NAME : JAGADEESH R
REGNO : 2021506314
PROGRAM : AVL TREE -> Insertion, Deletion, Find, Display
ADS Lab 07

SOURCE CODE :

```
#include<iostream>
#include<stdlib.h>
using namespace std;
#define TRUE 1
#define FALSE 0
class AVL;
class AVLNODE
{
   friend class AVL;
private:
   int data;
   AVLNODE *left,*right;
   int bf;
};
class AVL
{
private:
   AVLNODE *root;
public:
   AVLNODE *loc,*par;
   AVL()
   {
      root=NULL;
   }
   int insert(int);
   void displayitem();
   void display(AVLNODE *);
   void removeitem(int);
   void remove1(AVLNODE *,AVLNODE *,int);
   void remove2(AVLNODE *,AVLNODE *,int);
   void search(int x);
   void search1(AVLNODE *,int);
};
int AVL::insert(int x)
{
   AVLNODE *a,*b,*c,*f,*p,*q,*y,*clchild,*crchild;
   int found,unbalanced;
   int d;
   if(!root)
```

```
{
   y=new AVLNODE;
   y->data=x;
   root=y;
   root->bf=0;
   root->left=root->right=NULL;
   return TRUE;
}
f=NULL;
a=p=root;
q=NULL;
found=FALSE;
while(p&&!found)
{
   if(p->bf)
   {
      a=p;
      f=q;
   }
   if(x<p->data)
   {
      q=p;
      p=p->left;
   }
   else if(x>p->data)
   {
      q=p;
      p=p->right;
   }
   else
   {
      y=p;
      found=TRUE;
   }
}
if(!found)
{
   y = new AVLNODE;
   y->data=x;
   y->left=y->right=NULL;
   y->bf=0;
   if(x<q->data)
      q->left=y;
   else
      q->right=y;
   if(x>a->data)
   {
```

```
   p=a->right;
   b=p;
   d=-1;
}
else
{
   p=a->left;
   b=p;
   d=1;
}
while(p!=y)
   if(x>p->data)
   {
      p->bf=-1;
      p=p->right;
   }
   else
   {
      p->bf=1;
      p=p->left;
   }
unbalanced=TRUE;
if(!(a->bf)||!(a->bf+d))
{
   a->bf+=d;
   unbalanced=FALSE;
}
if(unbalanced)
{
   if(d==1)
   {
      if(b->bf==1) //rotation type LL
      {
         a->left=b->right;
         b->right=a;
         a->bf=0;
         b->bf=0;
      }
      else //rotation type LR
      {
         c=b->right;
         b->right=c->left;
         a->left=c->right;
         c->left=b;
         c->right=a;
         switch(c->bf)
         {
```

```
        case 1:
           a->bf=-1; //LR(b)
           b->bf=0;
           break;
        case -1:
           b->bf=1; //LR(c)
           a->bf=0;
           break;
        case 0:
           b->bf=0; //LR(a)
           a->bf=0;
           break;
        }
        c->bf=0;
        b=c;
     } //end of LR
  }
  else
  {
     if(b->bf==-1) //rotation type RR
     {
        a->right=b->left;
        b->left=a;
        a->bf=0;
        b->bf=0;
     }
     else //rotation type LR
     {
        c=b->right;
        b->right=c->left;
        a->right=c->left;
        c->right=b;
        c->left=a;
        switch(c->bf)
        {
        case 1:
           a->bf=-1; //LR(b)
           b->bf=0;
           break;
        case -1:
           b->bf=1; //LR(c)
           a->bf=0;
           break;
        case 0:
           b->bf=0; //LR(a)
           a->bf=0;
           break;
```

```
                }
                c->bf=0;
                b=c;
            } //end of LR
        }
        if(!f)
            root=b;
        else if(a==f->left)
            f->left=b;
        else if(a==f->right)
            f->right=b;
    }
    return TRUE;
  }
  return FALSE;
}

void AVL::displayitem()
{
  display(root);
}
void AVL::display(AVLNODE *temp)
{
  if(temp==NULL)
    return;
  cout<<temp->data<<" ";
  display(temp->left);
  display(temp->right);
}
void AVL::removeitem(int x)
{
  search(x);
  if(loc==NULL)
  {
    cout<<"\nElement not found ";
    return;
  }
  if(loc->right!=NULL&&loc->left!=NULL)
    remove1(loc,par,x);
  else
    remove2(loc,par,x);
}
void AVL::remove1(AVLNODE *l,AVLNODE *p,int x)
{
  AVLNODE *ptr,*save,*suc,*psuc;
  ptr=l->right;
  save=l;
```

```
   while(ptr->left!=NULL)
   {
      save=ptr;
      ptr=ptr->left;
   }
   suc=ptr;
   psuc=save;
   remove2(suc,psuc,x);
   if(p!=NULL)
      if(l==p->left)
         p->left=suc;
      else
         p->right=suc;
   else
      root=l;
   suc->left=l->left;
   suc->right=l->right;
   return;
}
void AVL::remove2(AVLNODE *s,AVLNODE *p,int x)
{
   AVLNODE *child;
   if(s->left==NULL && s->right==NULL)
