# Program - 07

Program for Red-black Tree Insertion:-

```
Node* BSTInsert(Node* root, Node *pt)
{
    if(root == NULL)
        return pt;

    if (pt->data < root->data)
    {
        root->left = BSTInsert(root->left, pt);
        root->left->parent = root;
    }
    else if (pt->data > root->data)
    {
        root->right = BSTInsert(root->right, pt);
        root->right->parent = root;
    }
    return root;
}
```

```
Void levelOrder(Node *root)
{
  if(root == NULL)
    return;

  std::queue<Node *> q;
  q.push(root);

  while(!q.empty())
  {
    Node *temp = q.front();
    cout << temp->data << " ";
    q.pop();

    if(temp->left != NULL)
      q.push(temp->left);

    if(temp->right != NULL)
      q.push(temp->right);
  }
}

Void RBTree::rotateLeft(Node *&root, Node *&pt)
{
  Node *pt_right = pt->right;

  pt->right = pt->right->left;

  if(pt->right != NULL)
    pt->right->parent = pt;

  pt->right->parent = pt->parent;
```

```
if(pt->parent==NULL)
    root=pt_right;

else if(pt==pt->parent->left)
    pt->parent->left=pt_right;

else
    pt->parent->right=pt_right;
pt->pt_right->left=pt;
    pt->parent=pt_right;
}

Void RBTree::rotateRight(Node *&root, Node *&pt)
{
Node *pt_left=pt->left;

 pt->left=pt_left->right;

 if(pt->left!=NULL)
    pt->left->parent=pt;

pt_left->parent=pt->parent;

if(pt->parent==NULL)
    root=pt_left;

else if(pt==pt->parent->left)
      pt->parent->left=pt_left;

else
    pt->parent->right=pt_left;
 pt_left->right=pt;
pt->parent=pt_left;
```

```
void RBTree::fixViolation(Node *&root, Node *&pt)
{
    Node *parent_pt = NULL;
    Node *grand_parent_pt = NULL;

    while ((pt != root) && (pt->color != BLACK) &&
           (pt->parent->color == RED))
    {
        parent_pt = pt->parent;
        grand_parent_pt = pt->parent->parent;

        if(parent_pt == grand_parent_pt->left)
        {
            Node *uncle_pt = grand_parent_pt->right;
            if(uncle_pt != NULL && uncle_pt->color == RED)
            {
                grand_parent_pt->color = RED;
                parent_pt->color = BLACK;
                uncle_pt->color = BLACK;
                pt = grand_parent_pt;
            }
            else
            {
                if(pt == parent_pt->right)
                {
                    rotateLeft(root, parent_pt);
                    pt = parent_pt;
                    parent_pt = pt->parent;
                }
                rotateRight(root, grand_parent_pt);
                swap(parent_pt->color, grand_parent_pt->color);
                pt = parent_pt;
            }
        }
    }
```

```
else
{
    Node *uncle_pt = grand_parent_pt->left;
    if((uncle_pt != NULL) && (uncle_pt->color == RED))
    {
        grand_parent_pt->color = RED;
        parent_pt->color = BLACK;
        uncle_pt->color = BLACK;
        pt = grand_parent_pt;
    }
    else
    {
        if(pt == parent_pt->left)
        {
            rotateRight(root, parent_pt);
            pt = parent_pt;
            parent_pt = pt->parent;
        }
        rotateLeft(root, grand_parent_pt);
        swap(parent_pt->color, grand_parent_pt->color);
        pt = parent_pt;
    }
  }
}
    root->color = BLACK;
}

void RBTree::insert(const int &data)
{
  Node *pt = New Node(data);
  root = BSTInsert(root, pt);
  fixViolation(root, pt);
}
```