```cpp
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node *parent;
    Node *left;
    Node *right;
    int color;
};

typedef Node *NodePtr;

class RedBlackTree {
  private:
    NodePtr root;
    NodePtr TNULL;

    void initializeNULLNode(NodePtr node, NodePtr parent) {
        node->data = 0;
        node->parent = parent;
        node->left = nullptr;
        node->right = nullptr;
        node->color = 0;
    }

    void rbTransplant(NodePtr u, NodePtr v) {
        if (u->parent == nullptr) {
            root = v;
        } else if (u == u->parent->left) {
            u->parent->left = v;
        } else {
            u->parent->right = v;
        }
        v->parent = u->parent;
    }

    // For balancing the tree after insertion
    void insertFix(NodePtr k) {
        NodePtr u;
        while (k->parent->color == 1) {
            if (k->parent == k->parent->parent->right) {
                u = k->parent->parent->left;
                if (u->color == 1) {
                    u->color = 0;
                    k->parent->color = 0;
                    k->parent->parent->color = 1;
                    k = k->parent->parent;
                } else {
                    if (k == k->parent->left) {
                        k = k->parent;
                        rightRotate(k);
                    }
                    k->parent->color = 0;
                    k->parent->parent->color = 1;
                    leftRotate(k->parent->parent);
                }
            } else {
                u = k->parent->parent->right;

                if (u->color == 1) {
                    u->color = 0;
                    k->parent->color = 0;
                    k->parent->parent->color = 1;
                    k = k->parent->parent;
                } else {
                    if (k == k->parent->right) {
                        k = k->parent;
                        leftRotate(k);
                    }
                    k->parent->color = 0;
                    k->parent->parent->color = 1;
                    rightRotate(k->parent->parent);
                }
            }
            if (k == root) {
                break;
            }
        }
        root->color = 0;
    }

    void printHelper(NodePtr root, string indent, bool last) {
        if (root != TNULL) {
            cout << indent;
            if (last) {
                cout << "R----";
                indent += "    ";
            } else {
                cout << "L----";
                indent += "|   ";
            }

            string sColor = root->color ? "RED" : "BLACK";
            cout << root->data << "(" << sColor << ")" << endl;
            printHelper(root->left, indent, false);
            printHelper(root->right, indent, true);
        }
    }

  public:
    RedBlackTree() {
        TNULL = new Node;
        TNULL->color = 0;
        TNULL->left = nullptr;
        TNULL->right = nullptr;
        root = TNULL;
    }

    NodePtr minimum(NodePtr node) {
        while (node->left != TNULL) {
            node = node->left;
        }
        return node;
    }

    NodePtr maximum(NodePtr node) {
        while (node->right != TNULL) {
            node = node->right;
        }
        return node;
    }
```

```
    NodePtr successor(NodePtr x) {
        if (x->right != TNULL) {
            return minimum(x->right);
        }

        NodePtr y = x->parent;
        while (y != TNULL && x == y->right) {
            x = y;
            y = y->parent;
        }
        return y;
    }

    NodePtr predecessor(NodePtr x) {
        if (x->left != TNULL) {
            return maximum(x->left);
        }

        NodePtr y = x->parent;
        while (y != TNULL && x == y->left) {
            x = y;
            y = y->parent;
        }

        return y;
    }

    void leftRotate(NodePtr x) {
        NodePtr y = x->right;
        x->right = y->left;
        if (y->left != TNULL) {
            y->left->parent = x;
        }
        y->parent = x->parent;
        if (x->parent == nullptr) {
            this->root = y;
        } else if (x == x->parent->left) {
            x->parent->left = y;
        } else {
            x->parent->right = y;
        }
        y->left = x;
        x->parent = y;
    }

    void rightRotate(NodePtr x) {
        NodePtr y = x->left;
        x->left = y->right;
        if (y->right != TNULL) {
            y->right->parent = x;
        }
        y->parent = x->parent;
        if (x->parent == nullptr) {
            this->root = y;
        } else if (x == x->parent->right) {
            x->parent->right = y;
        } else {
            x->parent->left = y;
        }
        y->right = x;
        x->parent = y;
    }

    // Inserting a node
    void insert(int key) {
        NodePtr node = new Node;
        node->parent = nullptr;
        node->left = TNULL;
        node->right = TNULL;
        node->color = 1;
        node->data = key;

        NodePtr y = nullptr;
        NodePtr x = this->root;

        while (x != TNULL) {
            y = x;
            if (node->data < x->data) {
                x = x->left;
            } else {
                x = x->right;
            }
        }

        node->parent = y;
        if (y == nullptr) {
            root = node;
        } else if (node->data < y->data) {
            y->left = node;
        } else {
            y->right = node;
        }

        if (node->parent == nullptr) {
            node->color = 0;
            return;
        }

        if (node->parent->parent == nullptr) {
            return;
        }

        insertFix(node);
    }

    NodePtr getRoot() {
        return this->root;
    }

    void printTree() {
        if (root) {
            printHelper(this->root, "", true);
        }
    }
};

int main() {
    RedBlackTree bst;
    int n;
    cout << "Enter Number of Nodes: ";
    cin >> n;

    while (n--) {
        int k;
        cin >> k;
        bst.insert(k);
    }

    bst.printTree();
}
```

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)
```