==========

Roll Number: SYCOC303 Division: C

PRN Number: 122B2B303 Batch: C4

Name: VINAYAK MADAN SHETE

==========

Problem Statement:

- **Write a C++ program to perform the following operations on a height balanced tree:**
  - **i) Insert a node**
  - **ii) Search a node**
  - **iii) Display it in ascending order**

==========

INPUT:

```
/*
 *     ====================================
 *            Program Name: AVL.cpp
 *     Created on: December 10, 2022
 *       Author: Vinayak Shete
 *  ====================================
 */


//AVL Tree Implementation
#include <iostream>
using namespace std;

struct Node
{
    int data;
    Node* left;
    Node* right;
};

class AVLTree
```

```
{
public:
    Node* root;
    AVLTree()
    {
        root=NULL;
    }

      //function to find max of two
    int max(int a, int b)
    {
        if(a>b)
        {
            return a;
             }
        else
        {
            return b;
             }
    }

    //function to get the leftmost leaf node
    Node * minValueNode(Node* temp)
      {
             Node* current = temp;
      while (current->left != NULL)
             {
             current=current->left;
      }
      return current;
      }

      //function to calculate the height of the tree
    int height(Node* temp)
    {
        if(temp==NULL)
            return -1;
```

```
    else if(temp->left==NULL && temp->right==NULL)
        return 0;
    return (1+max(height(temp->left), height(temp->right)));
}

   //function for LL rotation
Node* LL(Node* p)
{
    Node* temp;
    temp=p->left;
    p->left=temp->right;
    temp->right=p;
    return temp;
}

   //function for RR rotation
Node* RR(Node* p)
{
    Node* temp;
    temp=p->right;
    p->right=temp->left;
    temp->left=p;
    return temp;
}

   //function for LR rotation
Node* LR(Node* p)
{
    p->left=RR(p->left);
    p=LL(p);
    return p;
}

   //function for RL rotation
Node* RL(Node* p)
{
    p->right=LL(p->right);
```

```
    p=RR(p);
    return p;
}

   //function for inserting a node into tree
Node* insertNode(int key, Node* t)
{
    if(t==NULL)
    {
        Node* ptr=new Node;
        ptr->data=key;
        ptr->left=NULL;
        ptr->right=NULL;
        t=ptr;
        return t;
    }

         //inserting it to left side
    else if(t->data>key)
    {
        t->left=insertNode(key, t->left);
        //calculating and checking the balance factor after inserting the node
        if((height(t->left)- height(t->right))==2)
        {
            if(t->left->data>key)
                t=LL(t);
            else
                t=LR(t);
        }
        return t;
    }

         //inserting it to right side
    else if(t->data<key)
    {
        t->right=insertNode(key, t->right);
        //calculating and checking the balance factor after inserting the node
```

```
        if((height(t->left)- height(t->right))==-2)
        {
            if(t->right->data>key)
                t=RL(t);
            else
                t=RR(t);
        }
        return t;
     }
}

   //function for deleting a Node
   Node* deleteNode(Node* temp, int key)
   {
   if (temp==NULL)
          {
          return NULL;
          }

   //node is present in left sub-tree
   else if (key<temp-> data)
          {
          temp->left = deleteNode(temp->left,key);
   }

          //node is present in right subtree
          else if (key>temp->data)
          {
          temp->right = deleteNode(temp->right,key);
   }
   else
          {
          // node with only one child or no child
          if (temp->left == NULL)
                {
                Node* t=temp->right;
                delete temp;
```

```
                return t;
        }
                else if (temp->right == NULL)
                {
                Node* t=temp->left;
                delete temp;
                return t;
        }
                else
                {
                // node with two children: Get the inorder successor (smallest
                // in the right subtree)
                Node* t= minValueNode(temp->right);
                // Copy the inorder successor's content to this node
                temp->data = t->data;
                // Delete the inorder successor
                temp->right = deleteNode(temp->right,t->data);
                //deleteNode(r->right, temp->value);
        }
}
}

//function for searching an element iteratively
Node* search(int key)
{
if (root == NULL)
        {
        return root;
}
        else
        {
        Node* temp = root;
        while (temp != NULL)
                {
                if (key==temp->data)
                        {
                        return temp;
```

```
                }
                        else if(key<temp->data)
                        {
                        temp = temp -> left;
                }
                        else
                        {
                        temp = temp -> right;
                }
          }
          return NULL;
   }
   }

   //function for printing nodes in an ascending order
void inorder( Node* t)
{
    if(t!=NULL)
    {
        inorder(t->left);
        cout<<t->data<<" ";
        inorder(t->right);
    }
}

//function for printing nodes in preorder
void preorder( Node* t)
{
    if(t!=NULL)
    {
        cout<<t->data<<" ";
        preorder(t->left);
        preorder(t->right);
    }
}

//function for printing nodes in postorder
```

```
    void postorder( Node* t)
    {
        if(t!=NULL)
        {
            postorder(t->left);
            postorder(t->right);
            cout<<t->data<<" ";
        }
    }
};

int main()
{
       int doch,ch,ele;
       Node* found;
    AVLTree a;
    cout<<"\n================WECLOME================";
    do
    {
       cout<<"\n1.Inserting a Node\t\t2.Deleting a Node\n3.Searching a
Node\t\t4.Display in Ascending Order\n5.Display\t\t\t6.Exit";
       cout<<"\nEnter your proper choice:";
       cin>>ch;
       switch(ch)
       {
              case 1:
                     cout<<"\n======INSERTION=========";
                     cout<<"\nEnter the number you want to insert in an AVL Tree:";
                     cin>>ele;
                     a.root=a.insertNode(ele,a.root);
                     break;

              case 2:
                     cout<<"\n======DELETION=========";
                     cout<<"\nThe numbers in AVL tree are:";
                            a.inorder(a.root);
                     cout<<"\nEnter the number you want to delete from an AVL Tree:";
```

```
        cin>>ele;
        a.deleteNode(a.root,ele);
        break;

case 3:
        cout<<"\n======SEARCHING=========";
        cout<<"\nEnter the number you want to search:";
        cin>>ele;
              found=a.search(ele);
        if(found)
        {
              cout<<"\nAn element"<<found->data<<" is found";
              }
              else
              {
                    cout<<"\nAn element"<<found->data<<" is not found";
              }
        break;
case 4:
        cout<<"\n======ASCENDING ORDER=========";
        cout<<"\nThe elements in Ascending Order are: ";
        a.inorder(a.root);
        break;

case 5:
        cout<<"\n======ALL TRAVERSALS=========";
        cout<<"\nInorder Traversal of the Tree is: ";
        a.inorder(a.root);
              cout<<"\nPreorder Traversal of the Tree is: ";
        a.preorder(a.root);
        cout<<"\nPostorder Traversal of the Tree is: ";
        a.postorder(a.root);
        break;

case 6:
        goto exit;
        break;
```

```
            default:
                  cout<<"\nPlease enter correct choice!";
            }
      cout<<"\n=======\nDo you want to continue?[1 for YES || 0 for No]-->";
      cin>>doch;
      }while(doch==1);
      exit:
            cout<<"\n================THANK YOU!================";
   return 0;
}
```

