CS102 - Lab 11 - 28/03/2024

```c
//SHIVAMBU DEV PANDEY
//23BCS123
//CSE B
//LAB 11
//PROGRAM 1
//Date : 28/03/2024

/*
1.Write a menu driven C program to implement various operations of a
Binary
Search Tree such as Insert, Search, Display (in-order) and Deleting a
specified key.
*/

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

typedef struct node {
   int data;
   struct node* left;
   struct node* right;
}node;

node* createNode(int val) {
   node* newNode = (node*)malloc(sizeof(node));
   if (newNode == NULL) {
       printf("Memory allocation failed.\n");
       exit(1);
   }
   newNode->data = val;
   newNode->left = NULL;
   newNode->right = NULL;
   return newNode;
}

node* insert(node* root, int val) {
   if (root == NULL) {
       root = createNode(val);
```

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

node* search(node* root, int key) {
    if (root == NULL || root->data == key)
        return root;


    if (root->data < key)
        return search(root->right, key);


    return search(root->left, key);
}

node* minValueNode(node* root) {
    node* current = root;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}

node* deleteNode(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 {
        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 = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

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

int main() {
    node* root = NULL;
    int choice, key, data;
    while (1) {
        printf("\n1. Insert 2. Search 3. Delete 4. Display (In-order) 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);
                printf("Data inserted successfully.\n");
                break;
            case 2:
                printf("Enter key to search: ");
                scanf("%d", &key);
                node* result = search(root, key);
                if (result)
```

```
                printf("Key %d found in the tree.\n", key);
            else
                printf("Key %d not found in the tree.\n", key);
            break;
        case 3:
            printf("Enter key to delete: ");
            scanf("%d", &key);
            root = deleteNode(root, key);
            printf("Deleted node with key %d from the binary search
tree.\n", key);
            break;
        case 4:
            printf("In-order traversal of the binary search tree: ");
            inorder(root);
            printf("\n");
            break;
        case 5:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice.\n");
      }
   }
   return 0;
}
```

OUTPUT1:

```
1. Insert 2. Search 3. Delete 4. Display (In-order) 5. Exit
Enter your choice: 4
In-order traversal of the binary search tree: 8 10 12 15 16 18 19 20 25 30

1. Insert 2. Search 3. Delete 4. Display (In-order) 5. Exit
Enter your choice: 3
Enter key to delete: 20
Deleted node with key 20 from the binary search tree.

1. Insert 2. Search 3. Delete 4. Display (In-order) 5. Exit
Enter your choice: 4
In-order traversal of the binary search tree: 8 10 12 15 16 18 19 25 30

1. Insert 2. Search 3. Delete 4. Display (In-order) 5. Exit
Enter your choice: 2
Enter key to search: 20
Key 20 not found in the tree.
```

```c
//SHIVAMBU DEV PANDEY
//23BCS123
//CSE B
//LAB 11
//PROGRAM 2
//Date : 28/03/2024

/*
2.Write a C program to check if a given binary tree is a valid binary
search tree,
where the values of nodes in the left subtree are less than the node value
and
values in the right subtree are greater.
*/

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>

typedef struct node {
    int data;
    struct node* left;
    struct node* right;
}node;

node* createNode(int val) {
    node* newNode = (node*)malloc(sizeof(node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = val;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

bool checkBST(node* root, int min, int max) {
    if (root == NULL)
```

```
        return true;

    if (root->data < min || root->data > max)
        return false;

    return checkBST(root->left, min, root->data - 1) &&
checkBST(root->right, root->data + 1, max);
}

int main() {

    node* root = createNode(50);
    root->left = createNode(17);
    root->right = createNode(72);
    root->left->left = createNode(12);
    root->left->right = createNode(23);
    root->left->left->left = createNode(9);
    root->left->left->right = createNode(14);
    root->left->right->left = createNode(19);
    root->right->left = createNode(54);
    root->right->right = createNode(76);

    root->right->left->right = createNode(67);

    if (checkBST(root, INT_MIN, INT_MAX))
        printf("The binary tree is a valid BST.\n");
    else
        printf("The binary tree is not a valid BST.\n");

    return 0;
}
```

OUTPUT3:

```
PS C:\Users\shiva\Desktop\CODES> cd "c:\User
The binary tree is a valid BST.
PS C:\Users\shiva\Desktop\CODES\CS102\c>
```

```c
//SHIVAMBU DEV PANDEY
//23BCS123
//CSE B
//LAB 11
//PROGRAM 3
//Date : 28/03/2024

/*
3.Write a C program to check whether two BSTs are identical or not.
*/

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

typedef struct node {
    int data;
    struct node* left;
    struct node* right;
} node;

node* createNode(int val) {
    node* newNode = (node*)malloc(sizeof(node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = val;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

int idenBST(node* root1, node* root2) {
    if (root1 == NULL && root2 == NULL)
        return 1;
    else if (root1 == NULL || root2 == NULL)
        return 0;
    else if (root1->data != root2->data)
        return 0;
    else
        return idenBST(root1->left, root2->left) && idenBST(root1->right,
root2->right);
```

```c
}

int main() {
    node* root1 = createNode(6);
    root1->left = createNode(4);
    root1->right = createNode(9);
    root1->left->left = createNode(3);
    root1->left->right = createNode(5);

    node* root2 = createNode(6);
    root2->left = createNode(4);
    root2->right = createNode(9);
    root2->left->left = createNode(3);
    root2->left->right = createNode(5);

    if (idenBST(root1, root2))
        printf("BSTs are identical.\n");
    else
        printf("BSTs are not identical.\n");

    return 0;
}
```

OUTPUT3:

```
PS C:\Users\shiva\Desktop\CODES\CS102\c> cd
BSTs are identical.
PS C:\Users\shiva\Desktop\CODES\CS102\c>
```