```
enum color {RED, BLACK};
struct Node
 {
    int data
    bool color
    Node *left, *right, *parent;
    Node (int data)
      {
       this -> data = data;
       left = right = parent = NULL;
       this -> color = RED;
      }
};
class RBTree
 {
   private:
      Node *root;
   protected:
      void leftrotate (Node *&, Node *&);
      void rightrotate (Node *&, Node *&);
      void fixViolation (Node *&, Node *&);
   Public:
      void insert (const int &n)
      void inorder ();
      void levelOrder ();
};

Node* RBTree::insert(const int &data)
{
   Node *P = new Node(data);
```

```
root=BSTInsert(root, p)
fixviolation(root, p);
}
Node* BSTInsert (Node *root, Node *p)
{
  if root==NULL
    return p;
  if (p->data < root->data)
  {
    root->left = BSTInsert(root->left, p);
    root->left->parent = root;
  }
  else if (p->data > root->data)
  {
    root->right = BSTInsert(root->left, p);
    root->left->parent = root;
  }
return root;
}
void RBTree::fixviolation (Node *&root, Node *&p)
{
  Node *parent_p = NULL;
  Node *grand_p = NULL;
  while(p!=root && p->color!=Black && p->parent->color==Red)
  {
    parent_p= p->parent;
    grand_p = p->parent->parent;
    if (parent_p==grand_p->left)
    {
      Node *uncle_p = grand_p->right;
```

```
if (uncle_p != Null && uncle_p->clor == red)
  {
    grand_p->clor = Red;
    parent_p->color = Black;
    uncle_p->color = Black;
    p = grand_p;
  }
else
  {
    if (p == parent_p->right)
      {
        leftrotate (root, parent_p);
        p = parent_p;
        parent_p = p->parent;
      }
    rightrotate (root, grand_p);
    Swap (parent_p->color, grand_p->color);
    p = parent_p;
  }
}
else
{
Node * uncle_p = grand_p->left;
if (uncle_p != NULL && uncle_p->color == red)
  {
    grand_p->color = red;
    parent_p->color = Black;
    uncle_p->color = Black;
    p = grand_p;
  }
```

```
else
{
    if (p=parent_p->left)
    {
        rightrotate (root, parent_p);
        p=parent_p;
        parent_p = p->parent;
    }
    leftrotate (root, grandp)
    Swap (parent_p->color, grand_p->color);
    p=parent_p;
  }
 }
}
root->color = Black;
}

void leftrotate (Node *&root, Node *&pt)
{ Node *p_right = p->right;
  p->right = p_right->left
  if (p->right != NULL) p->right->parent = p;
  p_right->parent = p->parent;
  if (p->parent == NULL) root = p_right;
  else if (p == p->parent->left) p->parent->left = p_Right;
  else p->parent->right = p_right;
  p_right->left = p; p->parent = p_right;
}
void rightrotate (Node *&root, Node *&pt)
{
  Node *p_left = p->left;
```

```
p->left = p->left->right;
if (p->left != NULL)
    p->left->parent = p;
p->left->parent = p->parent;
if (p->parent == NULL)
    root = p->left;
else if (p == p->parent->left)
    p->parent->left = p->left;
else
    p->parent->right = p->left;
p->left->right = p;
p->parent = p->left;
}
```