==========

OUTPUT:

```
================WECLOME================
1.Inserting a Node              2.Deleting a Node
3.Searching a Node              4.Display in Ascending Order
5.Display                       6.Exit
Enter your proper choice:1

======INSERTION=========
Enter the number you want to insert in an AVL Tree:50

=======
Do you want to continue?[1 for YES || 0 for No]-->1

1.Inserting a Node              2.Deleting a Node
3.Searching a Node              4.Display in Ascending Order
5.Display                       6.Exit
Enter your proper choice:1

======INSERTION=========
Enter the number you want to insert in an AVL Tree:60

=======
Do you want to continue?[1 for YES || 0 for No]-->1

1.Inserting a Node              2.Deleting a Node
3.Searching a Node              4.Display in Ascending Order
5.Display                       6.Exit
Enter your proper choice:1

======INSERTION=========
Enter the number you want to insert in an AVL Tree:12

=======
Do you want to continue?[1 for YES || 0 for No]-->1

1.Inserting a Node              2.Deleting a Node
3.Searching a Node              4.Display in Ascending Order
5.Display                       6.Exit
Enter your proper choice:1

======INSERTION=========
Enter the number you want to insert in an AVL Tree:25
```

```
1.Inserting a Node              2.Deleting a Node
3.Searching a Node              4.Display in Ascending Order
5.Display                       6.Exit
Enter your proper choice:5

======ALL TRAVERSALS=========
Inorder Traversal of the Tree is: 12 25 50 60
Preorder Traversal of the Tree is: 50 12 25 60
Postorder Traversal of the Tree is: 25 12 60 50
=======
Do you want to continue?[1 for YES || 0 for No]-->1

1.Inserting a Node              2.Deleting a Node
3.Searching a Node              4.Display in Ascending Order
5.Display                       6.Exit
Enter your proper choice:3

======SEARCHING=========
Enter the number you want to search:25

An element25 is found
=======
```

```
1.Inserting a Node              2.Deleting a Node
3.Searching a Node              4.Display in Ascending Order
5.Display                       6.Exit
Enter your proper choice:4

======ASCENDING ORDER=========
The elements in Ascending Order are: 12 25 50 60
=======
Do you want to continue?[1 for YES || 0 for No]-->1

1.Inserting a Node              2.Deleting a Node
3.Searching a Node              4.Display in Ascending Order
5.Display                       6.Exit
Enter your proper choice:2

======DELETION=========
The numbers in AVL tree are:12 25 50 60
Enter the number you want to delete from an AVL Tree:50

=======
Do you want to continue?[1 for YES || 0 for No]-->1

1.Inserting a Node              2.Deleting a Node
3.Searching a Node              4.Display in Ascending Order
5.Display                       6.Exit
Enter your proper choice:4

======ASCENDING ORDER=========
The elements in Ascending Order are: 12 25 60
=======
Do you want to continue?[1 for YES || 0 for No]-->
```