Estructuras de Datos.
Grado en Ingeniería Informática
Doble Grado en Ingeniería Informática y Matemáticas
Ejercicio

1. Introducción

El ejercicio consiste en implementar la clase **Arbol General** y usarla para implementar algunas funciones concretas.

2. Representación del tipo *Arbol General*

Se usará como representación del árbol general la del bf hijo-mas-a-la-izquierda/hermano-a-la-derecha. A continuación se da una breve descripción de lo que podría ser la clase, aunque se tiene libertad para añadir o quitar las funciones que se estimen oportunas, o incluso diseñar la clase de otra forma.

2.1. Interfaz: *arbolgeneral.h*

```
#ifndef __ArbolGeneral_h__
#define __ArbolGeneral_h__

/**
   @memo T.D.A. ArbolGeneral

   @doc
   {\bf Definición:}
   Una instancia $a$ del tipo de dato abstracto {\em ArbolGeneral} sobre un
   dominio {\em Tbase} se puede construir como

   \begin{itemize}
   \item Un objeto vacío (árbol vacío) si no contiene ningún elemento. Lo
   denotamos \{\}
   \item Un árbol que contiene un elemento destacado, el nodo raíz, con un
   valor {\em e} en el dominio {\em Tbase} (denominado {\em etiqueta}), y $k$
   subárboles ($T_1$ $\ldots$ $T_k$) del T.D.A. {\em ArbolGeneral}
   sobre  {\em Tbase}.
   Se establece una relación {\em padre-hijomasalaizquierda-hermanoaladerecha} entre cada nodo
   y los nodos raíz  de los subárboles (si los hubiera) que cuelgan de él.
   \end{itemize}

   Para poder usar el tipo de dato {\em ArbolGeneral} se debe incluir el
   fichero

   \begin{center}
   {\em ArbolGeneral.h}
   \end{center}

   El espacio requerido para el almacenamiento es O(n). Donde n es el
   número de nodos del árbol.

   @author
```

```
   @version 1.0

*/
template <class Tbase>
class ArbolGeneral {
  private:
    /** @name Implementación de T.D.A. ArbolGeneral
        @memo Parte privada.
    */
    /*@{*/
      /**
       @memo Nodo
       @doc En cada  estructura {\em nodo} se almacena una etiqueta del
       árbol, que se implementa como un conjunto de nodos enlazados
       según la relación padre-hijo-hermano
      */
      struct nodo {
        /**
   @memo Elemento almacenado
   @doc En este campo se almacena la etiqueta que corresponde
    a este nodo

*/
        Tbase etiqueta;
/**
   @memo Puntero al hijo más a la izquierda
   @doc En este campo se almacena un puntero al nodo raíz del
   subárbol más a la izquierda,  o el valor 0 si no tiene.

*/
        struct nodo *izqda;
/**
   @memo Puntero al hermano derecho
   @doc En este campo se almacena un puntero al nodo raíz del
   subárbol hermano derecho,  o el valor 0 si no tiene.

*/
        struct nodo *drcha;
/**
   @memo Puntero al padre
   @doc En este campo se almacena un puntero al nodo padre,
   o el valor 0 si es la raiz.

*/
        struct nodo *padre;
      };
      /**
 @memo Puntero a la raiz.
 @doc Este miembro es un puntero al primer nodo, que corresponde
 a la raíz del árbol. Vale cero sin el árbol es vacío

      */
      struct nodo *laraiz;
      /** @name Invariante de la representación
```

```
 @memo Inv. de ArbolGeneral
         @doc
   Añadir el Invariante


     */
     /** @name Función de abstracción
 @memo F.A. de ArbolGeneral.
 @doc
   Añadir la función

     */
     /** */
     /**
@memo Destruye el subárbol
@param n: nodo que destruir junto con sus descendientes
@doc
  libera los recursos que ocupa {\em n} y sus descendientes.

     */
     void destruir(nodo * n);
     /**
@memo Copia un subárbol
@param dest: referencia al puntero del que cuelga la copia
@param orig: puntero a la raiz del subárbol a copiar
@doc
  Hace una copia de todo el subárbol que cuelga de {\em orig} en el
  puntero {\em dest}. Es importante ver que en {\em dest->padre}
  (si existe) no se asigna ningún valor, pues no se conoce.

     */
     void copiar(nodo *& dest, nodo * orig);
     /**
@memo Cuenta el número de nodos
@param n: nodo del que cuelga el subárbol de nodos a contabilizar.
@doc
