#include <iostream>

using namespace std;

struct AVLNode

{

int data ;

int balfact ;

AVLNode \*left ;

AVLNode \*right ;

} ;

class avltree

{

private :

AVLNode \*root ;

public :

avltree( ) ;

AVLNode\* insert ( int data, int \*h ) ;

static AVLNode\* buildtree ( AVLNode \*root, int data, int \*h ) ;

void display( AVLNode \*root ) ;

AVLNode\* deldata ( AVLNode\* root, int data, int \*h ) ;

static AVLNode\* del ( AVLNode \*node, AVLNode\* root, int \*h ) ;

static AVLNode\* balright ( AVLNode \*root, int \*h ) ;

static AVLNode\* balleft ( AVLNode\* root, int \*h ) ;

void setroot ( AVLNode \*avl ) ;

~avltree( ) ;

static void deltree ( AVLNode \*root ) ;

} ;

avltree :: avltree( )

{

root = NULL ;

}

AVLNode\* avltree :: insert ( int data, int \*h )

{

root = buildtree ( root, data, h ) ;

return root ;

}

AVLNode\* avltree :: buildtree ( AVLNode \*root, int data, int \*h )

{

AVLNode \*node1, \*node2 ;

if ( root == NULL )

{

root = new AVLNode ;

root -> data = data ;

root -> left = NULL ;

root -> right = NULL ;

root -> balfact = 0 ;

\*h = true ;

return ( root ) ;

}

if ( data < root -> data )

{

root -> left = buildtree ( root -> left, data, h ) ;

// If left subtree is higher

if ( \*h )

{

switch ( root -> balfact )

{

case 1 :

node1 = root -> left ;

if ( node1 -> balfact == 1 )

{

cout<<"\nRight rotation.";

root -> left = node1 -> right ;

node1 -> right = root ;

root -> balfact = 0 ;

root = node1 ;

}

else

{

cout << "\nDouble rotation, left then right." ;

node2 = node1 -> right ;

node1 -> right = node2 -> left ;

node2 -> left = node1 ;

root -> left = node2 -> right ;

node2 -> right = root ;

if ( node2 -> balfact == 1 )

root -> balfact = -1 ;

else

root -> balfact = 0 ;

if ( node2 -> balfact == -1 )

node1 -> balfact = 1 ;

else

node1 -> balfact = 0 ;

root = node2 ;

}

root -> balfact = 0 ;

\*h = false ;

break ;

case 0 :

root -> balfact = 1 ;

break ;

case -1 :

root -> balfact = 0 ;

\*h = false ;

}

}

}

if ( data > root -> data )

{

root -> right = buildtree ( root -> right, data, h ) ;

if ( \*h )

{

switch ( root -> balfact )

{

case 1 :

root -> balfact = 0 ;

\*h = false ;

break ;

case 0 :

root -> balfact = -1 ;

break ;

case -1 :

node1 = root -> right ;

if ( node1 -> balfact == -1 )

{

cout << "\nLeft rotation." ;

root -> right = node1 -> left ;

node1 -> left = root ;

root -> balfact = 0 ;

root = node1 ;

}

else

{

cout << "\nDouble rotation, right then left." ;

node2 = node1 -> left ;

node1 -> left = node2 -> right ;

node2 -> right = node1 ;

root -> right = node2 -> left ;

node2 -> left = root ;

if ( node2 -> balfact == -1 )

root -> balfact = 1 ;

else

root -> balfact = 0 ;

if ( node2 -> balfact == 1 )

node1 -> balfact = -1 ;

else

node1 -> balfact = 0 ;

root = node2 ;

}

root -> balfact = 0 ;

\*h = false ;

}

}

}

return ( root ) ;

}

void avltree :: display ( AVLNode\* root )

{

if ( root != NULL )

{

display ( root -> left ) ;

cout << root -> data << "\t" ;

display ( root -> right ) ;

}

}

AVLNode\* avltree :: deldata ( AVLNode \*root, int data, int \*h )

{

AVLNode \*node ;

if ( root -> data == 13 )

cout << root -> data ;

if ( root == NULL )

{

cout << "\nNo such data." ;

return ( root ) ;

}

else

{

if ( data < root -> data )

{

root -> left = deldata ( root -> left, data, h ) ;

if ( \*h )

root = balright ( root, h ) ;

}

else

{

if ( data > root -> data )

{

root -> right = deldata ( root -> right, data, h ) ;

if ( \*h )

root = balleft ( root, h ) ;

}

else

{

node = root ;

if ( node -> right == NULL )

{

root = node -> left ;

\*h = true ;

delete ( node ) ;

}

else

{

if ( node -> left == NULL )

{

root = node -> right ;

\*h = true ;

delete ( node ) ;

}

else

{

node -> right = del ( node -> right, node, h ) ;

if ( \*h )

root = balleft ( root, h ) ;

}

}

}

}

}

return ( root ) ;

