree using non-recursive function and display that.

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

// Define the structure of a node in the binary search tree
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

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

// Function to insert a node into the BST
void insertNode(struct Node** root, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;

    struct Node* current = *root;
    struct Node* parent = NULL;

    if (*root == NULL) {
```

```
        *root = newNode;
        return;
    }

    while (1) {
        parent = current;
        if (data < current->data) {
            current = current->left;
            if (current == NULL) {
                parent->left = newNode;
                return;
            }
        } else {
            current = current->right;
            if (current == NULL) {
                parent->right = newNode;
                return;
            }
        }
    }
}

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

int main() {
    struct Node* root = NULL;
    int data;

    // Inserting nodes into the BST
    printf("Enter elements to insert into BST (-1 to terminate):\n");
    while (1) {
        scanf("%d", &data);
        if (data == -1)
            break;
        insertNode(&root, data);
    }

    // Displaying the BST
    printf("Inorder traversal of BST: ");
    inorderTraversal(root);
    printf("\n");
```

```c
    return 0;
}
```

3) Write a C program to insert (by using a function) a specific element into an existing binary search
tree and then display that.

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

// Definition of a node in BST
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 = newNode->right = NULL;
    return newNode;
}

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

```
int main() {
    struct Node* root = NULL;
    int element;

    // Inserting elements into the BST
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    // Displaying the original BST
    printf("Original BST: ");
    inorderTraversal(root);
    printf("\n");

    // Inserting a specific element
    printf("Enter the element to insert: ");
    scanf("%d", &element);
    root = insert(root, element);

    // Displaying the modified BST
    printf("Modified BST after insertion: ");
    inorderTraversal(root);
    printf("\n");

    return 0;
}
```

4Write a C program to search an element in a BST and show the result.

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

// Definition of a node in BST
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 = newNode->right = NULL;
    return newNode;
}

// Function to insert a node into BST
struct Node* insert(struct Node* root, int value) {
    if (root == NULL) {
        return createNode(value);
```

```c
    }

    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 search for an element in BST
struct Node* search(struct Node* root, int key) {
    if (root == NULL || root->data == key) {
        return root;
    }

    if (key < root->data) {
        return search(root->left, key);
    } else {
        return search(root->right, key);
    }
}

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

int main() {
    struct Node* root = NULL;
    int elementToSearch;

    // Inserting elements into the BST
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    // Displaying the original BST
    printf("Original BST: ");
    inorderTraversal(root);
    printf("\n");

    // Prompting the user to enter an element to search
    printf("Enter the element to search: ");
    scanf("%d", &elementToSearch);

    // Searching for the element
    struct Node* result = search(root, elementToSearch);

    // Displaying the search result
    if (result != NULL) {
        printf("Element %d is found in the BST.\n", elementToSearch);
    } else {
        printf("Element %d is not found in the BST.\n", elementToSearch);
    }

    return 0;
```

}

5) Write a C program to take user name as input and display the sorted sequence of characters using
BST.

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

// Structure for a node in the binary search tree
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

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

// Function to insert a new node into the binary search tree
struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        root = createNode(data);
    } else if (data <= root->data) {
        root->left = insert(root->left, data);
    } else {
        root->right = insert(root->right, data);
    }
    return root;
}

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

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

    // Inserting elements into the binary search tree
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);

    // Displaying elements of the binary search tree
    printf("Inorder traversal of the BST: ");
    inorderTraversal(root);
    printf("\n");

    return 0;
}
```

6) Write a C program to sort a given set of integers using BST.

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

```c
#include <stdlib.h>

// Definition of a Node in BST
struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};

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

// Function to insert a node into BST
struct Node *insertNode(struct Node *root, int data) {
    if (root == NULL)
        return createNode(data);

    if (data < root->data)
        root->left = insertNode(root->left, data);
    else if (data > root->data)
        root->right = insertNode(root->right, data);

    return root;
}

// Function to traverse the BST in inorder and store elements in an array
void inorderTraversal(struct Node *root, int *arr, int *index) {
    if (root != NULL) {
        inorderTraversal(root->left, arr, index);
        arr[(*index)++] = root->data;
        inorderTraversal(root->right, arr, index);
    }
}

