# P6 Estructuras de Datos

Generated by Doxygen 1.8.8

Wed Feb 4 2015 10:41:26

# Contents


**1 Todo List** **1**

**2 Class Index** **3**

2.1 Class List . . . . . 3

**3 File Index** **5**

3.1 File List . . . . . 5

**4 Class Documentation** **7**

4.1 tree< T >::const_inorderiterator Class Reference . . . . . 7

4.1.1 Constructor & Destructor Documentation . . . . . 7

4.1.1.1 const_inorderiterator . . . . . 7

4.1.1.2 const_inorderiterator . . . . . 7

4.1.2 Member Function Documentation . . . . . 7

4.1.2.1 operator"!= . . . . . 7

4.1.2.2 operator* . . . . . 7

4.1.2.3 operator++ . . . . . 7

4.1.2.4 operator== . . . . . 7

4.2 gs1Set::const_iterator Class Reference . . . . . 8

4.2.1 Detailed Description . . . . . 8

4.2.2 Constructor & Destructor Documentation . . . . . 8

4.2.2.1 const_iterator . . . . . 8

4.2.2.2 const_iterator . . . . . 8

4.2.3 Member Function Documentation . . . . . 9

4.2.3.1 operator"!= . . . . . 9

4.2.3.2 operator* . . . . . 9

4.2.3.3 operator++ . . . . . 9

4.2.3.4 operator++ . . . . . 9

4.2.3.5 operator= . . . . . 9

4.2.3.6 operator== . . . . . 9

4.2.3.7 upper_IA . . . . . 9

4.2.4 Friends And Related Function Documentation . . . . . 9

4.2.4.1 gs1Set . . . 9
4.3 tree< T >::const_leveliterator Class Reference . . . 9
4.3.1 Constructor & Destructor Documentation . . . 10
4.3.1.1 const_leveliterator . . . 10
4.3.1.2 const_leveliterator . . . 10
4.3.2 Member Function Documentation . . . 10
4.3.2.1 operator"!= . . . 10
4.3.2.2 operator* . . . 10
4.3.2.3 operator++ . . . 10
4.3.2.4 operator== . . . 10
4.4 tree< T >::const_node Class Reference . . . 10
4.4.1 Constructor & Destructor Documentation . . . 11
4.4.1.1 const_node . . . 11
4.4.1.2 const_node . . . 11
4.4.1.3 const_node . . . 11
4.4.2 Member Function Documentation . . . 11
4.4.2.1 left . . . 11
4.4.2.2 next_sibling . . . 11
4.4.2.3 null . . . 11
4.4.2.4 operator"!= . . . 11
4.4.2.5 operator* . . . 12
4.4.2.6 operator= . . . 12
4.4.2.7 operator== . . . 12
4.4.2.8 parent . . . 12
4.4.3 Friends And Related Function Documentation . . . 12
4.4.3.1 tree< T > . . . 12
4.5 tree< T >::const_postorderiterator Class Reference . . . 12
4.5.1 Constructor & Destructor Documentation . . . 13
4.5.1.1 const_postorderiterator . . . 13
4.5.1.2 const_postorderiterator . . . 13
4.5.2 Member Function Documentation . . . 13
4.5.2.1 operator"!= . . . 13
4.5.2.2 operator* . . . 13
4.5.2.3 operator++ . . . 13
4.5.2.4 operator== . . . 13
4.6 tree< T >::const_preorderiterator Class Reference . . . 13
4.6.1 Constructor & Destructor Documentation . . . 13
4.6.1.1 const_preorderiterator . . . 13
4.6.1.2 const_preorderiterator . . . 13
4.6.2 Member Function Documentation . . . 13

