```
#include<stdio.h>
#include<stdlib.h>

struct rbNode
{
    int data;
    int color;
    struct rbNode *link[2];
};

typedef struct rbNode rbNode;

enum nodeColor
{
    RED,
    BLACK
};

rbNode *createNode(int data)
{
    rbNode *newnode;
    newnode = (rbNode *)malloc(sizeof(rbNode));
    newnode -> data = data;
    newnode -> color = RED;
    newnode -> link[0] = newnode -> link[1] = NULL;
    return newnode;
}


rbNode *Insert(rbNode *root, int data)
{
    rbNode *stack[98], *ptr, *newnode, *xPtr, *yPtr;
    int dir[98], ht = 0, index;
    ptr = root;
    if (!root)
    {
        root = createNode(data);
        return root;
    }

    stack[ht] = root;
    dir[ht++] = 0;
    while (ptr != NULL)
    {
        if(ptr -> data == data)
        {
            printf("\nSorry , duplicates not allowed...\n");
            return root;
        }
        index = (data - ptr -> data) > 0 ? 1 : 0;
```

```
    stack[ht] = ptr;
    ptr = ptr -> link[index];
    dir[ht++] = index;
}
stack[ht - 1] -> link[index] = newnode = createNode(data);
while((ht >= 3) && (stack[ht - 1] -> color == RED))
{
    if(dir[ht - 2] == 0)
    {
        yPtr = stack[ht - 2] -> link[1];
        if(yPtr != NULL && yPtr -> color == RED)
        {
            stack[ht - 2] -> color = RED;
            stack[ht - 1] -> color = yPtr -> color = BLACK;
            ht = ht - 2;
        }
        else
        {
            if(dir[ht - 1] == 0)
            {
                yPtr = stack[ht - 1];
            }
            else
            {
                xPtr = stack[ht - 1];
                yPtr = xPtr -> link[1];
                xPtr -> link[1] = yPtr -> link[0];
                yPtr -> link[0] = xPtr;
                stack[ht - 2] -> link[0] = yPtr;
            }
            xPtr = stack[ht - 2];
            xPtr -> color = RED;
            yPtr -> color = BLACK;
            xPtr -> link[0] = yPtr -> link[1];
            yPtr -> link[1] = xPtr;
            if(xPtr == root)
            {
                root = yPtr;
            }
            else
            {
                stack[ht - 3] -> link[dir[ht - 3]] = yPtr;
            }
            break;
        }
    }
    else
    {
        yPtr = stack[ht - 2] -> link[0];
        if((yPtr != NULL) && (yPtr -> color == RED))
```

```
            {
                stack[ht - 2] -> color = RED;
                stack[ht - 1] -> color = yPtr -> color = BLACK;
                ht = ht - 2;
            }
            else
            {
                if(dir[ht - 1] == 1)
                {
                    yPtr = stack[ht - 1];
                }
                else
                {
                    xPtr = stack[ht - 1];
                    yPtr = xPtr -> link[0];
                    xPtr -> link[0] = yPtr -> link[1];
                    yPtr -> link[1] = xPtr;
                    stack[ht - 2] -> link[1] = yPtr;
                }
                xPtr = stack[ht - 2];
                yPtr -> color = BLACK;
                xPtr -> color = RED;
                xPtr -> link[1] = yPtr -> link[0];
                yPtr -> link[0] = xPtr;
                if(xPtr == root)
                {
                    root = yPtr;
                }
                else
                {
                    stack[ht - 3] -> link[dir[ht - 3]] = yPtr;
                }
                break;
            }
        }
    }
    root -> color = BLACK;

    return root;
}

rbNode *Create(rbNode *root )
{
    int num, i, ele;
    printf("\n Enter number of nodes:");
    scanf("%d", &num );
    printf("\n Enter elements:");
    for( i = 0; i < num; i++)
    {
        scanf("%d", &ele);
```

```
        root = Insert( root, ele);
    }

    return root;
}

rbNode  *Delete(rbNode *root, int data)
{
