**Actividad de Aprendizaje 13. El Árbol AVL, implementación dinámica**

***Problema:***

Reutilice el programa resultante de la actividad 12. Agregue y/o modifique los métodos necesarios para que el árbol sea implementado como AVL.

***Requerimientos:***

a)      El estilo de programación debe ser Orientado a Objetos

Estructura de Datos I

Christopher Ceballos Jiménez

219750442

Esta actividad resulto mas sencilla ya que los métodos faltantes eran muy similares al árbol binario con un pequeño cambio en deleteData.

MAIN

#include <iostream>  
#include <random>  
#include <chrono>  
#include <functional>  
  
#include "avltree.h"  
  
**using** **namespace** std**;**  
  
int main **(** **)** **{**  
       
     default\_random\_engine generator**(**chrono**::**system\_clock**::**now**().**time\_since\_epoch**().**count**());**  
     uniform\_int\_distribution**<**int**>** distribution**(**0**,**100000**);**  
     **auto** dice **=** bind**(**distribution**,** generator**);**  
       
     char op**;**  
     int value**,** x**;**  
     AVLTree **<**int**>** myTree**;**  
       
     **do{**  
          system**(**"cls"**);**  
            
          cout **<<** "Ingresa un cualquier numero entero a utilizar: "**;** cin **>>** x**;**  
            
          cout **<<** endl **<<** endl**;**  
            
          **for** **(**int i**(**0**);** i **<** x**;** i**++){**  
                 
               value **=** dice**();**  
               cout **<<** "Insertando: " **<<** value **<<** endl**;**  
                 
               myTree**.**insertData**(**value**);**  
          **}**  
            
          cout **<<** endl **<<** endl**;**  
            
          cout **<<** "Recorrido en PreOrder: " **<<** endl**;**  
          myTree**.**parsePreOrder**();**  
          cout **<<** endl **<<** endl**;**  
            
          cout **<<** "Recorrido en InOrder: " **<<** endl**;**  
          myTree**.**parseInOrder**();**  
          cout **<<** endl **<<** endl**;**  
            
          cout **<<** "Recorrido en PostOrder: " **<<** endl**;**  
          myTree**.**parsePostOrder**();**  
          cout **<<** endl **<<** endl**;**  
            
          cout **<<** "Altura del subarbol izquierdo: " **<<** myTree**.**getHeight**(**myTree**.**getRoot**()** **->** getLeft**());**  
          cout **<<** endl **<<** endl**;**  
            
          cout **<<** "Altura del subarbol derecho: " **<<** myTree**.**getHeight**(**myTree**.**getRoot**()** **->** getRight**());**  
            
          cout **<<** endl **<<** endl **<<** "Presiona 0 para salir. Presiona cualquier tecla para intentar nuevamente: "**;** cin **>>** op**;**  
            
          **if(**op **!=** '0'**){**  
               myTree**.**deleteAll**();**  
          **}**  
            
     **}while(**op **!=** '0'**);**  
  
AVLTREE

#ifndef AVLTree\_H\_INCLUDED  
#define AVLTree\_H\_INCLUDED  
  
#include <exception>  
#include <string>  
  
  
**template** **<class** T**>**  
**class** AVLTree **{**  
**private:**  
       
**class** Node **{**  
**private:**  
T**\*** dataPtr**;**  
Node**\*** left**;**  
Node**\*** right**;**  
  
**public:**  
**class** Exception **:** **public** std**::**exception**{**  
**private:**  
std**::**string msg**;**  
  
**public:**  
**explicit** Exception**(const** char**\*** message**)** **:** msg**(**message**){** **}**  
  
**explicit** Exception**(const** std**::**string**&** message**)** **:** msg**(**message**){** **}**  
  
**virtual** **~**Exception**()** **throw(){** **}**  
  
**virtual** **const** char**\*** what**()** **const** **throw** **()** **{**  
**return** msg**.**c\_str**();**  
**}**  
**};**  
  