4.6.2.1 operator"!= . . . 13
4.6.2.2 operator∗ . . . 13
4.6.2.3 operator++ . . . 13
4.6.2.4 operator== . . . 14
4.7 gs1Set Class Reference . . . 14
4.7.1 Detailed Description . . . 15
4.7.2 Member Typedef Documentation . . . 16
4.7.2.1 size_type . . . 16
4.7.3 Constructor & Destructor Documentation . . . 16
4.7.3.1 gs1Set . . . 16
4.7.3.2 gs1Set . . . 16
4.7.4 Member Function Documentation . . . 16
4.7.4.1 begin . . . 16
4.7.4.2 codesWithPrefix . . . 16
4.7.4.3 empty . . . 16
4.7.4.4 end . . . 16
4.7.4.5 erase . . . 16
4.7.4.6 find . . . 17
4.7.4.7 insert . . . 17
4.7.4.8 operator= . . . 17
4.7.4.9 print . . . 18
4.7.4.10 reading_gs1Set . . . 18
4.7.4.11 size . . . 18
4.8 tree< T >::inorderiterator Class Reference . . . 18
4.8.1 Detailed Description . . . 18
4.8.2 Constructor & Destructor Documentation . . . 19
4.8.2.1 inorderiterator . . . 19
4.8.2.2 inorderiterator . . . 19
4.8.3 Member Function Documentation . . . 19
4.8.3.1 operator"!= . . . 19
4.8.3.2 operator∗ . . . 19
4.8.3.3 operator++ . . . 19
4.8.3.4 operator== . . . 19
4.9 tree< T >::leveliterator Class Reference . . . 19
4.9.1 Detailed Description . . . 19
4.9.2 Constructor & Destructor Documentation . . . 19
4.9.2.1 leveliterator . . . 19
4.9.2.2 leveliterator . . . 19
4.9.3 Member Function Documentation . . . 19
4.9.3.1 operator"!= . . . 19

4.9.3.2 operator* . . . 20
4.9.3.3 operator++ . . . 20
4.9.3.4 operator== . . . 20
4.10 tree< T >::node Class Reference . . . 20
4.10.1 Detailed Description . . . 21
4.10.2 Constructor & Destructor Documentation . . . 21
4.10.2.1 node . . . 21
4.10.2.2 node . . . 21
4.10.3 Member Function Documentation . . . 21
4.10.3.1 left . . . 21
4.10.3.2 next_sibling . . . 21
4.10.3.3 null . . . 21
4.10.3.4 operator"!= . . . 21
4.10.3.5 operator* . . . 22
4.10.3.6 operator* . . . 22
4.10.3.7 operator= . . . 22
4.10.3.8 operator== . . . 22
4.10.3.9 parent . . . 22
4.10.3.10 setlabel . . . 22
4.10.4 Friends And Related Function Documentation . . . 22
4.10.4.1 tree< T > . . . 22
4.11 tree< T >::postorderiterator Class Reference . . . 23
4.11.1 Detailed Description . . . 23
4.11.2 Constructor & Destructor Documentation . . . 23
4.11.2.1 postorderiterator . . . 23
4.11.2.2 postorderiterator . . . 23
4.11.3 Member Function Documentation . . . 23
4.11.3.1 operator"!= . . . 23
4.11.3.2 operator* . . . 23
4.11.3.3 operator++ . . . 23
4.11.3.4 operator== . . . 23
4.12 tree< T >::preorderiterator Class Reference . . . 23
4.12.1 Detailed Description . . . 24
4.12.2 Constructor & Destructor Documentation . . . 24
4.12.2.1 preorderiterator . . . 24
4.12.2.2 preorderiterator . . . 24
4.12.3 Member Function Documentation . . . 24
4.12.3.1 operator"!= . . . 24
4.12.3.2 operator* . . . 24
4.12.3.3 operator++ . . . 24

4.12.3.4 operator++ . . . 24
4.12.3.5 operator== . . . 24
4.13 tree< T > Class Template Reference . . . 24
4.13.1 Detailed Description . . . 26
4.13.2 Member Typedef Documentation . . . 26
4.13.2.1 size_type . . . 26
4.13.3 Constructor & Destructor Documentation . . . 26
4.13.3.1 tree . . . 26
4.13.3.2 tree . . . 27
4.13.3.3 tree . . . 28
4.13.3.4 ∼tree . . . 28
4.13.4 Member Function Documentation . . . 28
4.13.4.1 assign_subtree . . . 28
4.13.4.2 beginInorder . . . 28
4.13.4.3 beginInorder . . . 28
4.13.4.4 beginlevel . . . 28
4.13.4.5 beginlevel . . . 28
4.13.4.6 beginPostorder . . . 28
4.13.4.7 beginPostorder . . . 28
4.13.4.8 beginPreorder . . . 28
4.13.4.9 beginPreorder . . . 28
4.13.4.10 clear . . . 28
4.13.4.11 empty . . . 29
4.13.4.12 endInorder . . . 29
4.13.4.13 endInorder . . . 29
4.13.4.14 endlevel . . . 29
4.13.4.15 endlevel . . . 29
4.13.4.16 endPostorder . . . 29
4.13.4.17 endPostorder . . . 29
4.13.4.18 endPreorder . . . 29
4.13.4.19 endPreorder . . . 29
4.13.4.20 insert_left . . . 29
4.13.4.21 insert_left . . . 29
4.13.4.22 insert_right_sibling . . . 29
4.13.4.23 insert_right_sibling . . . 30
4.13.4.24 is_external . . . 30
4.13.4.25 is_internal . . . 30
4.13.4.26 is_root . . . 30
4.13.4.27 null . . . 31
4.13.4.28 operator"!= . . . 31

