**Actividad de Aprendizaje 12. El Árbol Binario de Búsqueda, implementación dinámica**

***Problema:***

Haga un programa que genere una cantidad **N** (definida por el usuario) de valores aleatorios de tipo entero en el rango de 0 a 100,000 que se inserte al árbol conforme cada valor sea generado.

El programa mostrará en pantalla los valores generados en el orden en que se insertan, enseguida el resultado de los recorridos preorder, inorder y postorder, seguido de la altura correspondiente al subárbol izquierdo y al subárbol derecho debajo de la raíz del árbol.

***Requerimientos:***

a)      El estilo de programación debe ser Orientado a Objetos

Estructuras de Datos I

Christopher Ceballos Jimenez

Entender un árbol de forma que pudiera realizar los métodos faltantes fue complicado. Realice más investigación para poder entender a fondo el funcionamiento del árbol y algunos ejemplos donde implementaban métodos distintos. Al final, creo que cumplí con el objetivo.

MAIN

#include <iostream>  
#include <random>  
#include <chrono>  
#include <functional>  
  
#include "btree.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**;**  
     BTree **<**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'**);**  
**}**

ARBOL BINARIO

#ifndef BTREE\_H  
#define BTREE\_H  
  
#include "bnode.h"  
  
#include <exception>  
#include <string>  
  
**template** **<class** T**>**  
**class** BTree**{**  
**private:**  
       
     BNode**<**T**>\*** root**;**  
       
     void copyAll**(const** BTree**<**T**>&);**  
     void copyAll**(**BNode**<**T**>\*&,** BNode**<**T**>\*);**  
       
     void insertData**(**BNode**<**T**>\*&,** **const** T**&);**  
       
     BNode**<**T**>\*&** findData**(**BNode**<**T**>\*&,** **const** T**&);**  
       
     BNode**<**T**>\*&** getLowest**(**BNode**<**T**>\*&);**  
     BNode**<**T**>\*&** getHighest**(**BNode**<**T**>\*&);**  
       
     void parsePreOrder**(**BNode**<**T**>\*&);**  
     void parseInOrder**(**BNode**<**T**>\*&);**  
     void parsePostOrder**(**BNode**<**T**>\*&);**  
       
     void deleteAll**(**BNode**<**T**>\*&);**  
       
**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**();**  
          **}**  
            
     **};**  
       
     BTree**();**  
     BTree**(const** BTree**&);**  
       
     **~**BTree**();**  
       
     BNode**<**T**>\*&** getRoot**();**  
       
     bool isEmpty**()** **const;**  
       
     void insertData**(const** T**&);**  
       
     void deleteData**(**BNode**<**T**>\*&);**  
       
     T retrieve**(**BNode**<**T**>\*&)** **const;**  
       
     BNode**<**T**>\*&** findData**(const** T**&);**  
       
     BNode**<**T**>\*&** getLowest**();**  
     BNode**<**T**>\*&** getHighest**();**  
       
     bool isLeaf**(**BNode**<**T**>\*&)** **const;**  
       
     int getHeight**();**  
     int getHeight**(**BNode**<**T**>\*&);**  
       
     void parsePreOrder**();**  
     void parseInOrder**();**  
     void parsePostOrder**();**  
       
     void deleteAll**();**  
       
     BTree**&** **operator** **=** **(const** BTree**&);**  
       
**};**  
  
**template** **<class** T**>**  
void BTree**<**T**>::**copyAll**(const** BTree**<**T**>&** t**)** **{**  
     copyAll**(**root**,** t**.**root**);**  
**}**  
  
**template** **<class** T**>**  
void BTree**<**T**>::**copyAll**(**BNode**<**T**>\*&** nR**,** BNode**<**T**>\*** cR**)** **{**  
       
     **if(**cR **==** **nullptr)** **{**  
          **return;**  
     **}**  
       
     **if((**nR **=** **new** BNode**<**T**>(**cR **->** getData**()))** **==** **nullptr)** **{**  
          **throw** Exception**(**"Memoria Insuficiente, copyAll"**);**  
     **}**  
       
     copyAll**(**nR **->** getLeft**()** **,**cR **->** getLeft**());**  
     copyAll**(**nR **->** getRight**()** **,**cR **->** getRight**());**  
**}**  
  
**template** **<class** T**>**  
BTree**<**T**>::**BTree **(** **)** **:** root**(nullptr)** **{** **}**  
  
**template** **<class** T**>**  
BTree**<**T**>::**BTree **(const** BTree**&** t**)** **:** root**(nullptr){**  
     copyAll**(**t**);**  
**}**  
  
**template** **<class** T**>**  
BTree**<**T**>::~**BTree **(** **)** **{**  
     deleteAll**();**  
**}**  
  
**template** **<class** T**>**  
BNode**<**T**>\*&** BTree**<**T**>::**getRoot**(){**  
     **return** root**;**  
**}**  
  