  Copia cuántos nodos cuelgan de {\em n}, incluído éste.

     */
     void contar(nodo * n);
     /**
@memo Comprueba igualdad de subárbol
@param n1: Primer subárbol a comparar
@param n2: Segundo subárbol a comparar
@doc
  Comprueba si son iguales los subárboles que cuelgan de {\em n1}
  y {\em n2}. Para ello deberán tener los mismos nodos en las
  mismas posiciones y con las mismas etiquetas.

     */
     bool soniguales(nodo * n1, nodo * n2);
     /**
@memo Escribe un subárbol
@param out: stream de salida donde escribir
```

```
@param nod: nodo del que cuelga el subárbol a escribir
@doc
  Escribe en la salida todos los nodos del subárbol que cuelga
  del nodo {\em nod} siguiendo un recorrido en preorden. La forma
  de impresión de cada nodo es:
  \begin{itemize}
  \item Si el nodo es {\em nodo_nulo}, imprime el carácter 'x'
  \item Si el nodo no es {\em nodo_nulo}, imprime el carácter 'n'
  seguido de un espacio, al que sigue la impresión de la etiqueta.
  \end{itemize}

     */
     void escribe_arbol(std::ostream& out, nodo * nod) const;
     /**
@memo Lee un subárbol
@param in: stream de entrada desde el que leer
@param nod: referencia al nodo que contendrá el subárbol leído
@doc
  Lee de la entrada {\em in} los elementos de un árbol según el
  formato que se presenta en la función de escritura.
  @see escribe_arbol

     */
     void lee_arbol(std::istream& in, nodo *& nod);

   /*@}*/
 public:
   /**
      @memo Tipo Nodo
      @doc
        Este tipo nos permite manejar cada uno de los nodos del árbol.
Los valores que tomará serán tantos como nodos en el árbol (para
poder referirse a cada uno de ellos) y además un valor destacado
{\em nodo_nulo}, que indica que no se refiere a ninguno de ellos.

Una variable {\em n} de este tipo se declara

\begin{center}
{\em ArbolGeneral::Nodo n};
\end{center}

Las operaciones válidas sobre el tipo nodo son:

- Operador de Asignación (=)
- Operador de comprobación de igualdad (==).
- Operador de comprobación de no igualdad (!=).

   */
     typedef struct nodo * Nodo;
   /** */
   /**
      @memo Nodo nulo
      @doc
```

```
      El valor de nodo nulo se podrá indicar como

{\em ArbolGeneral::nodo_nulo}

   */
     // static const Nodo nodo_nulo = 0;
   /** */
   /** @name Operaciones de T.D.A. árbol General
       @memo Operaciones sobre ArbolGeneral
   */
   /*@{*/
     /**
        @memo Constructor por defecto
        @doc
          Reserva los recursos e iniciliza el árbol a vacío \{\}. La
  operación se realiza en tiempo O(1).

     */
     ArbolGeneral();
     /**
        @memo Constructor de raiz
        @doc
          Reserva los recursos e iniciliza el árbol con  un único
  nodo raiz que tiene la etiqueta {\em e}, es decir, el árbol
  \{e,\{\},\{\}\}. La operación se realiza en tiempo O(1).

     */
     ArbolGeneral(const Tbase& e);
     /**
        @memo Constructor de copias
@param v: ArbolGeneral a copiar
        @doc
  Construye el árbol duplicando el contenido
  de {\em v} en el árbol receptor.
  La operación se realiza en tiempo O(n), donde {\em n} es el número
  de elementos de {\em v}.

     */
     ArbolGeneral (const ArbolGeneral<Tbase>& v);
     /**
@memo Destructor
@doc
  Libera los recursos ocupados por el árbol receptor. La operación
  se realiza en tiempo O(n) donde {\em n} es el número de
  elementos del árbol receptor.

     */
     ~ArbolGeneral();
     /**
@memo Asignación
@param v: ArbolGeneral a copiar
@return Referencia al árbol receptor.
@doc
  Asigna el valor del árbol duplicando el contenido
```

```
  de {\em v} en el árbol receptor.
  La operación se realiza en tiempo O(n), donde {\em n} es el número
  de elementos de {\em v}.

     */
     ArbolGeneral<Tbase>& operator=(const ArbolGeneral<Tbase> &v);
     /**
@memo Asignar nodo raíz
@param e: etiqueta a asignar al nodo raíz
@doc
  Vacía el árbol receptor y le asigna como valor el árbol de un
  único nodo cuya etiqueta es {\em e}.

     */