4.13.4.29 operator= . . . 31
4.13.4.30 operator== . . . 31
4.13.4.31 prune_left . . . 31
4.13.4.32 prune_right_sibling . . . 31
4.13.4.33 root . . . 32
4.13.4.34 setroot . . . 32
4.13.4.35 size . . . 32

**5 File Documentation** **33**

5.1 generadorCodigos.cpp File Reference . . . 33
5.1.1 Function Documentation . . . 33
5.1.1.1 main . . . 33
5.2 gs1Set.cpp File Reference . . . 33
5.3 gs1Set.h File Reference . . . 33
5.4 nodetree.hxx File Reference . . . 34
5.5 prueba_gs1.cpp File Reference . . . 34
5.5.1 Function Documentation . . . 34
5.5.1.1 load . . . 34
5.5.1.2 main . . . 34
5.6 tree.h File Reference . . . 34
5.6.1 Detailed Description . . . 35
5.7 tree.hxx File Reference . . . 35

# Chapter 1

# Todo List

**Class gs1Set (**p. **14)**

Tareas a realizar: El alumno debera implementar la clase **gs1Set** (p. 14) , junto con el codigo de prueba de los distintos metodos.

**Member gs1Set::reading_gs1Set (**p. **18) ()**

implementar este metodo correctamente OJO ESTE METODO OS SIRVE PARA PODER CONSTRUIR UN ARBOL NO TIENE EN CUENTA EL INVARIANTE DE LA REPRESENTACION AL NO CONSIDERAR EL CAMPO INT DEL NODO!!!!! DEBEIS MODIFICARLO PARA QUE LO HAGA DE FORMA CORRECTA

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:


**tree< T >::const_inorderiterator** . . . . . 7
**gs1Set::const_iterator** . . . . . 8
**tree< T >::const_leveliterator** . . . . . 9
**tree< T >::const_node** . . . . . 10
**tree< T >::const_postorderiterator** . . . . . 12
**tree< T >::const_preorderiterator** . . . . . 13
**gs1Set** . . . . . 14
**tree< T >::inorderiterator** . . . . . 18
**tree< T >::leveliterator** . . . . . 19
**tree< T >::node** . . . . . 20
**tree< T >::postorderiterator** . . . . . 23
**tree< T >::preorderiterator** . . . . . 23
**tree< T >** . . . . . 24

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

**generadorCodigos.cpp** . . . . . 33
**gs1Set.cpp** . . . . . 33
**gs1Set.h** . . . . . 33
**nodetree.hxx** . . . . . 34
**prueba_gs1.cpp** . . . . . 34
**tree.h**
TDA tree . . . . . 34
**tree.hxx** . . . . . 35

# Chapter 4

# Class Documentation

## 4.1 tree< T >::const_inorderiterator Class Reference

`#include <tree.h>`

### Public Member Functions

- **const_inorderiterator** ()
- **const_inorderiterator** (**node** n)
- bool **operator!=** (const **const_inorderiterator** &i) const
- bool **operator==** (const **const_inorderiterator** &i) const
- const T & **operator*** () const
- **const_inorderiterator** & **operator++** ()

### 4.1.1 Constructor & Destructor Documentation

#### 4.1.1.1 template<typename T > tree< T >::const_inorderiterator::const_inorderiterator ( ) [inline]

#### 4.1.1.2 template<typename T> tree< T >::const_inorderiterator::const_inorderiterator ( node *n* )

### 4.1.2 Member Function Documentation

#### 4.1.2.1 template<typename T> bool tree< T >::const_inorderiterator::operator!= ( const const_inorderiterator & *i* ) const [inline]

#### 4.1.2.2 template<typename T > const T & tree< T >::const_inorderiterator::operator* ( ) const [inline]

#### 4.1.2.3 template<typename T > tree< T >::const_inorderiterator & tree< T >::const_inorderiterator::operator++ ( )

#### 4.1.2.4 template<typename T> bool tree< T >::const_inorderiterator::operator== ( const const_inorderiterator & *i* ) const [inline]