}

AVLNode\* avltree :: del ( AVLNode \*succ, AVLNode \*node, int \*h )

{

AVLNode \*temp = succ ;

if ( succ -> left != NULL )

{

succ -> left = del ( succ -> left, node, h ) ;

if ( \*h )

succ = balright ( succ, h ) ;

}

else

{

temp = succ ;

node -> data = succ -> data ;

succ = succ -> right ;

delete ( temp ) ;

\*h = true ;

}

return ( succ ) ;

}

AVLNode\* avltree :: balright ( AVLNode \*root, int \*h )

{

AVLNode \*temp1, \*temp2 ;

switch ( root -> balfact )

{

case 1 :

root -> balfact = 0 ;

break ;

case 0 :

root -> balfact = -1 ;

\*h = false ;

break ;

case -1 :

temp1 = root -> right ;

if ( temp1 -> balfact <= 0 )

{

cout << "\nLeft rotation." ;

root -> right = temp1 -> left ;

temp1 -> left = root ;

if ( temp1 -> balfact == 0 )

{

root -> balfact = -1 ;

temp1 -> balfact = 1 ;

\*h = false ;

}

else

{

root -> balfact = temp1 -> balfact = 0 ;

}

root = temp1 ;

}

else

{

cout << "\nDouble rotation, right then left." ;

temp2 = temp1 -> left ;

temp1 -> left = temp2 -> right ;

temp2 -> right = temp1 ;

root -> right = temp2 -> left ;

temp2 -> left = root ;

if ( temp2 -> balfact == -1 )

root -> balfact = 1 ;

else

root -> balfact = 0 ;

if ( temp2 -> balfact == 1 )

temp1 -> balfact = -1 ;

else

temp1 -> balfact = 0 ;

root = temp2 ;

temp2 -> balfact = 0 ;

}

}

return ( root ) ;

}

AVLNode\* avltree :: balleft ( AVLNode \*root, int \*h )

{

AVLNode \*temp1, \*temp2 ;

switch ( root -> balfact )

{

case -1 :

root -> balfact = 0 ;

break ;

case 0 :

root -> balfact = 1 ;

\*h = false ;

break ;

case 1 :

temp1 = root -> left ;

if ( temp1 -> balfact >= 0 )

{

cout << "\nRight rotation." ;

root -> left = temp1 -> right ;

temp1 -> right = root ;

if ( temp1 -> balfact == 0 )

{

root -> balfact = 1 ;

temp1 -> balfact = -1 ;

\*h = false ;

}

else

{

root -> balfact = temp1 -> balfact = 0 ;

}

root = temp1 ;

}

else

{

cout << "\nDouble rotation, left then right." ;

temp2 = temp1 -> right ;

temp1 -> right = temp2 -> left ;

temp2 -> left = temp1 ;

root -> left = temp2 -> right ;

temp2 -> right = root ;

if ( temp2 -> balfact == 1 )

root -> balfact = -1 ;

else

root -> balfact = 0 ;

if ( temp2-> balfact == -1 )

temp1 -> balfact = 1 ;

else

temp1 -> balfact = 0 ;

root = temp2 ;

temp2 -> balfact = 0 ;

}

}

return ( root ) ;

}

void avltree :: setroot ( AVLNode \*avl )

{

root = avl ;

}

avltree :: ~avltree( )

{

deltree ( root ) ;

}

void avltree :: deltree ( AVLNode \*root )

{

if ( root != NULL )

{

deltree ( root -> left ) ;

deltree ( root -> right ) ;

}

delete ( root ) ;

}

void main( )

{

avltree at ;

AVLNode \*avl = NULL ;

int h ;

system("cls");

avl = at.insert ( 20, &h ) ;

at.setroot ( avl ) ;

avl = at.insert ( 6, &h ) ;

at.setroot ( avl ) ;

avl = at.insert ( 29, &h ) ;

at.setroot ( avl ) ;

avl = at.insert ( 5, &h ) ;

at.setroot ( avl ) ;

avl = at.insert ( 12, &h ) ;

at.setroot ( avl ) ;

avl = at.insert ( 25, &h ) ;

at.setroot ( avl ) ;

avl = at.insert ( 32, &h ) ;

at.setroot ( avl ) ;

avl = at.insert ( 10, &h ) ;

at.setroot ( avl ) ;

avl = at.insert ( 15, &h ) ;

at.setroot ( avl ) ;

avl = at.insert ( 27, &h ) ;

at.setroot ( avl ) ;

avl = at.insert ( 13, &h ) ;

at.setroot ( avl ) ;

cout << endl << "AVL tree:\n" ;

at.display ( avl ) ;

avl = at.deldata ( avl, 20, &h ) ;

at.setroot ( avl ) ;

avl = at.deldata ( avl, 12, &h ) ;

at.setroot ( avl ) ;

cout << endl << "AVL tree after deletion of a node:\n" ;

at.display ( avl ) ;

system("PAUSE");

}