

```

1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node *parent;
7      Node *left;
8      Node *right;
9      int color;
10 };
11
12 typedef Node *NodePtr;
13
14 class RedBlackTree {
15 private:
16     NodePtr root;
17     NodePtr TNULL;
18
19     void initializeNULLNode(NodePtr node, NodePtr parent) {
20         node->data = 0;
21         node->parent = parent;
22         node->left = nullptr;
23         node->right = nullptr;
24         node->color = 0;
25     }
26
27     void rbTransplant(NodePtr u, NodePtr v) {
28         if (u->parent == nullptr) {
29             root = v;
30         } else if (u == u->parent->left) {
31             u->parent->left = v;
32         } else {
33             u->parent->right = v;
34         }
35         v->parent = u->parent;
36     }
37
38     // For balancing the tree after insertion
39     void insertFix(NodePtr k) {
40         NodePtr u;
41         while (k->parent->color == 1) {
42             if (k->parent == k->parent->parent->right) {
43                 u = k->parent->parent->left;
44                 if (u->color == 1) {
45                     u->color = 0;
46                     k->parent->color = 0;
47                     k->parent->parent->color = 1;
48                     k = k->parent->parent;
49                 } else {
50                     if (k == k->parent->left) {
51                         k = k->parent;
52                         rightRotate(k);
53                     }
54                     k->parent->color = 0;
55                     k->parent->parent->color = 1;
56                     leftRotate(k->parent->parent);
57                 }
58             } else {
59                 u = k->parent->parent->right;
60
61                 if (u->color == 1) {
62                     u->color = 0;
63                     k->parent->color = 0;
64                     k->parent->parent->color = 1;
65                     k = k->parent->parent;
66                 } else {
67                     if (k == k->parent->right) {
68                         k = k->parent;
69                         leftRotate(k);
70                     }
71                     k->parent->color = 0;
72                     k->parent->parent->color = 1;
73                     rightRotate(k->parent->parent);
74                 }
75             }
76             if (k == root) {
77                 break;
78             }
79         }
80         root->color = 0;
81     }
82
83     void printHelper(NodePtr root, string indent, bool last) {
84         if (root != TNULL) {
85             cout << indent;
86             if (last) {
87                 cout << "R----";
88                 indent += "    ";
89             } else {
90                 cout << "L----";
91                 indent += "|    ";
92             }
93
94             string sColor = root->color ? "RED" : "BLACK";
95             cout << root->data << "(" << sColor << ")" << endl;
96             printHelper(root->left, indent, false);
97             printHelper(root->right, indent, true);
98         }
99     }
100
101 public:
102     RedBlackTree() {
103         TNULL = new Node;
104         TNULL->color = 0;
105         TNULL->left = nullptr;
106         TNULL->right = nullptr;
107         root = TNULL;
108     }
109
110     NodePtr minimum(NodePtr node) {
111         while (node->left != TNULL) {
112             node = node->left;
113         }
114         return node;
115     }
116
117     NodePtr maximum(NodePtr node) {
118         while (node->right != TNULL) {
119             node = node->right;
120         }
121         return node;
122     }

```

```

123
124 NodePtr successor(NodePtr x) {
125     if (x->right != TNULL) {
126         return minimum(x->right);
127     }
128
129     NodePtr y = x->parent;
130     while (y != TNULL && x == y->right) {
131         x = y;
132         y = y->parent;
133     }
134     return y;
135 }
136
137 NodePtr predecessor(NodePtr x) {
138     if (x->left != TNULL) {
139         return maximum(x->left);
140     }
141
142     NodePtr y = x->parent;
143     while (y != TNULL && x == y->left) {
144         x = y;
145         y = y->parent;
146     }
147
148     return y;
149 }
150
151 void leftRotate(NodePtr x) {
152     NodePtr y = x->right;
153     x->right = y->left;
154     if (y->left != TNULL) {
155         y->left->parent = x;
156     }
157     y->parent = x->parent;
158     if (x->parent == nullptr) {
159         this->root = y;
160     } else if (x == x->parent->left) {
161         x->parent->left = y;
162     } else {
163         x->parent->right = y;
164     }
165     y->left = x;
166     x->parent = y;
167 }
168
169 void rightRotate(NodePtr x) {
170     NodePtr y = x->left;
171     x->left = y->right;
172     if (y->right != TNULL) {
173         y->right->parent = x;
174     }
175     y->parent = x->parent;
176     if (x->parent == nullptr) {
177         this->root = y;
178     } else if (x == x->parent->right) {
179         x->parent->right = y;
180     } else {
181         x->parent->left = y;
182     }
183     y->right = x;
184     x->parent = y;
185 }
186
187 // Inserting a node
188 void insert(int key) {
189     NodePtr node = new Node;
190     node->parent = nullptr;
191     node->left = TNULL;
192     node->right = TNULL;
193     node->color = 1;
194     node->data = key;
195
196     NodePtr y = nullptr;
197     NodePtr x = this->root;
198
199     while (x != TNULL) {
200         y = x;
201         if (node->data < x->data) {
202             x = x->left;
203         } else {
204             x = x->right;
205         }
206     }
207
208     node->parent = y;
209     if (y == nullptr) {
210         root = node;
211     } else if (node->data < y->data) {
212         y->left = node;
213     } else {
214         y->right = node;
215     }
216
217     if (node->parent == nullptr) {
218         node->color = 0;
219         return;
220     }
221
222     if (node->parent->parent == nullptr) {
223         return;
224     }
225
226     insertFix(node);
227 }
228
229 NodePtr getRoot() {
230     return this->root;
231 }
232
233 void printTree() {
234     if (root) {
235         printHelper(this->root, "", true);
236     }
237 }
238 };
239
240 int main() {
241     RedBlackTree bst;
242     int n;
243     cout << "Enter Number of Nodes: ";
244     cin >> n;
245
246     while (n-- > 0) {
247         int k;
248         cin >> k;
249         bst.insert(k);
250     }
251
252     bst.printTree();
253 }

```

× Terminal

Enter Number of Nodes: 6

1

3

5

4

6

2

R-----3(BLACK)

 L-----1(BLACK)

 | R-----2(RED)

 R-----5(BLACK)

 L-----4(RED)

 R-----6(RED)