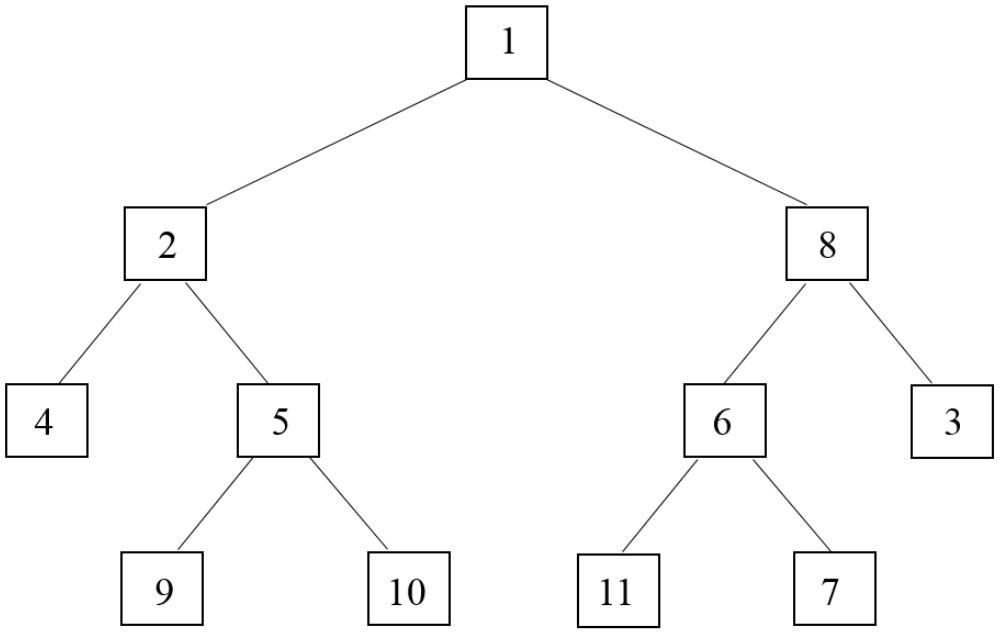
UNIVERSIDAD NEBRIJA

Escuela Politécnica Superior

Estructuras de datos y algoritmos

Práctica 3

Un árbol es un tipo abstracto de datos no lineal. El TAD árbol define una colección de elementos denominados nodos, donde uno de los nodos del árbol es el nodo raíz e impone una jerarquía entre los nodos.



El TAD árbol binario es una colección de nodos donde cada uno de ellos tiene como máximo dos descendientes, el hijo izquierdo y el hijo derecho.

La interfaz del TAD árbol binario.

```
inicializa(R, v, izquierdo, derecho) Crea el nodo raíz de R con el
           valor v e hijos izquierdo y derecho

  Precondición:  Ninguna
  Postcondición: Un árbol con raíz en el nodo R

vacio(R) Devuelve verdadero si el árbol R está vacío y falso en
         cualquier otro caso

  Precondición:  Ninguna
  Postcondición: true si el árbol R está vacío y false e.o.c.

insertaHoja(v) Crea un nuevo nodo con el valor v

  Precondición:  Ninguna
  Postcondición: La posición del nuevo nodo

insertaNodo(v, izquierdo, derecho) Crea un nuevo nodo con el valor v
            e hijos izquierdo y derecho

  Precondición:  Ninguna
  Postcondición: La posición del nuevo nodo

existe(R, v) Comprueba si el valor v está en el árbol R

  Precondición:  Ninguna
  Postcondición: true si el valor v está en el árbol R y
                 false e.o.c.

preorder(R) Recorre los nodos del árbol R en "preorder"

  Precondición:  Ninguna
  Postcondición: Ninguna

inorder(R) Recorre los nodos del árbol R en "inorder"

  Precondición:  Ninguna
  Postcondición: Ninguna

postorder(R) Recorre los nodos del árbol R en "postorder"

  Precondición:  Ninguna
  Postcondición: Ninguna
```

Implemente el TAD árbol binario utilizando las clases `NodoArbolBinario` y `ArbolBinario`.

```
class NodoArbolBinario {
public:

   int dato;
   NodoArbolBinario *izquierdo, *derecho;
};
```

La clase `ArbolBinario`.

