'''============================================================

    Author: Williams Wilman Alvarez Zabala

    ===========================================================

'''

class Nodo:

    ''' Class Node which is made for representing ADT structure contains

        data and links to connect to other nodes.

        Args:

            daata(obj): First Node's data value

        Attributes:

            data(obj): First Node's data value

            hijoIzq: A link that points to its left child node

            hijoDer: A link that ponits to its right child node

            level(int): An integer specifies the nodes's level within

                        the binary tree. Defaults to zero(0)

    '''

    def \_\_init\_\_(self, dato):

        ''' Inits class Nodo with default values in their attributes '''

        self.\_\_data = dato

        self.\_\_HijoIzquierdo = None

        self.\_\_HijoDerecho = None

        self.\_\_level = 0

        self.\_\_parent = None

#. Methods getter and setter for getting and setting the parent

    def setParent(self, parent):

        ''' Setter method for setting the parent node of the

        data structure

        '''

        self.\_\_parent = parent

    def getParent(self):

        ''' Getter method for returning the parent node of the

            data structure

        '''

        return self.\_\_parent

    #. Methods getter and setter for getting and setting the left node

    def getHijoIzquierdo(self):

        ''' Getter method for returning the left child node of the

            data structure

        '''

        return self.\_\_HijoIzquierdo

    def setHijoIzquierdo(self, HI):

        ''' Setter method for setting the left child node of the

            data structure

        '''

        self.\_\_HijoIzquierdo = HI

    #. Methods getter and setter for getting and setting the right node

    def getHijoDerecho(self):

        ''' Getter method for returning the right child node of the

            data structure

        '''

        return self.\_\_HijoDerecho

    def setHijoDerecho(self, HD):

        ''' Setter method for setting the left child node of the

            data structure

        '''

        self.\_\_HijoDerecho = HD

    #. Methods getter and setter for getting and setting data in a node

    def getData(self):

        ''' Getter method for getting the data from a node of the

            data structure

        '''

        return self.\_\_data

    def setData(self, dato):

        ''' Setter method for Setting the data from a node of the

            data structure

        '''

        self.\_\_data = dato

    #. Methods getter and setter for getting and setting node's level

    def setLevel(self, level):

        ''' Setter method for setting the level from a node of the

            data structure

        '''

        self.\_\_level = level

    def getLevel(self):

        ''' Getter method for getting the level from a node of the

            data structure

        '''

        return self.\_\_level

'''============================================================

    Author: Williams Wilman Alvarez Zabala

    ===========================================================

'''

from AvlTreeNodeClass import Nodo

class ArbolAVL:

    ''' Class ArbolAVL which represents AVLTree ADT is self balanging

        tree that cointains a collection of nodes which will be ordered

        by standard rules.

        Attributes:

            Raiz: It creates the link to the first node in an AVLTree.

                  It defaults to None.

            Rango: It's the balance factor. Defaults 1.

    '''

    def \_\_init\_\_(self):

        ''' Inits a instance of AvlTree with default values in their

             attirbutes

        '''

        self.\_\_raiz = None

        self.\_\_Rango = 1

    def setRoot(self,link):

        ''' Setter method for setting the root node of AVLTree

            Args:

                link(Node)

        '''

        self.raiz = link

    def getRoot(self):

        ''' Getter method for returning the root node of AVLTree '''

        return self.\_\_raiz

    def alturaRecursivo(self, nodo):

        ''' Determines a node's height

            Args:

                nodo(Node)

        '''

        if (nodo == None):

            return 0

        else:

            alturaIzquierdo = self.alturaRecursivo(nodo.getHijoIzquierdo())

            alturaDerecho = self.alturaRecursivo(nodo.getHijoDerecho())

            if (alturaIzquierdo > alturaDerecho):

                return alturaIzquierdo + 1

            else:

                return alturaDerecho + 1

    def existeDato(self, dato):

        ''' Go through all the AVLTree to verify if a node already

            owns a data

            Args:

                dato(int)

            Return:

                true: Data already exists in the AVLTree

                false: Data doesn't exist in the AVLTree

        '''

        if (self.esVacio()):

            return False

        nodo = self.\_\_raiz

        while (nodo != None):

            if (dato == nodo.getData()):

                return True

            else:

                if (dato < nodo.getData()):

                    nodo = nodo.getHijoIzquierdo()

                else:

                    nodo = nodo.getHijoDerecho()

        return False

    def esVacio(self):

        ''' Verify if an AVLTree is empty

            Return:

                true: it's empty

                flase: it's not empty

        '''

        if self.\_\_raiz == None:

            return True

        else:

            return False

    def insertar(self, dato):

        ''' Method that add a node to the AVLTree in their correct

            position by following standard rules in order to maintain

            the balance in the AVLTree

            Args:

                dato(int): data is being attempted to insert

        '''

        if (self.existeDato(dato)):

            return None

        else:

            self.\_\_raiz = self.insertarRecursivo(self.\_\_raiz,dato)

            #self.\_\_raiz = self.UpdateParent()

            #self.UpdateLevel(self.\_\_Raiz)

    def insertarRecursivo(self, nodo, dato):

        ''' Recursive Method which is in charge to complete the

            insertation of a data within the AVLTree.

