Prog-7 : A prog to implement insertion oper^n on a Red-Black tree. During insertion, appropriately show how recolouring (or) rotation oper^n is used.

```
using name space std;
enum Colour {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;
    }
 };

/* utility fun to insert a new node with given key in BST */
 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;
 }
```

// The fun fixes violations caused by BST insertion

```
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 -> colour = 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 ;
```

REDMI NOTE 5 PRO

```
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-p
                      -> 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);
  }
```