```
#include<stdio.h>
#include<conio.h>
struct rbtNode

{
    int data;
    char color;
    struct rbtNode *left, *right, *parent;
};
struct rbtNode *root=NULL;
void leftRotate(struct rbtNode *x)
{
    struct rbtNode *y;
    y= x->right;
    x->right = y->left;


    if (y->left != NULL)
         y->left->parent = x;

    y->parent = x->parent;
    if (x->parent == NULL)// x is root
         root = y;

    else if((x->parent->left != NULL) && (x->data==x->parent->left-
>data))
         x->parent->left = y;

    else
         x->parent->right = y;

    y->left = x;
    x->parent = y;
    return;
}
void rightRotate(struct rbtNode *y)

{

    struct rbtNode *x   ;
    x= y->left;

    y->left = x->right;

    if (x->right != NULL)

     x->right->parent = y;

    x->parent =y->parent;

    if (y->parent == NULL)
     root= x;
```

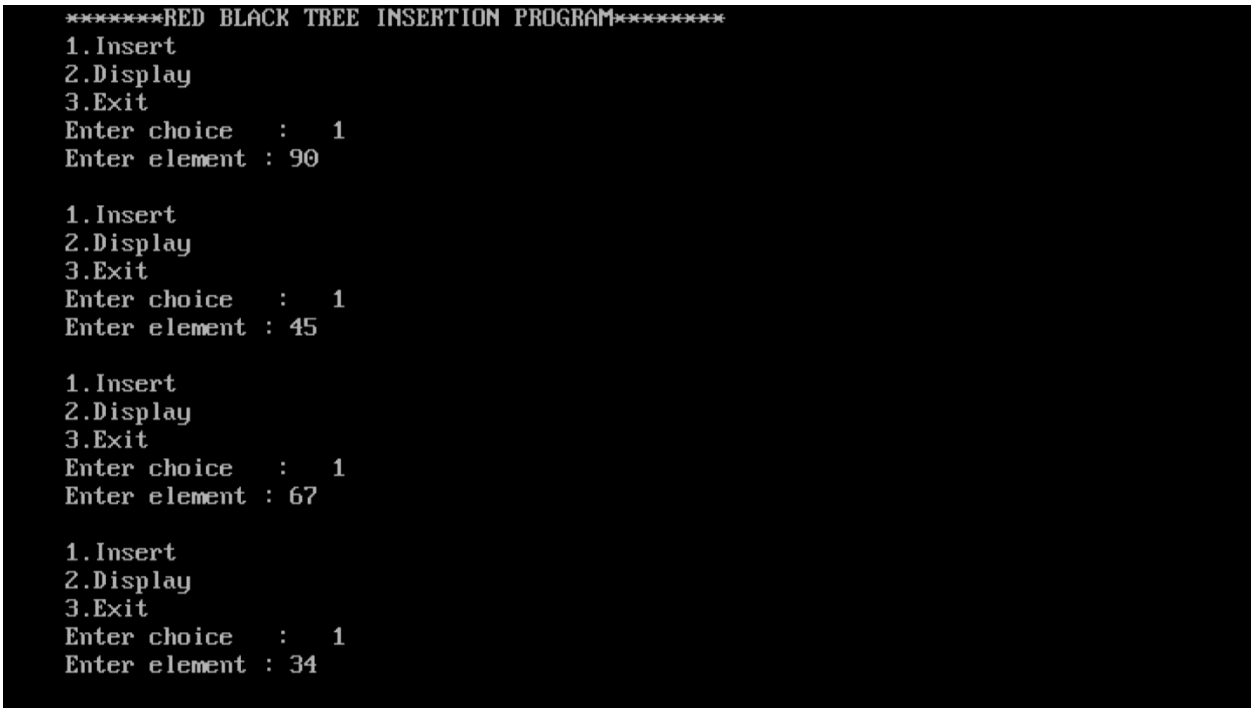
```
    else if((y->parent->left!=NULL)&&(y->data==y->parent->left->data))

      y->parent->left = x;

    else
      y->parent->right = x;

    x->right = y;

    y->parent = x;
 return;
}
void color(struct rbtNode *z)
{
    struct rbtNode *y=NULL;
    while((z->parent!=NULL)&&(z->parent->color=='R'))
    {
      if((z->parent->parent->left!=NULL)&&(z->parent->data==z->parent-
>parent->left->data))
      {
          if(z->parent->parent->right!=NULL)
            y=z->parent->parent->right;
          if((y!=NULL)&&(y->color=='R'))
          {
            z->parent->color='B';
            y->color='B';
            z->parent->parent->color = 'R';
            if(z->parent->parent!=NULL)
                z = z->parent->parent;

          }
          else
          {
            if((z->parent->right!=NULL)&&(z->data==z->parent->right-
>data))
            {
                z=z->parent;
                leftRotate(z);
            }
            z->parent->color='B';
                  z->parent->parent->color='R';
            rightRotate(z->parent->parent) ;
          }
      }
      else
      {
            if(z->parent->parent->left!=NULL)
            y=z->parent->parent->left;
          if((y!=NULL)&&(y->color=='R'))
          {
            z->parent->color='B';
```

```
            y->color='B';
            z->parent->parent->color='R';
            if(z->parent->parent!=NULL)
                z=z->parent->parent;
          }
          else