The documentation for this class was generated from the following files:

- **tree.h**
- **tree.hxx**

## 4.2 gs1Set::const_iterator Class Reference

`#include <gs1Set.h>`

### Public Member Functions

- **const_iterator** ()

  *Constructor primitivo.*
- **const_iterator** (const **const_iterator** &it)

  *Constructor de copia.*
- string **operator*** ()

  *devuelve el codigo completo al que apunta el iterador. Nota: Se encuentra en el camino que hay desde el nodo hacia la raiz.*
- **const_iterator** & **operator++** ()

  *avanza hacia el siguiente final de (sub)codigo en preorden en el **gs1Set** (p. 14) . Nota: Avanza por el arbol hasta el siguiente nodo que es final de (sub)codigo.*
- **const_iterator operator++** (int)

  *avanza hacia el siguiente final de (sub)codigo en preorden en el **gs1Set** (p. 14). Nota: Avanza por el arbol hasta el siguiente nodo que es final de (sub)codigo.*
- **const_iterator** & **upper_IA** ()

  *sube hacia el identificador de aplicacion (IA) anterior en el codigo.*
- **const_iterator** & **operator=** (const **const_iterator** &it)
- bool **operator==** (const **const_iterator** &it) const
- bool **operator!=** (const **const_iterator** &it) const

### Friends

- class **gs1Set**

### 4.2.1 Detailed Description

**const_iterator** (p. 8): **const_iterator** (p. 8), operator*, operator++

Descripcion

Un objeto de la clase **gs1Set::const_iterator** (p. 8) representara un iterador sobre el conjunto de codigos en el **gs1Set** (p. 14) .

Nota: Solo itera sobre (subcodigos) y no sobre los elementos individuales

### 4.2.2 Constructor & Destructor Documentation

#### 4.2.2.1 gs1Set::const_iterator::const_iterator ( )

Constructor primitivo.

#### 4.2.2.2 gs1Set::const_iterator::const_iterator ( const **const_iterator** & *it* )

Constructor de copia.

### 4.2.3 Member Function Documentation

#### 4.2.3.1 bool gs1Set::const_iterator::operator!= ( const const_iterator & *it* ) const

#### 4.2.3.2 string gs1Set::const_iterator::operator* ( )

devuelve el codigo completo al que apunta el iterador. Nota: Se encuentra en el camino que hay desde el nodo hacia la raiz.

#### 4.2.3.3 gs1Set::const_iterator & gs1Set::const_iterator::operator++ ( )

avanza hacia el siguiente final de (sub)codigo en preorden en el **gs1Set** (p. 14) . Nota: Avanza por el arbol hasta el siguiente nodo que es final de (sub)codigo.

#### 4.2.3.4 gs1Set::const_iterator gs1Set::const_iterator::operator++ ( int )

avanza hacia el siguiente final de (sub)codigo en preorden en el **gs1Set** (p. 14). Nota: Avanza por el arbol hasta el siguiente nodo que es final de (sub)codigo.

#### 4.2.3.5 gs1Set::const_iterator & gs1Set::const_iterator::operator= ( const const_iterator & *it* )

#### 4.2.3.6 bool gs1Set::const_iterator::operator== ( const const_iterator & *it* ) const

#### 4.2.3.7 gs1Set::const_iterator & gs1Set::const_iterator::upper_IA ( )

sube hacia el identificador de aplicacion (IA) anterior en el codigo.

### 4.2.4 Friends And Related Function Documentation

#### 4.2.4.1 friend class gs1Set [friend]

The documentation for this class was generated from the following files:

- **gs1Set.h**
- **gs1Set.cpp**

## 4.3 tree< T >::const_leveliterator Class Reference

`#include <tree.h>`

### Public Member Functions

- **const_leveliterator** ()
- **const_leveliterator** (**node** n)
- bool **operator!=** (const **const_leveliterator** &i) const
- bool **operator==** (const **const_leveliterator** &i) const
- const T & **operator*** () const
- **const_leveliterator** & **operator++** ()

### 4.3.1 Constructor & Destructor Documentation

#### 4.3.1.1 template<typename T > tree< T >::const_leveliterator::const_leveliterator ( ) [inline]

#### 4.3.1.2 template<typename T> tree< T >::const_leveliterator::const_leveliterator ( node *n* )

### 4.3.2 Member Function Documentation

#### 4.3.2.1 template<typename T> bool tree< T >::const_leveliterator::operator!= ( const const_leveliterator & *i* ) const [inline]