    rbNode *stack[98], *ptr, *xPtr, *yPtr;
    rbNode *pPtr, *qPtr, *rPtr;
    int dir[98], ht = 0, diff, i;
    enum nodeColor color;

    if(!root)
    {
        printf("\nEmpty tree\n");
        return root;
    }

    ptr = root;
    while(ptr != NULL)
    {
        if((data - ptr -> data) == 0)
            break;
        diff = (data - ptr -> data) > 0 ? 1 : 0;
        stack[ht] = ptr;
        dir[ht++] = diff;
        ptr = ptr -> link[diff];
    }

    if(ptr -> link[1] == NULL)
    {
        if((ptr == root) && (ptr -> link[0] == NULL))
        {
            free(ptr);
            root = NULL;
        }
        else if(ptr == root)
        {
            root = ptr -> link[0];
            free(ptr);
        }
        else
        {
            stack[ht - 1] -> link[dir[ht - 1]] = ptr -> link[0];
        }
    }
    else
    {
        xPtr = ptr -> link[1];
```

```
if(xPtr -> link[0] == NULL)
{
    xPtr -> link[0] = ptr -> link[0];
    color = xPtr -> color;
    xPtr -> color = ptr -> color;
    ptr -> color = color;

    if(ptr == root)
    {
        root = xPtr;
    }
    else
    {
        stack[ht - 1] -> link[dir[ht - 1]] = xPtr;
    }

    dir[ht] = 1;
    stack[ht++] = xPtr;
}
else
{
    i = ht++;
    while(1)
    {
        dir[ht] = 0;
        stack[ht++] = xPtr;
        yPtr = xPtr -> link[0];
        if(!yPtr -> link[0])
            break;
        xPtr = yPtr;
    }

    dir[i] = 1;
    stack[i] = yPtr;
    if(i > 0)
        stack[i - 1] -> link[dir[i - 1]] = yPtr;
    yPtr -> link[0] = ptr -> link[0];

    xPtr -> link[0] = yPtr -> link[1];
    yPtr -> link[1] = ptr -> link[1];

    if(ptr == root)
    {
        root = yPtr;
    }

    color = yPtr -> color;
    yPtr -> color = ptr -> color;
    ptr -> color = color;
}
```

```
    }

    if(ht < 1)
        return root;

    if(ptr -> color == BLACK)
    {
        while(1)
        {
            pPtr = stack[ht - 1] -> link[dir[ht - 1]];
            if(pPtr && pPtr -> color == RED)
            {
                pPtr -> color = BLACK;
                break;
            }

            if(ht < 2)
                break;

            if(dir[ht - 2] == 0)
            {
                rPtr = stack[ht - 1] -> link[1];

                if(!rPtr)
                    break;
                if(rPtr -> color == RED)
                {
                    stack[ht - 1] -> color = RED;
                    rPtr -> color = BLACK;
                    stack[ht - 1] -> link[1] = rPtr -> link[0];
                    rPtr -> link[0] = stack[ht - 1];

                    if(stack[ht - 1] == root)
                    {
                        root = rPtr;
                    }
                    else
                    {
                        stack[ht - 2] -> link[dir[ht - 2]] = rPtr;
                    }

                    dir[ht] = 0;
                    stack[ht] = stack[ht - 1];
                    stack[ht - 1] = rPtr;
                    ht++;
                    rPtr = stack[ht - 1] -> link[1];
                }

                if((!rPtr -> link[0] || rPtr -> link[0] -> color == BLACK) &&
(!rPtr -> link[1] || rPtr -> link[1] -> color == BLACK))
```

```
    {
        rPtr -> color = RED;
    }
    else
    {
        if(!rPtr -> link[1] || rPtr -> link[1] -> color == BLACK)
        {
            qPtr = rPtr -> link[0];
            rPtr -> color = RED;
            qPtr -> color = BLACK;
            rPtr -> link[0] = qPtr -> link[1];
            qPtr -> link[1] = rPtr;
            rPtr = stack[ht - 1] -> link[1] = qPtr;
        }

        rPtr -> color = stack[ht - 1] -> color;
        stack[ht - 1] -> color = BLACK;
        rPtr -> link[1] -> color = BLACK;
        stack[ht - 1] -> link[1] = rPtr -> link[0];
        rPtr -> link[0] = stack[ht - 1];