Node**();**  
Node**(const** T**&);**  
  
**~**Node**();**  
  
T**\*** getDataPtr**()** **const;**  
T getData**()** **const;**  
Node**\*&** getLeft**();**  
Node**\*&** getRight**();**  
  
void setDataPtr**(**T**\*);**  
void setData**(const** T**&);**  
void setLeft**(**Node**\*&);**  
void setRight**(**Node**\*&);**  
**};**  
  
Node**\*** root**;**  
  
void copyAll**(const** AVLTree**<**T**>&);**  
void copyAll**(**Node**\*&,** Node**\*);**  
  
void insertData**(**Node**\*&,** **const** T**&);**  
  
Node**\*&** findData**(**Node**\*&,** **const** T**&);**  
  
Node**\*&** getLowest**(**Node**\*&);**  
Node**\*&** getHighest**(**Node**\*&);**  
  
void parsePreOrder**(**Node**\*&);**  
void parseInOrder**(**Node**\*&);**  
void parsePostOrder**(**Node**\*&);**  
  
/// AVL.  
int getBalanceFactor**(**Node**\*&);**  
  
void simpleLeftRot**(**Node**\*&);**  
void simpleRightRot**(**Node**\*&);**  
void doubleLeftRot**(**Node**\*&);**  
void doubleRightRot**(**Node**\*&);**  
  
void doBalancing**(**Node**\*&);**  
  
void deleteAll**(**Node**\*&);**  
  
**public:**  
**class** Exception **:** **public** std**::**exception**{**  
**private:**  
std**::**string msg**;**  
  
**public:**  
**explicit** Exception**(const** char**\*** message**)** **:** msg**(**message**){** **}**  
  
**explicit** Exception**(const** std**::**string**&** message**)** **:** msg**(**message**){** **}**  
  
**virtual** **~**Exception**()** **throw(){** **}**  
  
**virtual** **const** char**\*** what**()** **const** **throw** **()** **{**  
**return** msg**.**c\_str**();**  
**}**  
**};**  
  
AVLTree**();**  
AVLTree**(const** AVLTree**&);**  
  
**~**AVLTree**();**  
  
bool isEmpty**()** **const;**  
  
void insertData**(const** T**&);**  
  
void deleteData**(**Node**\*&);**  
  
T retrieve**(**Node**\*&)** **const;**  
  
int getHeight**(**Node**\*&);**  
  
Node**\*&** findData**(const** T**&);**  
  
Node**\*&** getLowest**();**  
Node**\*&** getHighest**();**  
  
bool isLeaf**(**Node**\*&)** **const;**  
  
int getHeight**();**  
int getHeightLeft**();**  
int getHeightRight**();**  
  
void parsePreOrder**();**  
void parseInOrder**();**  
void parsePostOrder**();**  
  
void deleteAll**();**  
  
Node**\*&** getRoot**();**  
  
AVLTree**&** **operator** **=** **(const** AVLTree**&);**  
**};**  
  
///Node  
**template** **<class** T**>**  
AVLTree**<**T**>::**Node**::**Node**()** **:** dataPtr**(nullptr),** left**(nullptr),** right**(nullptr)** **{** **}**  
  
**template** **<class** T**>**  
AVLTree**<**T**>::**Node**::**Node**(const** T**&** e**)** **:** dataPtr**(new** T**(**e**)),** left**(nullptr),** right**(nullptr)** **{**  
**if(**dataPtr **==** **nullptr){**  
**throw** Exception**(**"Memoria insuficiente, creando nodo"**);**  
**}**  
**}**  
  
**template** **<class** T**>**  
AVLTree**<**T**>::**Node**::~**Node**()** **{**  
**delete** dataPtr**;**  
**}**  
  
**template** **<class** T**>**  
T**\*** AVLTree**<**T**>::**Node**::**getDataPtr**()** **const** **{**  
**return** dataPtr**;**  
**}**  
  