     void AsignaRaiz(const Tbase& e);
     /**
@memo Raíz del árbol
@return Nodo raíz del árbol receptor
@doc Devuelve el nodo raíz, que coincide con
{\em nodo_nulo}  si el árbol está vacío.
La operación se realiza en tiempo O(1).

     */
     Nodo raiz() const;
     /**
@memo Hijo más a la izquierda
@param n: nodo del que se quiere obtener el hijo más a la izquierda.
  {\em n} no es {\em nodo_nulo}
@return Nodo hijo más a la izquierda
@doc Devuelve el nodo hijo más a la izquierda de {\em n}, que valdrá
{\em nodo_nulo} si no tiene hijo más a la izquierda.
La operación se realiza en tiempo O(1).

     */
     Nodo hijomasizquierda(const Nodo n) const;
     /**
@memo Hermano derecha
@param n: nodo del que se quiere obtener el hermano a la derecha.
  {\em n} no es {\em nodo_nulo}
@return Nodo hermano a la derecha
@doc Devuelve el nodo hermano a la derecha de {\em n}, que valdrá
{\em nodo_nulo} si no tiene hermano a la derecha.
La operación se realiza en tiempo O(1).

     */
     Nodo hermanoderecha(const Nodo n) const;
     /**
@memo Nodo padre
@param n: nodo del que se quiere obtener el padre. {\em n} no
     es {\em nodo_nulo}
@return Nodo padre
@doc Devuelve el nodo padre de {\em n}, que valdrá
{\em nodo_nulo} si es la raíz.
La operación se realiza en tiempo O(1).
```

```
     */
     Nodo padre(const Nodo n) const;
     /**
@memo Etiqueta en un nodo
@param n: nodo en el que se encuentra el elemento. {\em n} no
     es {\em nodo_nulo}
@return Referencia al elemento del nodo {\em n}
@doc Devuelve una referencia al elemento del nodo {\em n} y
por tanto se puede modificiar o usar el valor.
La operación se realiza en tiempo O(1).

     */
     Tbase& etiqueta(const Nodo n);
     /**
@memo Etiqueta en un nodo
@param n: nodo en el que se encuentra el elemento. {\em n} no
     es {\em nodo_nulo}
@return Referencia constante al elemento del nodo {\em n}.
@doc Devuelve una referencia al elemento del nodo {\em n}. Es
constante y por tanto no se puede modificiar el valor.
La operación se realiza en tiempo O(1).

     */
     const Tbase& etiqueta(const Nodo n) const;
     /**
@memo Copia Subárbol
@param orig: árbol desde el que se va a copiar una rama
@param nod: nodo raíz del subárbol que se copia. Es un nodo
          del árbol {\em orig} y no es {\em nodo_nulo}
@doc
  El árbol receptor acaba con un valor copia del subárbol que cuelga
  del nodo {\em nod} en el árbol {\em orig}. La operación se realiza
  en tiempo O(n) donde {\em n} es el número de nodos del subárbol
  copiado.

     */
     void asignar_subarbol(const ArbolGeneral<Tbase>& orig, const Nodo nod);
     /**
@memo Podar subárbol más a la izquierda
@param n: Nodo al que se le podará la rama hijo más a la izquierda. No es
  {\em nodo_nulo} y es un nodo válido del árbol receptor.
@param dest: árbol que recibe la rama cortada
@doc
  Asigna un nuevo valor al árbol {\em dest}, con todos los elementos
del subárbol izquierdo del nodo {\em n} en el árbol receptor.
éste se queda sin dichos nodos.
La operación se realiza en tiempo O(1).

     */
     void podar_hijomasizquierda(Nodo n, ArbolGeneral<Tbase>& dest);
     /**
@memo Podar subárbol hermano derecha
@param n: Nodo al que se le podará la rama hermano derecha. No es
```

```
  {\em nodo_nulo} y es un nodo válido del árbol receptor.
@param dest: árbol que recibe la rama cortada
@doc
  Asigna un nuevo valor al árbol {\em dest}, con todos los elementos
del subárbol hermano derecho del nodo {\em n} en el árbol receptor.
éste se queda sin dichos nodos.
La operación se realiza en tiempo O(1).

     */
     void podar_hermanoderecha(Nodo n, ArbolGeneral<Tbase>& dest);
     /**
@memo Insertar subárbol izquierda
@param n: Nodo al que se insertará el árbol {\em rama} como
  hijo más a la izquierda. No es
  {\em nodo_nulo} y es un nodo válido del árbol receptor
@param rama: árbol que se insertará como hijo más a la izquierda.
@doc
  El árbol {\em rama} se inserta como hijo más a la izquierda del nodo