          {
                 if((z->parent->left!=NULL)&&(z->data==z->parent->left-
>data))
            {
                z=z->parent;
                rightRotate(z);
            }
            z->parent->color='B';
            z->parent->parent->color='R';
            leftRotate(z->parent->parent);
          }

      }
    }
    root->color='B';
}
void insert(int val)
{
    struct rbtNode *x,*y,*z;
    z=(struct rbtNode *)malloc(sizeof(struct rbtNode));
    z->data=val;
    z->left=NULL;
    z->right=NULL;
    z->color='R';

    x=root;

    if(root==NULL)// root node or not
    {
      root=z;
      root->color='B';
      return;
    }

    while(x!=NULL)
    {
      y=x;
      if(z->data<x->data)
          x=x->left;
      else
          x=x->right;
    }
    z->parent=y;
    if(y==NULL)
          root=z;
    else if(z->data<y->data)
```

```
          y->left=z;
     else
          y->right=z;

     color(z);

}
void inorder(struct rbtNode *root)

{
     struct rbtNode *temp=root;
     if (temp != NULL)
     {
       inorder(temp->left);

          printf("%d-%c   ", temp->data,temp->color);

       inorder(temp->right);
     }
     return;
}
void displayTree(struct rbtNnode *root,int level)
{
   int i;
   struct rbtNode *temp=root;

   if(temp!=NULL)
   {
     displayTree(temp->right,level+1);
     printf("\n\n");
     for(i=0;i<level;i++)
           printf("    ");
     printf("%d%c",temp->data,temp->color);
     displayTree(temp->left,level+1);
   }
}
void main()
{
     int ch,val;
     clrscr();
     printf("*******RED BLACK TREE INSERTION PROGRAM********")  ;
     while(1)
     {
       printf("\n1.Insert\n2.Display\n3.Exit\nEnter choice   :   ");
       scanf("%d",&ch);
       switch(ch)
       {
           case 1 :
           printf("Enter element : ");
           scanf("%d",&val);
           insert(val);
           break;
```

```
            case 2:
            displayTree(root,1);
            printf("\n\nINORDER TRAVERSAL :     ");
            inorder(root);break;
            case 3:exit(0);
        }
      }

}
```

OUTPUT



```
1.Insert
2.Display
3.Exit
Enter choice   :   1
Enter element : 34

1.Insert
2.Display
3.Exit
Enter choice   :   2


        90B

    67B

        45B

            34R

INORDER TRAVERSAL :     34-R   45-B   67-B   90-B
1.Insert
2.Display
3.Exit
Enter choice   :
```