// Function to sort an array of integers using BST
void sortUsingBST(int arr[], int n) {
    struct Node *root = NULL;

    // Inserting elements into BST
    for (int i = 0; i < n; i++) {
        root = insertNode(root, arr[i]);
    }

    // Traversing BST in inorder and storing elements in array
    int index = 0;
    inorderTraversal(root, arr, &index);
}

// Function to print array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int arr[] = {5, 3, 8, 2, 7, 1, 9, 4};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original array: ");
    printArray(arr, n);
```

```
    sortUsingBST(arr, n);

    printf("Sorted array: ");
    printArray(arr, n);

    return 0;
}
```

7) Write a C program to display a BST using In-order, Pre-order, Post-order.

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

// Structure for a node in BST
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

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

// Function to insert a new node in BST
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;
}

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

// Function to perform pre-order traversal
void preorderTraversal(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

// Function to perform post-order traversal
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 values[] = {50, 30, 70, 20, 40, 60, 80};
    int n = sizeof(values) / sizeof(values[0]);

    // Inserting elements into BST
    for (int i = 0; i < n; i++) {
        root = insert(root, values[i]);
    }

    printf("In-order traversal: ");
    inorderTraversal(root);
    printf("\n");

    printf("Pre-order traversal: ");
    preorderTraversal(root);
    printf("\n");

    printf("Post-order traversal: ");
    postorderTraversal(root);
    printf("\n");

    return 0;
}
```

8) Write a C program to Count the number of nodes present in an existing BST and display the
highest element present in the BST.

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

// Structure for a node in BST
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

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

// Function to insert a new node in BST
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;
}

// Function to perform in-order traversal
void inorderTraversal(struct Node* root) {
```

```c
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

// Function to perform pre-order traversal
void preorderTraversal(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

// Function to perform post-order traversal
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 values[] = {50, 30, 70, 20, 40, 60, 80};
    int n = sizeof(values) / sizeof(values[0]);

    // Inserting elements into BST
    for (int i = 0; i < n; i++) {
        root = insert(root, values[i]);
    }

    printf("In-order traversal: ");
    inorderTraversal(root);
    printf("\n");

    printf("Pre-order traversal: ");
    preorderTraversal(root);
    printf("\n");

    printf("Post-order traversal: ");
    postorderTraversal(root);
    printf("\n");

    return 0;
}
```
9) Write a C program to prove that binary search tree is better than binary tree.
```c
#include <stdio.h>
#include <stdlib.h>

// Node structure for binary tree
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

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

```c
}

// Function to insert a new node into binary tree
struct Node* insertBT(struct Node* node, int data) {
    if (node == NULL)
        return createNode(data);
    if (data < node->data)
        node->left = insertBT(node->left, data);
    else if (data > node->data)
        node->right = insertBT(node->right, data);
    return node;
}

// Function to search for a key in binary tree
struct Node* searchBT(struct Node* root, int key) {
    if (root == NULL || root->data == key)
        return root;
    if (root->data < key)
        return searchBT(root->right, key);
    return searchBT(root->left, key);
}

// Function to create a binary search tree
struct Node* insertBST(struct Node* root, int data) {
    if (root == NULL)
        return createNode(data);
    if (data < root->data)
        root->left = insertBST(root->left, data);
    else if (data > root->data)
        root->right = insertBST(root->right, data);
    return root;
}

// Function to search for a key in binary search tree
struct Node* searchBST(struct Node* root, int key) {
    if (root == NULL || root->data == key)
        return root;
    if (root->data < key)
        return searchBST(root->right, key);
    return searchBST(root->left, key);
}

// Function to perform inorder traversal of a binary tree
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

int main() {
    struct Node* rootBT = NULL;
    struct Node* rootBST = NULL;

    // Insert elements into binary tree
    rootBT = insertBT(rootBT, 50);
    insertBT(rootBT, 30);
    insertBT(rootBT, 20);
    insertBT(rootBT, 40);
    insertBT(rootBT, 70);
    insertBT(rootBT, 60);
    insertBT(rootBT, 80);

    // Insert elements into binary search tree
    rootBST = insertBST(rootBST, 50);
```

```c
    insertBST(rootBST, 30);
    insertBST(rootBST, 20);
    insertBST(rootBST, 40);
    insertBST(rootBST, 70);
    insertBST(rootBST, 60);
    insertBST(rootBST, 80);

    printf("Binary Tree (BT) inorder traversal: ");
    inorder(rootBT);
    printf("\n");

    printf("Binary Search Tree (BST) inorder traversal: ");
    inorder(rootBST);
    printf("\n");

    // Search for an element in both trees
    int key = 70;
    struct Node* resultBT = searchBT(rootBT, key);
    struct Node* resultBST = searchBST(rootBST, key);

    if (resultBT != NULL)
        printf("%d found in Binary Tree (BT)\n", key);
    else
        printf("%d not found in Binary Tree (BT)\n", key);

    if (resultBST != NULL)
        printf("%d found in Binary Search Tree (BST)\n", key);
    else
        printf("%d not found in Binary Search Tree (BST)\n", key);

    return 0;
}
```