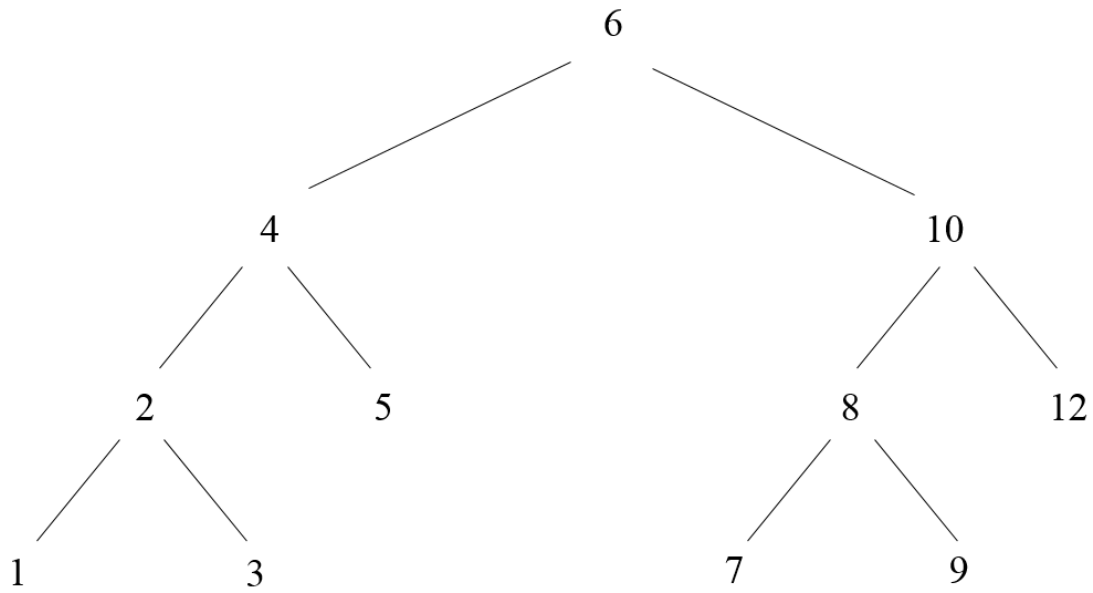
```
#include <string>
#include "NodoArbolBinario.hpp"

class ArbolBinario {

public:

   ArbolBinario(int v, NodoArbolBinario* izquierdo,
                NodoArbolBinario* derecho);
   ~ArbolBinario();

   bool vacio();
   static NodoArbolBinario* insertaHoja(int v);
   static NodoArbolBinario* insertaNodo(int v,
                            NodoArbolBinario* izquierdo,
                            NodoArbolBinario* derecho);
   bool existe(int v);

   std::string imprime();
   std::string preorder();
   std::string inorder();
   std::string postorder();

private:

   NodoArbolBinario* raiz;

   bool existe(NodoArbolBinario* r, int v);

   std::string preorder(NodoArbolBinario* r);
   std::string inorder(NodoArbolBinario* r);
   std::string postorder(NodoArbolBinario* r);

};
```

El programa de prueba del TAD árbol binario.

```
#include <iostream>
#include "ArbolBinario.hpp"

int main() {

   NodoArbolBinario* n1 = ArbolBinario::insertaHoja(20);
   NodoArbolBinario* n2 = ArbolBinario::insertaHoja(10);
   NodoArbolBinario* n3 = ArbolBinario::insertaNodo(30, n1, n2);

   NodoArbolBinario* m1 = ArbolBinario::insertaHoja(40);
   NodoArbolBinario* m2 = ArbolBinario::insertaHoja(60);
   NodoArbolBinario* m3 = ArbolBinario::insertaNodo(100, m1, m2);

   ArbolBinario ab = ArbolBinario(5, n3, m3);

   std:: cout << ab.imprime();

   std::cout << "\nexiste(20) = " << ab.existe(20) <<
                ", existe(15) = " << ab.existe(15) << "\n";

   return 0;

}
```

