/\*

\* **Java Program to Implement AVL Tree**

\*/

import java.util.Scanner;

/\* Class AVLNode \*/

class AVLNode

{

AVLNode left, right;

int data;

int height;

/\* Constructor \*/

public AVLNode()

{

left = null;

right = null;

data = 0;

height = 0;

}

/\* Constructor \*/

public AVLNode(int n)

{

left = null;

right = null;

data = n;

height = 0;

}

}

/\* Class AVLTree \*/

class AVLTree

{

private AVLNode root;

/\* Constructor \*/

public AVLTree()

{

root = null;

}

/\* Function to check if tree is empty \*/

public boolean isEmpty()

{

return root == null;

}

/\* Make the tree logically empty \*/

public void makeEmpty()

{

root = null;

}

/\* Function to insert data \*/

public void insert(int data)

{

root = insert(data, root);

}

/\* Function to get height of node \*/

private int height(AVLNode t)

{

return t == null ? -1 : t.height;

}

/\* Function to max of left/right node \*/

private int max(int lhs, int rhs)

{

return lhs > rhs ? lhs : rhs;

}

/\* Function to insert data recursively \*/

private AVLNode insert(int x, AVLNode t)

{

if (t == null)

t = new AVLNode(x);

else if (x < t.data)

{

t.left = insert( x, t.left );

if( height( t.left ) - height( t.right ) == 2 )

if( x < t.left.data )

t = rotateWithLeftChild( t );

else

t = doubleWithLeftChild( t );

}

else if( x > t.data )

{

t.right = insert(x,t.right );

if( height( t.right ) - height( t.left ) == 2 )

if( x > t.right.data)

t = rotateWithRightChild( t );

else

t = doubleWithRightChild( t );

}

else;// Duplicate; do nothing

t.height = max( height( t.left ), height( t.right ) ) + 1;

return t;

}

/\* Rotate binary tree node with left child \*/

private AVLNode rotateWithLeftChild(AVLNode k2)

{

AVLNode k1 = k2.left;

k2.left = k1.right;

k1.right = k2;

k2.height = max( height( k2.left ), height( k2.right ) ) + 1;

k1.height = max( height( k1.left ), k2.height ) + 1;

return k1;

}

/\* Rotate binary tree node with right child \*/

private AVLNode rotateWithRightChild(AVLNode k1)

{

AVLNode k2 = k1.right;

k1.right = k2.left;

k2.left = k1;

k1.height = max( height( k1.left ), height( k1.right ) ) + 1;

k2.height = max( height( k2.right ), k1.height ) + 1;

return k2;

}

/\*\*

\* Double rotate binary tree node: first left child

\* with its right child; then node k3 with new left child \*/

private AVLNode doubleWithLeftChild(AVLNode k3)

{

k3.left = rotateWithRightChild( k3.left );

return rotateWithLeftChild( k3 );

}

/\*\*

\* Double rotate binary tree node: first right child

\* with its left child; then node k1 with new right child \*/

private AVLNode doubleWithRightChild(AVLNode k1)

{

k1.right = rotateWithLeftChild( k1.right );

return rotateWithRightChild( k1 );

}

/\* Functions to count number of nodes \*/

public int countNodes()

{

return countNodes(root);

}

private int countNodes(AVLNode r)

{

if (r == null)

return 0;

else

{

int l = 1;

l += countNodes(r.left);

l += countNodes(r.right);

return l;

}

}

/\* Functions to search for an element \*/

public boolean search(int val)

{

return search(root, val);

}

private boolean search(AVLNode r, int val)

{

boolean found = false;

while ((r != null) && !found)

{

int rval = r.data;

if (val < rval)

r = r.left;

else if (val > rval)

r = r.right;

else

{

found = true;

break;

}

found = search(r, val);

}

return found;

}

/\* Function for inorder traversal \*/

public void inorder()

{

inorder(root);

}

private void inorder(AVLNode r)

{

if (r != null)

{

inorder(r.left);

System.out.print(r.data +" ");

inorder(r.right);

}

}

/\* Function for preorder traversal \*/

public void preorder()

{

preorder(root);

}

private void preorder(AVLNode r)

{

if (r != null)

{

System.out.print(r.data +" ");

preorder(r.left);

preorder(r.right);

}

}

/\* Function for postorder traversal \*/

public void postorder()

{

postorder(root);

}

private void postorder(AVLNode r)

{

if (r != null)

{

postorder(r.left);

postorder(r.right);

System.out.print(r.data +" ");