#### 4.3.2.2 template<typename T > const T & tree< T >::const_leveliterator::operator* ( ) const [inline]

#### 4.3.2.3 template<typename T > tree< T >::const_leveliterator & tree< T >::const_leveliterator::operator++ ( )

#### 4.3.2.4 template<typename T> bool tree< T >::const_leveliterator::operator== ( const const_leveliterator & *i* ) const [inline]

The documentation for this class was generated from the following files:

- **tree.h**
- **tree.hxx**

## 4.4 tree< T >::const_node Class Reference

`#include <tree.h>`

### Public Member Functions

- **const_node** ()
  *Constructor primitivo.*
- **const_node** (const **const_node** &n)
  *Constructor copia.*
- **const_node** (const **node** &n)
  *Constructor copia, transforma un node en un **const_node** (p. 10).*
- bool **null** () const
  *Devuelve si el nodo es nulo.*
- **const_node parent** () const
  *Devuelve el padre del nodo receptor.*
- **const_node left** () const
  *Devuelve el hizo izquierdo del nodo receptor.*
- **const_node next_sibling** () const
  *Devuelve el hermano derecho del nodo receptor.*
- const T & **operator*** () const
  *Devuelve la etiqueta del nodo.*
- **const_node** & **operator=** (const **const_node** &n)
  *Operador de asignación.*
- bool **operator==** (const **const_node** &n) const
  *Operador de comparación de igualdad.*
- bool **operator!=** (const **const_node** &n) const
  *Operador de comparación de desigualdad.*

### Friends

- class **tree**< **T** >

### 4.4.1 Constructor & Destructor Documentation

#### 4.4.1.1 template<typename T > tree< T >::const_node::const_node ( ) [inline]

Constructor primitivo.

#### 4.4.1.2 template<typename T> tree< T >::const_node::const_node ( const const_node & *n* ) [inline]

Constructor copia.

#### 4.4.1.3 template<typename T> tree< T >::const_node::const_node ( const node & *n* ) [inline]

Constructor copia, transforma un node en un **const_node** (p. 10).

### 4.4.2 Member Function Documentation

#### 4.4.2.1 template<typename T > tree< T >::const_node tree< T >::const_node::left ( ) const [inline]

Devuelve el hizo izquierdo del nodo receptor.

**Precondition**

El nodo receptor no puede ser nulo

#### 4.4.2.2 template<typename T > tree< T >::const_node tree< T >::const_node::next_sibling ( ) const [inline]

Devuelve el hermano derecho del nodo receptor.

**Precondition**

El nodo receptor no puede ser nulo

#### 4.4.2.3 template<typename T > bool tree< T >::const_node::null ( ) const [inline]

Devuelve si el nodo es nulo.

#### 4.4.2.4 template<typename T > bool tree< T >::const_node::operator!= ( const const_node & *n* ) const [inline]

Operador de comparación de desigualdad.

**Parameters**

| | |
|---|---|
| *n* | el nodo con el que se compara |

#### 4.4.2.5 template<typename T > const T & tree< T >::const_node::operator* ( ) const [inline]

Devuelve la etiqueta del nodo.

**Precondition**

El nodo receptor no puede ser nulo

#### 4.4.2.6 template<typename T> tree< T >::const_node & tree< T >::const_node::operator= ( const const_node & *n* ) [inline]

Operador de asignación.

**Parameters**

| | |
|---|---|
| *n* | el nodo a asignar |

#### 4.4.2.7 template<typename T > bool tree< T >::const_node::operator== ( const const_node & *n* ) const [inline]

Operador de comparación de igualdad.

**Parameters**

| | |
|---|---|
| *n* | el nodo con el que se compara |

#### 4.4.2.8 template<typename T > tree< T >::const_node tree< T >::const_node::parent ( ) const [inline]

Devuelve el padre del nodo receptor.