**template** **<class** T**>**  
T AVLTree**<**T**>::**Node**::**getData**()** **const** **{**  
**if(**dataPtr **==** **nullptr){**  
**throw** Exception**(**"Dato inexistente, getData"**);**  
**}**  
**return** **\***dataPtr**;**  
**}**  
  
**template** **<class** T**>**  
**typename** AVLTree**<**T**>::**Node**\*&** AVLTree**<**T**>::**Node**::**getLeft**()** **{**  
**return** left**;**  
**}**  
  
**template** **<class** T**>**  
**typename** AVLTree**<**T**>::**Node**\*&** AVLTree**<**T**>::**Node**::**getRight**()** **{**  
**return** right**;**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**Node**::**setDataPtr**(**T**\*** p**)** **{**  
dataPtr **=** p**;**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**Node**::**setData**(const** T**&** e**){**  
**if(**dataPtr **==** **nullptr){**  
**if((**dataPtr **=** **new** T**(**e**))==nullptr){**  
**throw** Exception**(**"Memoria no disponible, setData"**);**  
**}**  
**}**  
**else{**  
**\***dataPtr **=** e**;**  
**}**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**Node**::**setLeft**(**Node**\*&** p**)** **{**  
left **=** p**;**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**Node**::**setRight**(**Node**\*&** p**)** **{**  
right **=** p**;**  
**}**  
  
  
///Binary Tree  
**template** **<class** T**>**  
void AVLTree**<**T**>::**copyAll**(const** AVLTree**<**T**>&** t**)** **{**  
     copyAll**(**root**,** t**.**root**);**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**copyAll**(**Node**\*&** nR**,** Node**\*** cR**)** **{**  
       
     **if(**cR **==** **nullptr)** **{**  
          **return;**  
     **}**  
       
     **if((**nR **=** **new** Node**(**cR **->** getData**()))** **==** **nullptr)** **{**  
          **throw** Exception**(**"Memoria Insuficiente, copyAll"**);**  
     **}**  
       
     copyAll**(**nR **->** getLeft**()** **,**cR **->** getLeft**());**  
     copyAll**(**nR **->** getRight**()** **,**cR **->** getRight**());**  
**}**  
  
**template** **<class** T**>**  
AVLTree**<**T**>::**AVLTree**()** **:** root**(nullptr)** **{** **}**  
  
**template** **<class** T**>**  
AVLTree**<**T**>::**AVLTree**(const** AVLTree**&** t**)** **:** root**(nullptr)** **{**  
copyAll**(**t**);**  
**}**  
  
**template** **<class** T**>**  
AVLTree**<**T**>::~**AVLTree**()** **{**  
deleteAll**();**  
**}**  
  
**template** **<class** T**>**  
bool AVLTree**<**T**>::**isEmpty**()** **const** **{**  
**return** root **==** **nullptr;**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**insertData**(const** T**&** e**)** **{**  
insertData**(**root**,** e**);**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**insertData**(**Node**\*&** r**,** **const** T**&** e**)** **{**  
**if(**r **==** **nullptr){**  
**try{**  
**if((**r **=** **new** Node**(**e**))** **==** **nullptr){**  
**throw** Exception**(**"Memoria no disponible, insertData"**);**  
**}**  
**}** **catch** **(typename** Node**::**Exception ex**){**  
**throw** Exception**(**ex**.**what**());**  
**}**  
**}**  
**else{**  
**if(**e **<** r**->**getData**()){**  
insertData**(**r**->**getLeft**(),** e**);**  
**}**  
**else{**  
insertData**(**r**->**getRight**(),** e**);**  
**}**  
**}**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**simpleLeftRot**(**Node**\*&** r**)** **{**  
Node**\*** aux1**(**r**->**getRight**());**  
Node**\*** aux2**(**aux1**->**getLeft**());**  
r**->**setRight**(**aux2**);**  
aux1**->**setLeft**(**r**);**  
r**=**aux1**;**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**simpleRightRot**(**Node**\*&** r**)** **{**  
Node**\*** aux1**(**r**->**getRight**());**  
Node**\*** aux2**(**aux1**->**getLeft**());**  
r**->**setLeft**(**aux2**);**  
aux1**->**setRight**(**r**);**  
r**=**aux1**;**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**doubleLeftRot**(**Node**\*&** r**)** **{**  
simpleRightRot**(**r**->**getRight**());**  
simpleLeftRot**(**r**);**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**doubleRightRot**(**Node**\*&** r**)** **{**  
simpleLeftRot**(**r**->**getLeft**());**  
simpleRightRot**(**r**);**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**doBalancing**(**Node**\*&** r**)** **{**  
**switch(**getBalanceFactor**(**r**)){**  
**case** **-**2**:**  
**if(**getBalanceFactor**(**r**->**getLeft**())==-**1**){**  
simpleRightRot**(**r**);**  
**}**  
**else{**  
doubleRightRot**(**r**);**  
**}**  
**break;**  
**case** 2**:**  
**if(**getBalanceFactor**(**r**->**getRight**())==**1**){**  
simpleLeftRot**(**r**);**  
**}**  
**else{**  
doubleLeftRot**(**r**);**  
**}**  
**break;**  
**default:**  
**;**  
**}**  
**}**  
  