**template** **<class** T**>**  
bool BTree**<**T**>::**isEmpty **(** **)** **const** **{**  
     **return** root **==** **nullptr;**  
**}**  
  
**template** **<class** T**>**  
void BTree**<**T**>::**insertData **(const** T**&** e**)** **{**  
     insertData**(**root**,** e**);**  
**}**  
  
**template** **<class** T**>**  
void BTree**<**T**>::**insertData **(**BNode**<**T**>\*&** r**,** **const** T**&** e**)** **{**  
       
     **if(**r **==** **nullptr){**  
          **try{**  
               **if((**r **=** **new** BNode**<**T**>(**e**))** **==** **nullptr){**  
                    **throw** Exception**(**"Memoria no disponible, InsertData."**);**  
               **}**  
          **}catch(typename** BNode**<**T**>::**Exception ex**)** **{**  
               **throw** Exception**(**ex**.**what**());**  
          **}**  
     **}**  
     **else{**  
          **if(**e **<** r **->** getData**()){**  
               insertData**(**r **->** getLeft**(),** e**);**  
          **}**  
          **else{**  
               insertData**(**r **->** getRight**(),** e**);**  
          **}**  
     **}**  
**}**  
**template** **<class** T**>**  
void BTree**<**T**>::**deleteData **(**BNode**<**T**>\*&** e**)** **{**  
       
     **if(**root **==** **nullptr** **or** e **==** **nullptr)** **{**  
          **throw** Exception**(**"Insuficiencia de datos, deleteData"**);**  
     **}**  
       
     **if(**isLeaf**(**e**))** **{**  
          **delete** e**;**  
          e **=** **nullptr;**  
     **}**  
     **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 BTree**<**T**>::**retrieve **(**BNode**<**T**>\*&** r**)** **const** **{**  
       
     **if(**r **==** **nullptr){**  
          **throw** Exception**(**"Elemento inexistente, retrieve."**);**  
     **}**  
       
     **return** r **->** getData**();**  
**}**  
  
**template** **<class** T**>**  
BNode**<**T**>\*&** BTree**<**T**>::**findData **(const** T**&** e**)** **{**  
     **return** findData**(**root**,** e**);**  
**}**  
  
**template** **<class** T**>**  
BNode**<**T**>\*&** BTree**<**T**>::**findData **(**BNode**<**T**>\*&** 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**>**  
BNode**<**T**>\*&** BTree**<**T**>::**getLowest **(** **)** **{**  
     **return** getLowest**(**root**);**  
**}**  
  
**template** **<class** T**>**  
BNode**<**T**>\*&** BTree**<**T**>::**getLowest **(**BNode**<**T**>\*&** r**)** **{**  
       
     **if(**r **==** **nullptr** **or** r **->** getLeft**()** **==** **nullptr){**  
          **return** r**;**  
     **}**  
       
     **return** getLowest**(**r **->** getLeft**());**  
**}**  
  
**template** **<class** T**>**  
BNode**<**T**>\*&** BTree**<**T**>::**getHighest **(** **)** **{**  
     **return** getHighest**(**root**);**  
**}**  
  
**template** **<class** T**>**  
BNode**<**T**>\*&** BTree**<**T**>::**getHighest **(**BNode**<**T**>\*&** r**)** **{**  
       
     **if(**r **==** **nullptr** **or** r **->** getRight**()** **==** **nullptr){**  
          **return** r**;**  
     **}**  
       
     **return** getHighest**(**r **->** getRight**());**  
**}**  
  
**template** **<class** T**>**  
bool BTree**<**T**>::**isLeaf **(**BNode**<**T**>\*&** r**)** **const** **{**  
     **return** r **!=** **nullptr** **and** r **->** getLeft**()** **==** r **->** getRight**();**  
**}**  
  
**template** **<class** T**>**  
int BTree**<**T**>::**getHeight **(** **)** **{**  
     **return** getHeight**(**root**);**  
**}**  
  
**template** **<class** T**>**  
int BTree**<**T**>::**getHeight **(**BNode**<**T**>\*&** 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 BTree**<**T**>::**parsePreOrder **(** **)** **{**  
     parsePreOrder**(**root**);**  
**}**  
  
**template** **<class** T**>**  
void BTree**<**T**>::**parsePreOrder **(**BNode**<**T**>\*&** r**)** **{**  
       
     **if(**r **==** **nullptr){**  
          **return;**  
     **}**  
       
     std**::**cout **<<** r **->** getData**()** **<<** ", "**;**  
       
     parsePreOrder**(**r **->** getLeft**());**  
     parsePreOrder**(**r **->** getRight**());**  
**}**  
  
**template** **<class** T**>**  
void BTree**<**T**>::**parseInOrder **(** **)** **{**  
     parseInOrder**(**root**);**  
**}**  
  