Un árbol binario de búsqueda (Binary Search Tree) es un árbol binario en el que para cualquier nodo se cumplen las siguientes condiciones:

- El hijo izquierdo, si no es nulo, almacena un valor menor que el del nodo padre
- El hijo derecho, si no es nulo, almacena un valor mayor que el del nodo padre



La interfaz del TAD árbol binario de búsqueda.

```
inicializa(R) Inicializa R a un árbol vacío

  Precondición:  Ninguna
  Postcondición: Un árbol binario de búsqueda vacío

vacio(R) Devuelve verdadero si el árbol R está vacío y falso en
         cualquier otro caso

  Precondición:  Ninguna
  Postcondición: true si el árbol R está vacío y false e.o.c.

inserta(R, v) Inserta un nodo con el valor v

  Precondición:  Ninguna
  Postcondición: El nodo con el valor v ocupa la posición que
                 le corresponde en el árbol R

existe(R, v) Comprueba si el valor v está en el árbol R

  Precondición:  Ninguna
  Postcondición: true si v está en el árbol R y false e.o.c.

elimina(R, v) Elimina el nodo de R que almacena el valor v

  Precondición:  Ninguna
  Postcondición: El árbol R ya no tiene un nodo con el valor v

preorder(R) Recorre los nodos del árbol R en "preorder"

  Precondición:  Ninguna
  Postcondición: Ninguna

inorder(R) Recorre los nodos del árbol R en "inorder"
  Precondición:  Ninguna
  Postcondición: Ninguna

postorder(R) Recorre los nodos del árbol R en "postorder"
  Precondición:  Ninguna
  Postcondición: Ninguna
```

Implemente el TAD árbol binario de búsqueda utilizando las clases `NodoArbolBinarioBusqueda` y `ArbolBinarioBusqueda`.

```
class NodoArbolBinarioBusqueda {
public:

   int dato;
   NodoArbolBinarioBusqueda *izquierdo, *derecho;
};
```

La clase `ArbolBinarioBusqueda`.

```
#include <string>
#include "NodoArbolBinarioBusqueda.hpp"

class ArbolBinarioBusqueda {

public:

   ArbolBinarioBusqueda();
   ~ArbolBinarioBusqueda();

   bool vacio();
   void inserta(int v);
   bool existe(int v);

   std::string imprime();
   std::string preorder();
   std::string inorder();
   std::string postorder();

private:

   NodoArbolBinarioBusqueda* raiz;

   void inserta(NodoArbolBinarioBusqueda*& r, int v);
   NodoArbolBinarioBusqueda* insertaNodo(int v);
   bool existe(NodoArbolBinarioBusqueda* r, int v);

   std::string preorder(NodoArbolBinarioBusqueda* r);
   std::string inorder(NodoArbolBinarioBusqueda* r);
   std::string postorder(NodoArbolBinarioBusqueda* r);

};
```

El programa de prueba del TAD árbol binario de búsqueda.

```
#include <iostream>
#include "ArbolBinarioBusqueda.hpp"

int main() {

   std::cout << "Arbol binario de busqueda \n";

   ArbolBinarioBusqueda abb = ArbolBinarioBusqueda();

   abb.inserta(10);
   abb.inserta(15);
   abb.inserta(9);
   abb.inserta(5);
   abb.inserta(7);
   abb.inserta(1);
   abb.inserta(12);

   std::cout << abb.imprime();

   std::cout << "\n";
   std::cout << "\existe(12) = " << abb.existe(12) <<
                ", existe(25) = " << abb.existe(25) << "\n";
   std::cout << "\n";

   return 0;
}
```