ASSIGNMENT 4

Write a C program to search an element recursively in a binary search tree.

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

// Define the structure of a node in the binary search tree
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

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

// Function to insert a new node into the binary search tree
struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    } else {
        if (data <= root->data) {
            root->left = insert(root->left, data);
        } else {
            root->right = insert(root->right, data);
        }
        return root;
    }
```

```
}

// Function to search for a key recursively in the binary search tree
struct Node* search(struct Node* root, int key) {
    // Base Cases: root is NULL or key is present at root
    if (root == NULL || root->data == key)
        return root;

    // Key is greater than root's key
    if (root->data < key)
        return search(root->right, key);

    // Key is smaller than root's key
    return search(root->left, key);
}

int main() {
    struct Node* root = NULL;
    root = insert(root, 5);
    insert(root, 3);
    insert(root, 7);
    insert(root, 1);
    insert(root, 4);

    int key = 4;
    struct Node* result = search(root, key);
    if (result != NULL) {
        printf("Element %d found in the binary search tree.\n", key);
    } else {
        printf("Element %d not found in the binary search tree.\n", key);
    }

    return 0;
}
```

2Write a C program to delete a node having two children from a binary search tree.Write a C program to delete a node having no child from a binary search tree.

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

// Structure for a node
struct node {
    int data;
    struct node *left;
    struct node *right;
};

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

// Function to insert a node into the binary search tree
struct node *insertNode(struct node *root, int data) {
    if (root == NULL) {
        return createNode(data);
    }

    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else if (data > root->data) {
        root->right = insertNode(root->right, data);
```

```
    }

    return root;
}

// Function to delete a node from the binary search tree
struct node *deleteNode(struct node *root, int key) {
    if (root == NULL) {
        return root;
    }

    if (key < root->data) {
        root->left = deleteNode(root->left, key);
    } else if (key > root->data) {
        root->right = deleteNode(root->right, key);
    } else {
        // Node with no child or one 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, get the inorder successor
        struct node *temp = root->right;
        while (temp->left != NULL) {
            temp = temp->left;
        }

        // 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 inorder traversal of the binary search tree
void inorderTraversal(struct node *root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

int main() {
    struct node *root = NULL;
    root = insertNode(root, 50);
    root = insertNode(root, 30);
    root = insertNode(root, 20);
    root = insertNode(root, 40);
    root = insertNode(root, 70);
    root = insertNode(root, 60);
    root = insertNode(root, 80);

    printf("Inorder traversal before deletion: ");
    inorderTraversal(root);
    printf("\n");

    int key = 20; // Key of the node to be deleted
```

```c
    root = deleteNode(root, key);

    printf("Inorder traversal after deletion of %d: ", key);
    inorderTraversal(root);
    printf("\n");

    return 0;
}
```

3) Write a C program to delete a node having one child from a binary search tree.

