

DSA Practical\codes\Practical1st.cpp

```
1  #include <iostream>
2  using namespace std;
3  struct node
4  {
5      int data;
6      node *left, *right;
7      node(int val)
8      {
9          data = val;
10         left = NULL;
11         right = NULL;
12     }
13 };
14 void inorder(node *root)
15 {
16     if (root == NULL)
17     {
18         return;
19     }
20     inorder(root->left);
21     cout << root->data << " ";
22     inorder(root->right);
23 }
24 void preorder(node *root)
25 {
26     if (root == NULL)
27     {
28         return;
29     }
30     cout << root->data;
31     cout << " ";
32     inorder(root->left);
33     inorder(root->right);
34 }
35 void postorder(node *root)
36 {
37     if (root == NULL)
38     {
39         return;
40     }
41     inorder(root->left);
42     inorder(root->right);
43     cout << " ";
44     cout << root->data;
45 }
46 node *search(node *root, int key)
47 {
48     if (root == NULL)
```

```

49     {
50         return NULL;
51     }
52     else
53     {
54         if (root->data == key)
55         {
56             return root;
57         }
58         else if (root->data > key)
59         {
60             return search(root->left, key);
61         }
62         else
63         {
64             return search(root->right, key);
65         }
66     }
67 }
68 node *insert(node *root, int val)
69 {
70     if (root == NULL)
71     {
72         root = new node(val);
73     }
74     else if (val > root->data)
75     {
76         root->right = insert(root->right, val);
77     }
78     else if (val < root->data)
79     {
80         root->left = insert(root->left, val);
81     }
82     else if (root->data == val)
83     {
84         cout << "Duplicate Value\n";
85     }
86     else
87     {
88         cout << "Invalid Entity";
89     }
90     return root;
91 }
92 node *inordersucc(node *root)
93 {
94     node *curr = root;
95     while (curr && curr->left != NULL)
96     {
97         curr = curr->left;
98     }

```

```

99     return curr;
100 }
101 node *deletenode(node *root, int del)
102 {
103     if (del < root->data)
104     {
105         root->left = deletenode(root->left, del);
106     }
107     else if (del > root->data)
108     {
109         root->right = deletenode(root->right, del);
110     }
111     else
112     {
113         if (root->left == NULL)
114         {
115             node *temp = root->right;
116             free(root);
117             cout << "Node deleted"<<endl;
118             return temp;
119         }
120         else if (root->right == NULL)
121         {
122             node *temp = root->left;
123             free(root);
124             cout << "Node deleted"<<endl;
125             return temp;
126         }
127         node *temp = inordersucc(root->right);
128         root->data = temp->data;
129         root->right = deletenode(root->right, temp->data);
130     }
131     return root;
132 }
133 int main()
134 {
135     int x, value;
136     char ch, Y, N, n, y;
137     node *root = NULL;
138     cout << "Welcome to Binary Search Tree" << endl;
139     while (true)
140     {
141         cout << "1]Insert node in tree" << endl
142             << "2]Search for node in tree" << endl
143             << "3]Traverse the tree" << endl
144             << "4]Delete the node inside the tree" << endl
145             << "5]To exit Program" << endl;
146         cout << "Enter your choice:";
147         cin >> x;
148         switch (x)

```

```

149 {
150     case 1:
151         cout << "Do you want to insert the node" << endl;
152         while (true)
153         {
154             cout << "Y/y to continue or N/n to exit:";
155             cin >> ch;
156             if (ch == 'Y' || ch == 'y')
157             {
158                 cout << "Enter value to insert" << endl;
159                 cin >> value;
160                 root = insert(root, value);
161             }
162             else if (ch == 'N' || ch == 'n')
163             {
164                 cout << "Exiting...."<<endl;
165                 break;
166             }
167             else
168             {
169                 cout << "Invalid Entity" << endl;
170             }
171         }
172         break;
173     case 2:
174         cout << "Enter value to search for:" << endl;
175         cin >> value;
176         search(root, value);
177         if (search(root, value) == NULL)
178         {
179             cout << "Element not found"<<endl;
180         }
181         else
182         {
183             cout << "Element found"<<endl;
184         }
185         break;
186     case 3:
187         cout << "Enter your choice:"<< endl;
188         cout << "1]Inorder Traversal" << endl
189             << "2]Preorder Traversal " << endl
190             << "3]Postorder Traversal" << endl;
191         cin >> x;
192         switch (x)
193         {
194             case 1:
195                 inorder(root);
196                 cout<<endl;
197                 break;
198             case 2:

```

```
199         preorder(root);
200         cout<<endl;
201         break;
202     case 3:
203         postorder(root);
204         cout<<endl;
205         break;
206     default:
207         cout << "Invalid choice";
208     }
209     break;
210 case 4:
211     cout << "Enter value to delete" << endl;
212     cin >> value;
213     deletenode(root, value);
214     break;
215 case 5:
216     cout << "Exiting....";
217     break;
218 default:
219     cout << "Invalid choice";
220     break;
221 }
222 if (x == 5)
223 {
224     break;
225 }
226 }
227 return 0;
228 }
229
```