      child=NULL;
   else if(s->left!=NULL)
      child=s->left;
   else
      child=s->right;
   if(p!=NULL)
      if(s==p->left)
         p->left=child;
      else
         p->right=child;
   else
      root=child;
}
void AVL::search(int x)
{
   search1(root,x);
}
void AVL::search1(AVLNODE *temp,int x)
{
   AVLNODE *ptr,*save;
   int flag;
   if(temp==NULL)
   {
      cout<<"\nThe tree is empty";
```

```
        return;
    }
    if(temp->data==x)
    {
        cout<<"\nThe Element is rootnode and it is found ";
        par=NULL;
        loc=temp;
        par->left=NULL;
        par->right=NULL;
        return;
    }
    if( x < temp->data)
    {
        ptr=temp->left;
        save=temp;
    }
    else
    {
        ptr=temp->right;
        save=temp;
    }
    while(ptr!=NULL)
    {
        if(x==ptr->data)
        {
            flag=1;
            cout<<"\nElement found ";
            loc=ptr;
            par=save;
        }
        if(x<ptr->data)
            ptr=ptr->left;
        else
            ptr=ptr->right;
    }
    if(flag!=1)
    {
        cout<<"Element not found ";
        loc=NULL;
        par=NULL;
        cout<<loc;
        cout<<par;
    }
}
int main()
{
    AVL a;
```

```
    int x,y,c;
    while(1)
    {
       cout<<"\n"<<"\n1.Insert an Element into the tree";
       cout<<"\n2.Display an Elements in the tree";
       cout<<"\n3.Delete an Element in the tree";
       cout<<"\n4.Find an Element in the tree";
       cout<<"\n5.Exit";
       cout<<"\nEnter your choice : ";
       cin>>c;
       switch(c)
       {
       case 1:
          cout<<"\nEnter an Element to be inserted :";
          cin>>x;
          a.insert(x);
          break;
       case 2:
          a.displayitem();
          break;
       case 3:
          cout<<"\nEnter an Element for Deletion :";
          cin>>y;
          a.removeitem(y);
          break;
       case 4:
          cout<<"\nEnter an Element to perform Search :";
          cin>>c;
          a.search(c);
          break;
       case 5:
          exit(0);
          break;
       default :
          cout<<"\nInvalid option";
       }
     }
       return 0;
}
```

OUTPUT :

```
Output
/tmp/SMCEY4ZPsS.o
1.Insert an Element into the tree
2.Display an Elements in the tree
3.Delete an Element in the tree
4.Find an Element in the tree
5.Exit
Enter your choice : 1
Enter an Element to be inserted :2
1.Insert an Element into the tree
2.Display an Elements in the tree
3.Delete an Element in the tree
4.Find an Element in the tree
5.Exit
Enter your choice : 1
Enter an Element to be inserted :5
1.Insert an Element into the tree
2.Display an Elements in the tree
3.Delete an Element in the tree
4.Find an Element in the tree
5.Exit
Enter your choice : 1
Enter an Element to be inserted :9
1.Insert an Element into the tree
2.Display an Elements in the tree
3.Delete an Element in the tree
4.Find an Element in the tree
5.Exit
Enter your choice : 1
Enter an Element to be inserted :7
1.Insert an Element into the tree
2.Display an Elements in the tree
3.Delete an Element in the tree
4.Find an Element in the tree
5.Exit
```

```
Enter your choice : 1
Enter an Element to be inserted :4
1.Insert an Element into the tree
2.Display an Elements in the tree
3.Delete an Element in the tree
4.Find an Element in the tree
5.Exit
Enter your choice : 2
5 2 4 9 7

1.Insert an Element into the tree
2.Display an Elements in the tree
3.Delete an Element in the tree
4.Find an Element in the tree
5.Exit
Enter your choice : 3
Enter an Element for Deletion :4
Element found

1.Insert an Element into the tree
2.Display an Elements in the tree
3.Delete an Element in the tree
4.Find an Element in the tree
5.Exit
Enter your choice : 4
Enter an Element to perform Search :2
Element found

1.Insert an Element into the tree
2.Display an Elements in the tree
3.Delete an Element in the tree
4.Find an Element in the tree
5.Exit
```

```
Enter your choice : 2
5 2 4

1.Insert an Element into the tree
2.Display an Elements in the tree
3.Delete an Element in the tree
4.Find an Element in the tree
5.Exit
Enter your choice :
```