	SVKM's NMIMS School of Technology Management & Engineering Navi Mumbai Campus
	Department of Computer Engineering

Name: Aryan Khatu	SAP ID: 70562200066
Semester: III	Year: II
Subject: Data Structures andAlgorithm	Roll No.: A095
Practical: 8	Date: 30/09/2023
Batch: 2	

Aim: Implementation of Binary Search Tree: Insertion, deletion and Search operation on the tree data structure.

Code/Implementation:

```
#include "iostream"
#include "cstdlib"
using namespace std;
struct Tree {
    int data;
    struct Tree *left, *right;
};
struct Tree* root = nullptr;
void preorder(struct Tree* current) {
    if(current == nullptr) {
        cout<<"Tree is empty!"<<endl;
        return;
    }
    cout<<"\t"<<current->data;
    preorder(current->left);
    preorder(current->right);
}
void inorder(struct Tree* current){
    if(current == nullptr) {
        cout<<"Tree is empty!"<<endl;
        return;
    }
    inorder(current->left);
    cout<<"\t"<<current->data;
    inorder(current->right);
}
void postorder(struct Tree* current){
    if(current == nullptr) {
        cout<<"Tree is empty!"<<endl;
        return;
    }
```

```
    postorder(current->left);
    postorder(current->right);
    cout<<"\t"<<current->data;
}
void insert_val(int val) {
    struct Tree* newNode = (struct Tree*) malloc(sizeof(struct Tree)), *current = root, *prev = root;
    newNode->data = val;
    newNode->left = nullptr;
    newNode->right = nullptr;
    if(root == nullptr) {
        root = newNode;
        cout << val << " is created as root!" << endl;
    }
    else {
        while(current != nullptr) {
            prev = current;
            if (val == current->data) {
                cout << val << " already in the tree!" << endl;
                return;
            }
            else if (val < current->data)
                current = current->left;
            else
                current = current->right;
        }
        if(val < prev->data)
            prev->left = newNode;
        else
            prev->right = newNode;
        cout<< val << " added to the tree!"<<endl;
    }
}
struct Tree* inorder_pred_succ(struct Tree* current) {
    int ch;
    cout << "Repalce element with:\n1. Previous\n2. Next\nEnter choice: ";
    cin >> ch;
    if(ch == 1) {
        if(current->left != nullptr) {
            current = current->left;
            while(current->right != nullptr)
                current = current->right;
        }
```

```
    }
    else {
        if(current->right != nullptr) {
            current = current->right;
            while(current->left != nullptr)
                current = current->left;
        }
    }
    return current;
}
void delete_val(int val, struct Tree* curr) {
    struct Tree* temp;
    if(curr == nullptr) {
        cout << "Tree is empty!" << endl;
        return;
    }
    else if(curr->left == nullptr && curr->right == nullptr) {
        cout<<"Element deleted!"<<endl;
        delete(curr);
        return;
    }
```

```
    else {
        while(curr != nullptr) {
            if(val < curr->data)
                curr = curr->left;
            else if(val > curr->data)
                curr = curr->right;
            else {
                temp = inorder_pred_succ(curr);
                curr->data = temp->data;
                if (val > curr->data)
                    delete_val(temp->data, curr->left);
                else
                    delete_val(temp->data, curr->right);
            }
        }
    }
}
void search(int val) {
    if(root == nullptr){
        cout << "Tree is empty! No element to find." << endl;
        return;
    }
    struct Tree* temp = root, *parent = nullptr;
    while(temp != nullptr && temp->data != val) {
        parent = temp;
        if(val < temp->data)
            temp = temp->left;
        else
            temp = temp->right;
    }
    if(temp == nullptr)
        cout<<"Element not found!"<<endl;
    else {
        cout << "Element found!\nParent: " << parent->data<<endl;
        if(temp->data == parent->left->data)
            cout << "Sibling: " << parent->right->data;
        else
            cout << "Sibling: " << parent->left->data;
        if(temp->left != nullptr || temp->right != nullptr)
            cout<<"\nChildren: "<<temp->left->data<<" and "<<temp->right->data<<endl;
        else
            cout<<"Leaf Node! Hence no children."<<endl;
```

```
    }

}
int main()
{
    int choice, case_choice, val;
    do {
        cout << "1. Insert\n2. Search\n3. Display\n4. Delete\n5. Exit\nEnter choice: ";
        cin >> choice;
        switch (choice) {
            case 3:
                cout << "1. Preorder\n2. Inorder\n3. Postorder\nEnter choice: ";
                cin >> case_choice;
                switch (case_choice) {
                    case 1:
                        cout << "Pre-Order Traversal: ";
                        preorder(root);
                        cout<<endl;
                        break;
                    case 2:
                        cout << "In-Order Traversal: ";
```

```
                    inorder(root);
                    cout<<endl;
                    break;
                case 3:
                    cout << "Post-Order Traversal: ";
                    postorder(root);
                    cout<<endl;
                    break;
                default:
                    cout << "No such order! Please try again." << endl;
            }
            break;
        case 1:
            cout << "Enter value: ";
            cin >> val;
            insert_val(val);
            break;
        case 2:
            cout << "Enter value: ";
            cin >> val;
            search(val);
            break;
        case 4:
            cout<<"Enter value: ";
            cin>>val;
            delete_val(val, root);
            break;
        case 5:
            cout << "Program Run Successful" << endl;
            break;
        default:
            cout << "Wrong Choice! Please try again." << endl;
    }
  } while(choice != 5);
}
```

Output:

```
1. Insert   2. Search   3. Display  4. Delete   5. Exit Enter choice: 1
Enter value: 50
50 is created as root!
1. Insert   2. Search   3. Display  4. Delete   5. Exit Enter choice: 1
Enter value: 30
30 added to the tree!
1. Insert   2. Search   3. Display  4. Delete   5. Exit Enter choice: 1
Enter value: 80
80 added to the tree!
1. Insert   2. Search   3. Display  4. Delete   5. Exit Enter choice: 1
Enter value: 20
20 added to the tree!
1. Insert   2. Search   3. Display  4. Delete   5. Exit Enter choice: 1
Enter value: 35
35 added to the tree!
1. Insert   2. Search   3. Display  4. Delete   5. Exit Enter choice: 1
Enter value: 70
70 added to the tree!
1. Insert   2. Search   3. Display  4. Delete   5. Exit Enter choice: 1
Enter value: 90
90 added to the tree!
1. Insert   2. Search   3. Display  4. Delete   5. Exit Enter choice: 3
1. Preorder 2. Inorder  3. Postorder    Enter choice: 2
In-Order Traversal:     20  30  35  50  70  80  90
1. Insert   2. Search   3. Display  4. Delete   5. Exit Enter choice: 3
1. Preorder 2. Inorder  3. Postorder    Enter choice: 1
Pre-Order Traversal:    50  30  20  35  80  70  90
1. Insert   2. Search   3. Display  4. Delete   5. Exit Enter choice: 3
1. Preorder 2. Inorder  3. Postorder    Enter choice: 3
Post-Order Traversal:   20  35  30  70  90  80  50
1. Insert   2. Search   3. Display  4. Delete   5. Exit Enter choice: 2
Enter value: 30
Element found!
Parent: 50
Sibling: 80
Children: 20 and 35
```

```
1. Insert   2. Search   3. Display  4. Delete   5. Exit Enter choice: 4
Enter value: 30
Repalce element with:   1. Previous 2. Next Enter choice: 1
Element deleted!
1. Insert   2. Search   3. Display  4. Delete   5. Exit Enter choice: 5
Program Run Successful

Process finished with exit code 0
```

****