**template** **<class** T**>**  
int AVLTree**<**T**>::**getBalanceFactor**(**Node**\*&** r**)** **{**  
**return** getHeight**(**r**->**getRight**())** **-** getHeight**(**r**->**getLeft**());**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**deleteData **(**Node**\*&** e**)** **{**  
       
     **if(**root **==** **nullptr** **or** e **==** **nullptr)** **{**  
          **throw** Exception**(**"Insuficiencia de datos, deleteData"**);**  
     **}**  
       
     **if(**isLeaf**(**e**))** **{**  
          **delete** e**;**  
          e **=** **nullptr;**  
          doBalancing**(**root**);**  
     **}**  
     **else** **{**  
          **if(**e **->** getLeft**()** **!=** **nullptr)** **{**  
               e **->** setData**((**getHighest**(**e **->** getLeft**()))** **->** getData**());**  
               deleteData**((**getHighest**(**e **->** getLeft**())));**  
          **}**  
          **else** **{**  
               e **->** setData**((**getLowest**(**e **->** getRight**()))** **->** getData**());**  
               deleteData**((**getLowest**(**e **->** getRight**())));**  
          **}**  
     **}**  
**}**  
  
**template** **<class** T**>**  
T AVLTree**<**T**>::**retrieve**(**Node**\*&** r**)** **const** **{**  
**return** r**->**getData**();**  
**}**  
  
**template** **<class** T**>**  
**typename** AVLTree**<**T**>::**Node**\*&** AVLTree**<**T**>::**findData**(const** T**&** e**)** **{**  
**return** findData**(**root**,** e**);**  
**}**  
  
**template** **<class** T**>**  
**typename** AVLTree**<**T**>::**Node**\*&** AVLTree**<**T**>::**findData**(**Node**\*&** r**,** **const** T**&** e**)** **{**  
**if(**r **==** **nullptr** **or** r**->**getData**()** **==** e**){**  
**return** r**;**  
**}**  
**if(**e **<** r**->**getData**())** **{**  
**return** findData**(**r**->**getLeft**(),** e**);**  
**}**  
**return** findData**(**r**->**getRight**(),** e**);**  
**}**  
  
**template** **<class** T**>**  
**typename** AVLTree**<**T**>::**Node**\*&** AVLTree**<**T**>::**getLowest**()** **{**  
**return** getLowest**(**root**);**  
**}**  
  
**template** **<class** T**>**  
**typename** AVLTree**<**T**>::**Node**\*&** AVLTree**<**T**>::**getLowest**(**Node**\*&** r**)** **{**  
**if(**r **==** **nullptr** **or** r**->**getLeft**()** **==** **nullptr){**  
**return** r**;**  
**}**  
**return** getLowest**(**r**->**getLeft**());**  
**}**  
  