  {\em n} del árbol receptor. El árbol {\em rama} queda vacío
  y los nodos que estaban en el subárbol hijo más a la izquierda de {\em n}
  se desplazan a la derecha, de forma que el anterior hijo más a la izquierda pasa a ser el hermano

     */
     void insertar_hijomasizquierda(Nodo n, ArbolGeneral<Tbase>& rama);
     /**
@memo Insertar subárbol derecha
@param n: Nodo al que se insertará el árbol {\em rama} como
  hermano a la derecha. No es
  {\em nodo_nulo} y es un nodo válido del árbol receptor
@param rama: árbol que se insertará como hermano derecho.
@doc
  El árbol {\em rama} se inserta como hermano derecho del nodo
  {\em n} del árbol receptor. El árbol {\em rama} queda vacío
  y los nodos que estaban a la derecha del nodo n pasan a la derecha del nuevo nodo.

     */
     void insertar_hermanoderecha(Nodo n, ArbolGeneral<Tbase>& rama);
     /**
@memo Borra todos los elementos
@doc
  Borra todos los elementos del árbol receptor. Cuando termina,
  el árbol está vacío. La operación se realiza
  en tiempo O(n), donde {\em n} es el número
  de elementos del árbol receptor.

     */
     void clear();
     /**
@memo Número de elementos
@return
  El número de elementos del árbol receptor.
@doc
  La operación se realiza en tiempo O(n).
@see contar
```

```
     */
     int size() const;
     /**
@memo Vacío
@return
  Devuelve {\em true} si el número de elementos del árbol
  receptor es cero, {\em false}
  en otro caso.
@doc La operación se realiza en tiempo O(1).

     */
     bool empty() const;
     /**
@memo Igualdad
@param v: ArbolGeneral con la que se desea comparar.
@return
  Devuelve {\em true} si el árbol receptor tiene los mismos elementos
  y en el mismo orden. {\em false} en caso contrario.
@doc La operación se realiza en tiempo O(n).
@see soniguales

     */
     bool operator==(const ArbolGeneral<Tbase>& v) const;
     /**
@memo Distintos
@param v: ArbolGeneral con la que se desea comparar.
@return
  Devuelve {\em true} si el árbol receptor no tiene los mismos elementos
  y en el mismo orden. {\em false} en caso contrario.
@doc La operación se realiza en tiempo O(n).

     */
     bool operator!=(const ArbolGeneral<Tbase>& v) const;

     /**
@memo Entrada
@name operator>>
@param in: stream de entrada
@param v: árbol que leer
@return referencia al stream de entrada
@doc
  Lee de {\em in} un árbol y lo almacena en {\em v}. El formato
  aceptado para la lectura se puede consultar en la función
  de salida.
@see lee_arbol

     *//** */
     template<class T>
     friend std::istream& operator>>(std::istream& in, ArbolGeneral<T>& v);
     /**
@memo Salida
@name operator<<
@param out: stream de salida
```

```
 @param v: árbol que escribir
 @return referencia al stream de salida
 @doc
   Escribe en la salida todos los nodos del árbol {\em v} siguiendo
   un recorrido en preorden. La forma de impresión de cada nodo
   es:
   \begin{itemize}
   \item Si el nodo es {\em nodo_nulo}, imprime el carácter 'x'
   \item Si el nodo no es {\em nodo_nulo}, imprime el carácter 'n'
   seguido de un espacio, al que sigue la impresión de la etiqueta.
   \end{itemize}
 @see escribe_arbol

      *//** */
      template<class T>
      friend std::ostream& operator<< (std::ostream& out, const ArbolGeneral<T>& v);
    /*@}*/
};

#endif
```

Práctica a realizar

En este ejercicio se propone la implementación de la **clase arbolgeneral** y su aplicación en la implementación de las siguientes funciones:

- Implementar los recorridos preorden, inorden, postorden y por niveles y calcular la altura del árbol
- Construir una función para determinar un árbol general a partir de sus 3 recorridos.
- Construir una función para reflejar un árbol general, entendiendo por reflejado de un árbol general aquel que se obtiene con la misma raiz y permutando de orden todos sus hijos, haciendo lo mismo de forma recursiva en todos los niveles.

Como implementación adicional se propone:

- Construir una clase iteradora preorden