



```
1  #include <stdio.h>
2
3  typedef struct node
4  {
5      int data;
6      struct node *left, *right;
7      int ht;
8  }
9
10 node;
11
12 node* insert(node *, int);
13 node* Delete(node *, int);
14 void preorder(node*);
15 void inorder(node*);
16 int height(node*);
17 node* rotateright(node*);
18 node* rotateleft(node*);
19 node* RR(node*);
20 node* LL(node*);
21 node* LR(node*);
22 node* RL(node*);
23 int BF(node*);
24
25 int main()
26 {
27     node *root = NULL;
28     int x, n, i, op;
29     do {
30         printf("\n1)Create:");
31         printf("\n2)Insert:");
32         printf("\n3>Delete:");
33         printf("\n4)Print:");
34         printf("\n5)Quit:");
35         printf("\n\nEnter Your Choice:");
36         scanf("%d", &op);
37         switch (op)
38         {
39             case 1:
40                 printf("\nEnter no. of elements:");
41                 scanf("%d", &n);
42                 printf("\nEnter tree data:");
43                 root = NULL;
44                 for (i = 0; i < n; i++)
45                 {
46                     scanf("%d", &x);
47                     root = insert(root, x);
48                 }
49
50                 break;
51             case 2:
52                 printf("\nEnter a data:");
53                 scanf("%d", &x);
54                 root = insert(root, x);
55                 break;
56             case 3:
57                 printf("\nEnter a data:");
58                 scanf("%d", &x);
59                 root = Delete(root, x);
60                 break;
61             case 4:
62                 printf("\nPreorder sequence:\n");
63                 preorder(root);
64                 printf("\n\nInorder sequence:\n");
65                 inorder(root);
66                 printf("\n");
67                 break;
68         }
69     } while (op != 5);
70     return 0;
71 }
```

```

73 node* insert(node *T, int x)
74 {
75     if (T == NULL)
76     {
77         T = (node*) malloc(sizeof(node));
78         T->data = x;
79         T->left = NULL;
80         T->right = NULL;
81     }
82     else
83     if (x > T->data) // insert in right subtree
84     {
85         T->right = insert(T->right, x);
86         if (BF(T) == -2)
87             if (x > T->right->data)
88                 T = RR(T);
89             else
90                 T = RL(T);
91     }
92     else
93     if (x < T->data)
94     {
95         T->left = insert(T->left, x);
96         if (BF(T) == 2)
97             if (x < T->left->data)
98                 T = LL(T);
99             else
100                 T = LR(T);
101     }
102     T->ht = height(T);
103     return (T);
104 }
105
106
107 node* Delete(node *T, int x)
108 {
109     node * p;
110     if (T == NULL)
111     {
112         return NULL;
113     }
114     else
115     if (x > T->data) // insert in right subtree
116     {
117         T->right = Delete(T->right, x);
118         if (BF(T) == 2)
119             if (BF(T->left) >= 0)
120                 T = LL(T);
121             else
122                 T = LR(T);
123     }
124     else
125     if (x < T->data)
126     {
127         T->left = Delete(T->left, x);
128         if (BF(T) == -2) //Rebalance during windup
129             if (BF(T->right) <= 0)
130                 T = RR(T);
131             else
132                 T = RL(T);
133     }
134     else
135     {
136         //data to be deleted is found
137         if (T->right != NULL)
138         {
139             //delete its inorder successor
140             p = T->right;
141             while (p->left != NULL)
142                 p = p->left;
143             T->data = p->data;
144             T->right = Delete(T->right, p->data);
145             if (BF(T) == 2) //Rebalance during windup
146                 if (BF(T->left) >= 0)
147                     T = LL(T);
148                 else
149                     T = LR(T);
150         }
151         else
152             return (T->left);
153     }
154     T->ht = height(T);
155     return (T);
156 }
157
158
159 int height(node *T)
160 {
161     int lh, rh;
162     if (T == NULL)
163         return (0);
164     if (T->left == NULL)
165         lh = 0;
166     else
167         lh = 1 + T->left->ht;
168     if (T->right == NULL)
169         rh = 0;
170     else
171         rh = 1 + T->right->ht;
172     if (lh > rh)
173         return (lh);
174     return (rh);
175 }
176
177 node* rotateright(node *x)
178 {
179     node * y;
180     y = x->left;
181     x->left = y->right;
182     y->right = x;
183     x->ht = height(x);
184     y->ht = height(y);
185     return (y);
186 }
187
188 node* rotateleft(node *x)
189 {
190     node * y;
191     y = x->right;
192     x->right = y->left;
193     y->left = x;
194     x->ht = height(x);
195     y->ht = height(y);
196     return (y);
197 }
198
199 node* RR(node *T)
200 {
201     T = rotateleft(T);
202     return (T);
203 }
204
205 node* LL(node *T)
206 {
207     T = rotateright(T);
208     return (T);
209 }
210
211 node* LR(node *T)
212 {
213     T->left = rotateleft(T->left);
214     T = rotateright(T);
215     return (T);
216 }
217
218 node* RL(node *T)
219 {
220     T->right = rotateright(T->right);
221     T = rotateleft(T);
222     return (T);
223 }
224
225 int BF(node *T)
226 {
227     int lh, rh;
228     if (T == NULL)
229         return (0);
230
231     if (T->left == NULL)
232         lh = 0;
233     else
234         lh = 1 + T->left->ht;
235
236     if (T->right == NULL)
237         rh = 0;
238     else
239         rh = 1 + T->right->ht;
240
241     return (lh - rh);
242 }
243
244 void preorder(node *T)
245 {
246     if (T != NULL)
247     {
248         printf("%d(Bf=%d)", T->data, BF(T));
249         preorder(T->left);
250         preorder(T->right);
251     }
252 }
253
254 void inorder(node *T)
255 {
256     if (T != NULL)
257     {
258         inorder(T->left);
259         printf("%d(Bf=%d)", T->data, BF(T));
260         inorder(T->right);
261     }
262 }

```