        if(stack[ht - 1] == root)
        {
            root = rPtr;
        }
        else
        {
            stack[ht - 2] -> link[dir[ht - 2]] = rPtr;
        }
        break;
    }
}
else
{
    rPtr = stack[ht - 1] -> link[0];

    if(!rPtr)
        break;
    if(rPtr->color == RED)
    {
        stack[ht - 1]->color = RED;
        rPtr -> color = BLACK;
        stack[ht - 1] -> link[0] = rPtr -> link[1];
        rPtr -> link[1] = stack[ht - 1];

        if(stack[ht - 1] == root)
        {
            root = rPtr;
        }
        else
        {
```

```
                        stack[ht - 2] -> link[dir[ht - 2]] = rPtr;
                    }

                    dir[ht] = 1;
                    stack[ht] = stack[ht - 1];
                    stack[ht - 1] = rPtr;
                    ht++;

                    rPtr = stack[ht - 1] -> link[0];
                }

                if((!rPtr -> link[0] || rPtr -> link[0]->color == BLACK) && (!rPtr
-> link[1] || rPtr -> link[1]->color == BLACK))
                {
                    rPtr -> color = RED;
                }
                else
                {
                    if(!rPtr -> link[0] || rPtr -> link[0] -> color == BLACK)
                    {
                        qPtr = rPtr -> link[1];
                        rPtr -> color = RED;
                        qPtr -> color = BLACK;
                        rPtr -> link[1] = qPtr -> link[0];
                        qPtr -> link[0] = rPtr;
                        rPtr = stack[ht - 1] -> link[0] = qPtr;
                    }

                    rPtr -> color = stack[ht - 1] -> color;
                    stack[ht - 1] -> color = BLACK;
                    rPtr -> link[0] -> color = BLACK;
                    stack[ht - 1] -> link[0] = rPtr -> link[1];
                    rPtr -> link[1] = stack[ht - 1];
                    if(stack[ht - 1] == root)
                    {
                        root = rPtr;
                    }
                    else
                    {
                        stack[ht - 2]->link[dir[ht - 2]] = rPtr;
                    }
                    break;
                }
            }
            ht--;
        }
    }
    return root;
}
```

```
void Inorder(rbNode *root)
{
  if (root != NULL)
  {
      Inorder( root -> link[0]);
      printf("%d -> ", root -> data );
      Inorder( root -> link[1] );
  }
}

int main()
{

        rbNode *root = NULL;
        int ele;
        int e = 1, ch;

        while( e )
        {
                printf( "\n--------------MENU--------------\n" );
                printf( "\n\t1. Create\n\t2. Insert\n\t3. Inorder Traversal\n\t4.
Delete\n\t5. Exit\n" );
                printf( "\n--------------------------------\n" );
                printf( "\n Enter your choice:" );
                scanf( "%d", &ch );

                switch( ch )
                {
                        case 1: root = Create( root );
                             break;
                        case 2:
                                printf("\n Enter the element to insert:");
                                scanf("%d", &ele);
                                root = Insert( root, ele);
                                 break;
                        case 3: Inorder( root );
                                 break;
                        case 4:
                                printf("\n Enter the element to delete :");
                                scanf("%d", &ele);
                                root = Delete( root, ele );
                                 break;
                    case 5: e = 0;
                                 break;
                        default: printf( "\n Invalid choice \n" );
                }

        }
        return 0;
```

```
}
```