**template** **<class** T**>**  
**typename** AVLTree**<**T**>::**Node**\*&** AVLTree**<**T**>::**getHighest**()** **{**  
**return** getHighest**(**root**);**  
**}**  
  
**template** **<class** T**>**  
**typename** AVLTree**<**T**>::**Node**\*&** AVLTree**<**T**>::**getHighest**(**Node**\*&** r**)** **{**  
**if(**r **==** **nullptr** **or** r**->**getRight**()** **==** **nullptr){**  
**return** r**;**  
**}**  
**return** getHighest**(**r**->**getRight**());**  
**}**  
  
**template** **<class** T**>**  
bool AVLTree**<**T**>::**isLeaf**(**Node**\*&** r**)** **const** **{**  
**return** r **!=** **nullptr** **and** r**->**getLeft**()** **==** r**->**getRight**();**  
**}**  
  
**template** **<class** T**>**  
int AVLTree**<**T**>::**getHeight**()** **{**  
**return** getHeight**(**root**);**  
**}**  
  
**template** **<class** T**>**  
int AVLTree**<**T**>::**getHeightLeft**()** **{**  
**return** getHeight**(**root**->**getLeft**());**  
**}**  
  
**template** **<class** T**>**  
int AVLTree**<**T**>::**getHeightRight**()** **{**  
**return** getHeight**(**root**->**getRight**());**  
**}**  
  
**template** **<class** T**>**  
int AVLTree**<**T**>::**getHeight**(**Node**\*&** r**)** **{**  
**if(**r **==** **nullptr){**  
**return** 0**;**  
**}**  
int lH**(**getHeight**(**r**->**getLeft**()));**  
int rH**(**getHeight**(**r**->**getRight**()));**  
**return** **(**lH **>** rH **?** lH **:** rH**)** **+** 1**;**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**parsePreOrder**()** **{**  
parsePreOrder**(**root**);**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**parsePreOrder**(**Node**\*&** r**)** **{**  
**if(**r **==** **nullptr){**  
**return;**  
**}**  
std**::**cout **<<** r**->**getData**()** **<<** ", "**;**  
parsePreOrder**(**r**->**getLeft**());**  
parsePreOrder**(**r**->**getRight**());**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**parseInOrder**()** **{**  
parseInOrder**(**root**);**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**parseInOrder**(**Node**\*&** r**)** **{**  
**if(**r **==** **nullptr){**  
**return;**  
**}**  
parseInOrder**(**r**->**getLeft**());**  
std**::**cout **<<** r**->**getData**()** **<<** ", "**;**  
parseInOrder**(**r**->**getRight**());**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**parsePostOrder**()** **{**  
parsePostOrder**(**root**);**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**parsePostOrder**(**Node**\*&** r**)** **{**  
**if(**r **==** **nullptr){**  
**return;**  
**}**  
parsePostOrder**(**r**->**getLeft**());**  
parsePostOrder**(**r**->**getRight**());**  
std**::**cout **<<** r**->**getData**()** **<<** ", "**;**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**deleteAll **(** **)** **{**  
       
     deleteAll**(**root**);**  
     root **=** **nullptr;**  
**}**  
  
**template** **<class** T**>**  
void AVLTree**<**T**>::**deleteAll **(**Node**\*&** r**)** **{**  
       
     **if(**r **==** **nullptr){**  
          **return;**  
     **}**  
       
     deleteAll**(**r **->** getLeft**());**  
     deleteAll**(**r **->** getRight**());**  
       
     **delete** r**;**  
**}**  
  
**template** **<class** T**>**  
AVLTree**<**T**>&** AVLTree**<**T**>::operator** **=** **(const** AVLTree**&** t**)** **{**  
deleteAll**();**  
copyAll**(**t**);**  
**return** **\*this;**  
**}**  
  
**template** **<class** T**>**  
**typename** AVLTree**<**T**>::**Node**\*&** AVLTree**<**T**>::**getRoot**(){**  
     **return** root**;**  
**}**  
  
#endif *// AVLTree\_H\_INCLUDED*

CAPTURA