}

}

}

/\* Class AVL Tree Test \*/

public class AVLTreeTest

{

public static void main(String[] args)

{

Scanner scan = new Scanner(System.in);

/\* Creating object of AVLTree \*/

AVLTree avlt = new AVLTree();

System.out.println("AVLTree Tree Test\n");

char ch;

/\* Perform tree operations \*/

do

{

System.out.println("\nAVLTree Operations\n");

System.out.println("1. insert ");

System.out.println("2. search");

System.out.println("3. count nodes");

System.out.println("4. check empty");

System.out.println("5. clear tree");

int choice = scan.nextInt();

switch (choice)

{

case 1 :

System.out.println("Enter integer element to insert");

avlt.insert( scan.nextInt() );

break; case 2 :

System.out.println("Enter integer element to search");

System.out.println("Search result : "+ avlt.search( scan.nextInt() ));

break;

case 3 :

System.out.println("Nodes = "+ avlt.countNodes());

break;

case 4 :

System.out.println("Empty status = "+ avlt.isEmpty());

break;

case 5 :

System.out.println("\nTree Cleared");

avlt.makeEmpty();

break;

default :

System.out.println("Wrong Entry \n ");

break;

}

/\* Display tree \*/

System.out.print("\nPost order : ");

avlt.postorder();

System.out.print("\nPre order : ");

avlt.preorder();

System.out.print("\nIn order : ");

avlt.inorder();

System.out.println("\nDo you want to continue (Type y or n) \n");

ch = scan.next().charAt(0);

}

while (ch == 'Y'|| ch == 'y');

}

}

**Sample Output:**

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

4

Empty status = true

Post order :

Pre order :

In order :

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

1

Enter integer element to insert

10

Post order : 10

Pre order : 10

In order : 10

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

1

Enter integer element to insert

9

Post order : 9 10

Pre order : 10 9

In order : 9 10

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

1

Enter integer element to insert

8

Post order : 8 10 9

Pre order : 9 8 10

In order : 8 9 10

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

1

Enter integer element to insert

7

Post order : 7 8 10 9

Pre order : 9 8 7 10

In order : 7 8 9 10

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

1

Enter integer element to insert

6

Post order : 6 8 7 10 9

Pre order : 9 7 6 8 10

In order : 6 7 8 9 10

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

1

Enter integer element to insert

5

Post order : 5 6 8 10 9 7

Pre order : 7 6 5 9 8 10

In order : 5 6 7 8 9 10

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

1

Enter integer element to insert

4

Post order : 4 6 5 8 10 9 7

Pre order : 7 5 4 6 9 8 10

In order : 4 5 6 7 8 9 10

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

1

Enter integer element to insert

3

Post order : 3 4 6 5 8 10 9 7

Pre order : 7 5 4 3 6 9 8 10

In order : 3 4 5 6 7 8 9 10

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

1

Enter integer element to insert

2

Post order : 2 4 3 6 5 8 10 9 7

Pre order : 7 5 3 2 4 6 9 8 10

In order : 2 3 4 5 6 7 8 9 10

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

1

Enter integer element to insert

1

Post order : 1 2 4 6 5 3 8 10 9 7

Pre order : 7 3 2 1 5 4 6 9 8 10

In order : 1 2 3 4 5 6 7 8 9 10

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

1

Enter integer element to insert

0

Post order : 0 2 1 4 6 5 3 8 10 9 7

Pre order : 7 3 1 0 2 5 4 6 9 8 10

In order : 0 1 2 3 4 5 6 7 8 9 10

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

3

Nodes = 11

Post order : 0 2 1 4 6 5 3 8 10 9 7

Pre order : 7 3 1 0 2 5 4 6 9 8 10

In order : 0 1 2 3 4 5 6 7 8 9 10

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

2

Enter integer element to search

12

Search result : false

Post order : 0 2 1 4 6 5 3 8 10 9 7

Pre order : 7 3 1 0 2 5 4 6 9 8 10

In order : 0 1 2 3 4 5 6 7 8 9 10

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

2

Enter integer element to search

4

Search result : true

Post order : 0 2 1 4 6 5 3 8 10 9 7

Pre order : 7 3 1 0 2 5 4 6 9 8 10

In order : 0 1 2 3 4 5 6 7 8 9 10

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

5

Tree Cleared

Post order :

Pre order :

In order :

Do you want to continue (Type y or n)

y

AVLTree Operations

1. insert

2. search

3. count nodes

4. check empty

5. clear tree

4

Empty status = true

Post order :

Pre order :

In order :

Do you want to continue (Type y or n)

n