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

// Definition for a binary tree node
struct TreeNode {
    int data;
    struct TreeNode *left;
    struct TreeNode *right;
};

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

// Function to insert a new node into the binary search tree
struct TreeNode* insert(struct TreeNode* 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 the minimum value node in a given binary tree
struct TreeNode* minValueNode(struct TreeNode* node) {
    struct TreeNode* current = node;
    while (current && current->left != NULL) {
        current = current->left;
    }
    return current;
}

// Function to delete a node from the binary search tree
struct TreeNode* deleteNode(struct TreeNode* root, int key) {
    if (root == NULL) {
        return root;
    }
    // If the key to be deleted is smaller than the root's key, then it lies in the left subtree
    if (key < root->data) {
        root->left = deleteNode(root->left, key);
    }
    // If the key to be deleted is greater than the root's key, then it lies in the right subtree
    else if (key > root->data) {
        root->right = deleteNode(root->right, key);
    }
```

```
    // If the key is same as root's key, then this is the node to be
deleted
    else {
        // Node with only one child or no child
        if (root->left == NULL) {
            struct TreeNode* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct TreeNode* temp = root->left;
            free(root);
            return temp;
        }
        // Node with two children: Get the inorder successor (smallest
in the right subtree)
        struct TreeNode* temp = minValueNode(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 print inorder traversal of the binary search tree
void inorderTraversal(struct TreeNode* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

int main() {
    struct TreeNode* root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    printf("Inorder traversal before deletion: ");
    inorderTraversal(root);
    printf("\n");

    root = deleteNode(root, 20); // Deleting a node with one child
    printf("Inorder traversal after deletion: ");
    inorderTraversal(root);
    printf("\n");

    return 0;
}
```

4) Write a C program to delete a node having two children from a binary search tree.

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

// Structure of a node
struct node {
    int data;
    struct node *left;
    struct node *right;
};
```

```c
// 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 = newNode->right = NULL;
    return newNode;
}

// Function to find minimum value node in a subtree
struct node *minValueNode(struct node *root) {
    struct node *current = root;
    while (current->left != NULL)
        current = current->left;
    return current;
}

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

    // If the key to be deleted is smaller than the root's key, then it lies in the left subtree
    if (key < root->data)
        root->left = deleteNode(root->left, key);

    // If the key to be deleted is greater than the root's key, then it lies in the right subtree
    else if (key > root->data)
        root->right = deleteNode(root->right, key);

    // If key is same as root's key, then this is the node to be deleted
    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: Get the inorder successor (smallest in the right subtree)
        struct node *temp = minValueNode(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 perform inorder traversal of BST
void inorderTraversal(struct node *root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}
```

```
int main() {
    struct node *root = NULL;
    root = createNode(50);
    root->left = createNode(30);
    root->right = createNode(70);
    root->left->left = createNode(20);
    root->left->right = createNode(40);
    root->right->left = createNode(60);
    root->right->right = createNode(80);

    printf("Inorder traversal of the original BST: ");
    inorderTraversal(root);
    printf("\n");

    int key = 30;
    root = deleteNode(root, key);

    printf("Inorder traversal after deletion of node with key %d: ",
key);

    inorderTraversal(root);
    printf("\n");

    return 0;
}
```

5) Write a C program to delete a node from a binary search tree.

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

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

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

// Function to insert a node into the binary search tree
struct TreeNode* insert(struct TreeNode* 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 a BST
struct TreeNode* minValueNode(struct TreeNode* node) {
    struct TreeNode* current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}
```

```c
// Function to delete a node from BST
struct TreeNode* deleteNode(struct TreeNode* 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 {
        if (root->left == NULL) {
            struct TreeNode* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct TreeNode* temp = root->left;
            free(root);
            return temp;
        }
        struct TreeNode* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

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

int main() {
    struct TreeNode* root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    printf("Inorder traversal before deletion: ");
    inorderTraversal(root);
    printf("\n");

    root = deleteNode(root, 20);

    printf("Inorder traversal after deletion: ");
    inorderTraversal(root);
    printf("\n");

    return 0;
}
```