Pseudocode for Insertion into a Red-Black Tree

enum COLOR { rbRED, rbBLACK};

struct RBNode
{
    RBNode *left;
    RBNode *right;
    RBNode *parent;
    COLOR color;
    void *item;
};

RBInsert (Root node of RBTree, Item to insert)
{
    RBNode *X = node, *Y;
    Insert new node into RBTree in usual manner (it's a BST)
    Set new node's color to RED and call it X

    // The Red/Black property may have been destroyed
    // so we have to restore it
    while (X is not the Root) and (X's Parent is RED)
    {
        if (X's Parent is a LEFT child)
        {
            // If X's Parent is a LEFT, Uncle must be a RIGHT
            Set Y to X's Uncle (Uncle is Grandparent's RIGHT child)
            if (Y is RED)
            {
                // Case #1, Change the tri-node's colors
                Set X's Parent's color to BLACK
                Set Y's color to BLACK
                Set X's Grandparent's color to RED
                Set X to Grandparent
            }
            else // Y is BLACK
            {
                // and X is RIGHT, Case #2
                // move X up and rotate it (rotates child into Parent's spot)
                if (X is a RIGHT child)
                {
                    Set X to Parent
                    Rotate LEFT about new X
                }
                // Case #3
                Set X's Parent's color to BLACK
                Set X's Grandparent's color to RED
                Rotate RIGHT about X's Grandparent;
            }
        }
        else // X's Parent is a RIGHT child (symmetrical to above code)
        {
            // If X's Parent is a RIGHT, Uncle must be a LEFT
            Set Y to X's Uncle (Uncle is Grandparent's LEFT child)
            if (Y is RED)
            {
                // Case #1, Change the tri-node's colors
                Set X's Parent's color to BLACK
                Set Y's color to BLACK
                Set X's Grandparent's color to RED
                Set X to Grandparent
            }
            else
            {
                // and X is LEFT, Case #2
                // move X up and rotate it (rotates child into Parent's spot)
                if (X is a LEFT child)
                {
                    Set X to Parent
                    Rotate RIGHT about new X
                }
                // Case #3
                Set X's Parent's color to BLACK
                Set X's Grandparent's color to RED
                Rotate LEFT about X's Grandparent;
            }
        }
    }
}