            Args:

                nodo(Node): the root of the AVLTree to determine where

                            the new data is going to be inserted

                dato(int): data is being attempted to insert

            Return:

                Data has been placed in the correct location in the

                AVLTree

        '''

        if (nodo == None):

            return Nodo(dato)

        else:

            if (dato < nodo.getData()):

                nodo.setHijoIzquierdo(self.insertarRecursivo(nodo.getHijoIzquierdo(), dato))

                nodo.getHijoIzquierdo().setParent(nodo)

            else:

                if (dato > nodo.getData()):

                    nodo.setHijoDerecho(self.insertarRecursivo(nodo.getHijoDerecho(), dato))

                    nodo.getHijoDerecho().setParent(nodo)

        return self.balancear(nodo)

    def balancear(self, nodo):

        ''' Method assigned to verify if the AVLTree whether or not

            need to be balanced in case necessary

            Args:

                nodo(Node)

            Return:

                The AVLTree balanced

        '''

        alturaIzquierdo = self.alturaRecursivo(nodo.getHijoIzquierdo())

        alturaDerecho = self.alturaRecursivo(nodo.getHijoDerecho())

        if ((alturaIzquierdo - alturaDerecho) > self.\_\_Rango):

            nodoIzquierdo = nodo.getHijoIzquierdo()

            if (self.alturaRecursivo(nodoIzquierdo.getHijoDerecho()) > self.alturaRecursivo(nodoIzquierdo.getHijoDerecho())):

                return self.rotacionDobleDerecho(nodo)

            else:

                return self.rotacionSimpleDerecho(nodo)

        else:

            if ((alturaDerecho - alturaIzquierdo) > self.\_\_Rango):

                nodoDerecho = nodo.getHijoDerecho()

                if (self.alturaRecursivo(nodoDerecho.getHijoIzquierdo()) > self.alturaRecursivo(nodoDerecho.getHijoDerecho())):

                    return self.rotacionDobleIzquierdo(nodo)

                else:

                    return self.rotacionSimpleIzquierdo(nodo)

        return nodo

    def FindNode(self, valor):

        ''' Method go through AVLTree looking for a node whit a

        specific value

        Args:

            valor(int): value passed which is going to be searched

                        within the AVLTree

        Return:

            A reference to the node of value if found

            otherwise returns 0

        '''

        return self.FindNodeR(self.\_\_raiz, valor)

    def FindNodeR(self,raiz,valor):

        ''' Recursive method which is in charge to complete

            the looking-for within the whole AVLTree

            Note:

                This method also can be used to look for within a

                subtree by passing the subtree as an arg.

            Args:

                raiz(Node): The whole tree or a specific subtree

                valor(int): The value is being looked for

            Return:

                A reference to the node of value if found

                otherwise returns 0

        '''

        if raiz.getData() == valor:

            return raiz

        elif valor < raiz.getData():

            return self.FindNodeR(raiz.getHijoIzquierdo(), valor)

        elif valor > raiz.getData():

            return self.FindNodeR(raiz.getHijoDerecho(), valor)

        else:

            return 0

    def rotacionSimpleIzquierdo(self, nodo):

        ''' Left Rotation moves nodes to the left in order to balance

            the tree

            Args:

                nodo(Node)

            Return:

                The AVLTree has been balanced and its properties

                has been conserved

        '''

        nuevoNodo = nodo.getHijoDerecho()

        nodo.setHijoDerecho(nuevoNodo.getHijoIzquierdo())

        nuevoNodo.setHijoIzquierdo(nodo)

        # Now let's update parents node

        newparent = nuevoNodo.getHijoIzquierdo().getParent()

        nuevoNodo.setParent(newparent)

        nuevoNodo.getHijoIzquierdo().setParent(nuevoNodo)

        nuevoNodo.getHijoDerecho().setParent(nuevoNodo)

        return nuevoNodo

    def rotacionSimpleDerecho(self, nodo):

        ''' Right Rotation moves nodes to the right in order to balance

            the tree

            Args:

                nodo(Node)

            Return:

                The AVLTree has been balanced and its properties

                has been conserved

        '''

        nuevoNodo = nodo.getHijoIzquierdo()

        nodo.setHijoIzquierdo(nuevoNodo.getHijoDerecho())

        nuevoNodo.setHijoDerecho(nodo)

        # Now let's update parents node

        newparent = nuevoNodo.getHijoDerecho().getParent()

        nuevoNodo.setParent(newparent)

        nuevoNodo.getHijoDerecho().setParent(nuevoNodo)

        nuevoNodo.getHijoIzquierdo().setParent(nuevoNodo)

        return nuevoNodo

    def rotacionDobleIzquierdo(self, nodo):

        ''' Double Left Rotation moves nodes to the right and then to

            the left in order to balance the tree

            Args:

                nodo(Node)

            Return:

                The AVLTree has been balanced and its properties

                has been conserved

        '''

        nodo.setHijoDerecho(self.rotacionSimpleDerecho(nodo.getHijoDerecho()))

        return self.rotacionSimpleIzquierdo(nodo)

    def rotacionDobleDerecho(self, nodo):

        ''' Double Right Rotation moves nodes to the left and then to

            the right in order to balance the tree

            Args:

                nodo(Node)

            Return:

                The AVLTree has been balanced and its properties

                has been conserved

        '''

        nodo.setHijoIzquierdo(self.rotacionSimpleIzquierdo(nodo.getHijoIzquierdo()))

        return self.rotacionSimpleDerecho(nodo)

    def printInorder(self, node):

        ''' Go through all the AVLTree to print them in order '''

        if(node!=None):

            self.printInorder(node.getHijoIzquierdo())

            print(node.getData())

            self.printInorder(node.getHijoDerecho())

def main():

    print("Test of the AVLTree")

    A1 = ArbolAVL()

    A1.insertar(200)

    A1.insertar(350)

    A1.insertar(150)

    A1.insertar(100)

    A1.insertar(90)

    print(A1.printInorder(A1.getRoot()))

if \_\_name\_\_ == "\_\_main\_\_":

    main()