**template** **<class** T**>**  
void BTree**<**T**>::**parseInOrder **(**BNode**<**T**>\*&** r**)** **{**  
       
     **if(**r **==** **nullptr){**  
          **return;**  
     **}**  
       
     parsePreOrder**(**r **->** getLeft**());**  
     std**::**cout **<<** r **->** getData**()** **<<** ", "**;**  
     parsePreOrder**(**r **->** getRight**());**  
**}**  
  
**template** **<class** T**>**  
void BTree**<**T**>::**parsePostOrder **(** **)** **{**  
     parsePostOrder**(**root**);**  
**}**  
**template** **<class** T**>**  
void BTree**<**T**>::**parsePostOrder **(**BNode**<**T**>\*&** r**)** **{**  
       
     **if(**r **==** **nullptr){**  
          **return;**  
     **}**  
       
     parsePreOrder**(**r **->** getLeft**());**  
     parsePreOrder**(**r **->** getRight**());**  
       
     std**::**cout **<<** r **->** getData**()** **<<** ", "**;**  
**}**  
  
**template** **<class** T**>**  
void BTree**<**T**>::**deleteAll **(** **)** **{**  
       
     deleteAll**(**root**);**  
     root **=** **nullptr;**  
**}**  
  
**template** **<class** T**>**  
void BTree**<**T**>::**deleteAll **(**BNode**<**T**>\*&** r**)** **{**  
       
     **if(**r **==** **nullptr){**  
          **return;**  
     **}**  
       
     deleteAll**(**r **->** getLeft**());**  
     deleteAll**(**r **->** getRight**());**  
       
     **delete** r**;**  
**}**  
  
**template** **<class** T**>**  
BTree**<**T**>&** BTree**<**T**>::operator** **=** **(const** BTree**&** t**)** **{**  
       
     deleteAll**();**  
     copyAll**(**t**);**  
       
     **return** **\*this;**  
**}**  
  
#endif

NODO DEL ARBOL BINARIO

ifndef BNODE\_H  
#define BNODE\_H  
  
#include <exception>  
#include <string>  
  
**template** **<class** T**>**  
**class** BNode**{**  
**private:**  
       
     T**\*** dataPtr**;**  
     BNode**\*** left**;**  
     BNode**\*** 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**();**  
          **}**  
            
     **};**  
       
     BNode**();**  
     BNode**(const** T**&);**  
       
     **~**BNode**();**  
       
     T**\*&** getDataPtr**()** **;**  
     T getData**()** **const;**  
     BNode**\*&** getLeft**();**  
     BNode**\*&** getRight**();**  
       
     void setDataPtr**(**T**\*&);**  
     void setData**(const** T**&);**  
     void setLeft**(**BNode**\*&);**  
     void setRight**(**BNode**\*&);**  
       
**};**  
  
**template** **<class** T**>**  
BNode**<**T**>::**BNode **(** **)** **:** dataPtr**(nullptr),** left**(nullptr),** right**(nullptr)** **{** **}**  
  
**template** **<class** T**>**  
BNode**<**T**>::**BNode **(const** T**&** e**)** **:** dataPtr**(new** T**(**e**)),** left**(nullptr),** right**(nullptr){**  
       
     **if(**dataPtr **==** **nullptr){**  
          **throw** Exception**(**"Memoria insuficiente, creando nodo."**);**  
     **}**  
**}**  
  
**template** **<class** T**>**  
BNode**<**T**>::~**BNode **(** **)** **{**  
     **delete** dataPtr**;**  
**}**  
  
**template** **<class** T**>**  
T**\*&** BNode**<**T**>::**getDataPtr **(** **)** **{**  
     **return** dataPtr**;**  
**}**  
  
**template** **<class** T**>**  
T BNode**<**T**>::**getData **(** **)** **const** **{**  
       
     **if(**dataPtr **==** **nullptr){**  
          **throw** Exception**(**"Dato Inexistente, getData."**);**  
     **}**  
       
     **return** **\***dataPtr**;**  
**}**  
  
**template** **<class** T**>**  
BNode**<**T**>\*&** BNode**<**T**>::**getLeft **(** **)** **{**  
     **return** left**;**  
**}**  
  
**template** **<class** T**>**  
BNode**<**T**>\*&** BNode**<**T**>::**getRight **(** **)** **{**  
     **return** right**;**  
**}**  
  
**template** **<class** T**>**  
void BNode**<**T**>::**setDataPtr **(**T**\*&** p**)** **{**  
     dataPtr **=** p**;**  
**}**  
  
**template** **<class** T**>**  
void BNode**<**T**>::**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 BNode**<**T**>::**setLeft **(**BNode**\*&** p**)** **{**  
     left **=** p**;**  
**}**  
  
**template** **<class** T**>**  
void BNode**<**T**>::**setRight **(**BNode**\*&** p**)** **{**  
     right **=** p**;**  
**}**  
  
#endif

CAPTURA DE PANTALLA

