```
// rbtree.c
//Program to implement Red Black tree
#include <stdio.h>
#include <stdlib.h>
enum nodeColor {
      RED,
      BLACK
};

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

struct rbNode *root = NULL;

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

void preorder(struct rbNode *T)
{
    if(T!=NULL)
    {
        printf("%d(color is %d) ",T->data, T->color);
        preorder(T->link[0]);
        preorder(T->link[1]);
    }
}


  void insertion (int data) {
      struct rbNode *stack[98], *ptr, *newnode, *xPtr, *yPtr;
      int dir[98], ht = 0, index;
      ptr = root;
      if (!root) {
            root = createNode(data);
            return;
      }
      stack[ht] = root;
      dir[ht++] = 0;
      /* find the place to insert the new node */
      while (ptr != NULL) {
            if (ptr->data == data) {
```

```
                printf("Duplicates Not Allowed!!\n");
                return;
        }
        index = (data - ptr->data) > 0 ? 1 : 0;
        stack[ht] = ptr;
        ptr = ptr->link[index];
        dir[ht++] = index;
}
/* insert the new node */
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) {
                        /*
                         * Red node having red child. B- black, R-red
                         *     B               R
                         *    / \             /   \
                         *   R   R  =>      B     B
                         *  /               /
                         * R               R
                         */
                        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 {
                                /*
                                 * XR - node X with red color
                                 * YR - node Y with red color
                                 * Red node having red child
                                 *(do single rotation left b/w X and Y)
                                 *         B              B
                                 *        /              /
                                 *      XR     =>      YR
                                 *        \            /
                                 *         YR         XR
                                 * one more additional processing will be
                                 * performed after this else part.  Since
                                 * we have red node (YR) with red child(XR)
                                 */
                                xPtr = stack[ht - 1];
                                yPtr = xPtr->link[1];
                                xPtr->link[1] = yPtr->link[0];
                                yPtr->link[0] = xPtr;
                                stack[ht - 2]->link[0] = yPtr;
```

```
                        }
                        /*
                         *  Red node(YR) with red child (XR) - single
                         *  rotation b/w YR and XR for height balance.
```

Still,

```
                         *  red node (YR) is having red child.  So, change
```

the

```
                         *  color of Y to black and Black child B to Red R
                         *           B           YR          YB
                         *          /           /  \        /  \
                         *        YR  =>   XR   B  =>  XR  R
                         *       /
                         *      XR
                         */
                        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)) {
                        /*
                         * Red node with red child
                         *        B              R
                         *      /   \          /   \
                         *     R     R =>  B     B
                         *              \              \
                         *               R              R
                         *
                         */
                        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 {
                                /*
                                 * Red node(XR) with red child(YR)
                                 *   B          B
```

```
                                        rbtree.c
                                           *    \           \
                                           *     XR  => YR
                                           *    /             \
                                           *   YR              XR
                                           * Single rotation b/w XR(node x with red
color) & YR
                                           */
                                          xPtr = stack[ht - 1];
                                          yPtr = xPtr->link[0];
                                          xPtr->link[0] = yPtr->link[1];
                                          yPtr->link[1] = xPtr;
                                          stack[ht - 2]->link[1] = yPtr;
                                    }
                                    /*
                                     *   B              YR          YB
                                     *    \             /  \        /  \
                                     *     YR  =>   B   XR => R    XR
                                     *      \
                                     *       XR
                                     * Single rotation b/w YR and XR and change the
color to
                                     * satisfy rebalance property.
                                     */
                                    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;
        }

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

               if (!root) {
                      printf("Tree not available\n");
```

```
        return;
}

ptr = root;
/* search the node to delete */
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) {
        /* node with no children */
        if ((ptr == root) && (ptr->link[0] == NULL)) {
                free(ptr);
                root = NULL;
        } else if (ptr == root) {
                /* deleting root - root with one child */
                root = ptr->link[0];
                free(ptr);
        } else {
                /* node with one child */
                stack[ht - 1]->link[dir[ht - 1]] = ptr->link[0];
        }
} else {
        xPtr = ptr->link[1];
        if (xPtr->link[0] == NULL) {
                /*
                 * node with 2 children - deleting node
                 * whose right child has no left child
                 */
                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 {
```

```
                /* deleting node with 2 children */
                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;
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) {
        /*
         * incase if rPtr is red, we need
         * change it to black..
         *    aB                rPtr (red)
rPtr(black)
         *   /  \      =>      /    \  =>    /   \
         *  ST  rPtr(red)  aB    cB      aR   cB
         *       /  \       /  \           /  \
         *     bB  cB   ST  bB       ST  bB
         *  ST - subtree
         *  xB - node x with Black color
         *  xR - node x with Red color
         * the above operation will simply rebalace
         * operation in RB tree
         */
        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(black)          rPtr(Red)
         *     /    \          =>  /    \
         *    B      B            R      R
         *
         */
        rPtr->color = RED;
} else {
```

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

         /*
          * Below is a subtree. rPtr with red
left child
          * single rotation right b/w yR and
rPtr  &
          * change the color as needed
          *        wR
wR
          *       /  \                    /
 \
          *      xB   rPtr(Black) =>   xB  yB
          *     / \   /  \               /  \
 /  \
          *    a   b yR   e            a   b c
  rPtr(Red)
          *         /  \
     /  \
          *        c    d
    d    e
          */
         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;

 }
 /*
  * Below is a subtree. rPtr with Right red
child
  * single rotation b/w rPtr & wR and change
colors
  *       wR (stack[ht-1])      rPtr(Red)
  *     /   \                   /    \
  *    xB    rPtr(black)     wB     yB
  *   / \   /  \        =>    /   \   /  \
  *  a   b c    yR         xB    c d    e
  *             /  \       /  \
  *            d    e     a    b
  */
 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--;
    }
  }
}

void searchElement(int data) {
    struct rbNode *temp = root;
    int diff;

    while (temp != NULL) {
        diff = data - temp->data;
        if (diff > 0) {
            temp = temp->link[1];
        } else if (diff < 0) {
            temp = temp->link[0];
        } else {
            printf("Search Element Found!!\n");
            return;
        }
    }
    printf("Given Data Not Found in RB Tree!!\n");
    return;
}

void inorderTraversal(struct rbNode *node) {
    if (node) {
        inorderTraversal(node->link[0]);
        printf("%d  ", node->data);
        inorderTraversal(node->link[1]);
    }
    return;
}

int main() {
```

```
	int ch, data;
	while (1) {
		printf("1. Insertion\t2. Deletion\n");
		printf("3. Searching\t4. Inorder Traverse\n");
		printf("5. Exit\nEnter your choice:");
		scanf("%d", &ch);
		switch (ch) {
			case 1:
				printf("Enter the data to insert:");
				scanf("%d", &data);
				insertion(data);
				break;
			case 2:
				printf("Enter the data to delete:");
				scanf("%d", &data);
				deletion(data);
				break;
			case 3:
				printf("Enter the search element:");
				scanf("%d", &data);
				searchElement(data);
				break;
			case 4:
				printf("Inorder traversal of the tree\n");
				inorderTraversal(root);
				printf("\nPreorder traversal of the tree\n");
				preorder(root);
				printf("\n");
				break;
			case 5:
				exit(0);
			default:
				printf("You have entered wrong option!!\n");
				break;
		}
		printf("\n");
	}
	return 0;
}
```