**Precondition**

El nodo receptor no puede ser nulo

### 4.4.3 Friends And Related Function Documentation

#### 4.4.3.1 template<typename T> friend class tree< T > [friend]

The documentation for this class was generated from the following files:

- **tree.h**
- **nodetree.hxx**

## 4.5 tree< T >::const_postorderiterator Class Reference

`#include <tree.h>`

### Public Member Functions

- **const_postorderiterator** ()
- **const_postorderiterator** (**node** n)
- bool **operator!=** (const **const_postorderiterator** &i) const
- bool **operator==** (const **const_postorderiterator** &i) const
- const T & **operator*** () const
- **const_postorderiterator** & **operator++** ()

### 4.5.1 Constructor & Destructor Documentation

#### 4.5.1.1 template<typename T > tree< T >::const_postorderiterator::const_postorderiterator ( ) [inline]

#### 4.5.1.2 template<typename T> tree< T >::const_postorderiterator::const_postorderiterator ( node *n* )

### 4.5.2 Member Function Documentation

#### 4.5.2.1 template<typename T> bool tree< T >::const_postorderiterator::operator!= ( const const_postorderiterator & *i* ) const [inline]

#### 4.5.2.2 template<typename T > const T & tree< T >::const_postorderiterator::operator* ( ) const [inline]

#### 4.5.2.3 template<typename T > tree< T >::const_postorderiterator & tree< T >::const_postorderiterator::operator++ ( )

#### 4.5.2.4 template<typename T> bool tree< T >::const_postorderiterator::operator== ( const const_postorderiterator & *i* ) const [inline]

The documentation for this class was generated from the following files:

- **tree.h**
- **tree.hxx**

## 4.6 tree< T >::const_preorderiterator Class Reference

`#include <tree.h>`

### Public Member Functions

- **const_preorderiterator** ()
- **const_preorderiterator** (**node** n)
- bool **operator!=** (const **const_preorderiterator** &i) const
- bool **operator==** (const **const_preorderiterator** &i) const
- const T & **operator*** () const
- **const_preorderiterator** & **operator++** ()

### 4.6.1 Constructor & Destructor Documentation

#### 4.6.1.1 template<typename T > tree< T >::const_preorderiterator::const_preorderiterator ( ) [inline]

#### 4.6.1.2 template<typename T> tree< T >::const_preorderiterator::const_preorderiterator ( node *n* )

### 4.6.2 Member Function Documentation

#### 4.6.2.1 template<typename T> bool tree< T >::const_preorderiterator::operator!= ( const const_preorderiterator & *i* ) const [inline]

#### 4.6.2.2 template<typename T > const T & tree< T >::const_preorderiterator::operator* ( ) const [inline]

#### 4.6.2.3 template<typename T > tree< T >::const_preorderiterator & tree< T >::const_preorderiterator::operator++ ( )

#### 4.6.2.4 template<typename T> bool tree< T >::const_preorderiterator::operator== ( const const_preorderiterator & *i* ) const [inline]

The documentation for this class was generated from the following files:

- **tree.h**
- **tree.hxx**

## 4.7 gs1Set Class Reference

`#include <gs1Set.h>`

### Classes

- class **const_iterator**

### Public Types

- typedef unsigned int **size_type**

### Public Member Functions

- **gs1Set** ()

  *Constructor primitivo crea un **gs1Set** (p. 14) con el caracter '-' en el nodo raiz.*
- **gs1Set** (const **gs1Set** &x)

  *Constructor de copia.*
- void **reading_gs1Set** ()

  *Lectura de un **gs1Set** (p. 14) por teclado. Se genera el **gs1Set** (p. 14) utilizando un recorrido por nivel.*
- **gs1Set** & **operator=** (const **gs1Set** &org)

  *operador de asignacion*
- bool **insert** (const string &s)

  *Inserta una nuevo codigo dentro del **gs1Set** (p. 14).*
- bool **erase** (const string &s)

  *elimina el codigo de un **gs1Set** (p. 14)*
- **const_iterator find** (const string &s)

  *busca un codigo*
- list< string > **codesWithPrefix** (const string &pr)

  *obtiene todos los codigos que tienen la misma secuencia prefijo*
- **size_type size** () const

  *tamanioo*
- bool **empty** () const

  *Chequea si el **gs1Set** (p. 14) esta vacio (**size()** (p. 18)==0)*
- **const_iterator begin** () const

  *iterador a la primera palabra del conjunto.*
- **const_iterator end** () const

  *iterador al fin del conjunto*
- void **print** () const

  *imprime todos los codigos almacenados*

### 4.7.1 Detailed Description

**gs1Set** (p. 14) :**gs1Set** (p. 14) , operator=, size, empty, insert, erase, find,

Descripcion

(las tildes han sido omitidas deliberadamente, debido a fallos en la generacion de documentacion con doxygen)

Un objeto de la clase **gs1Set** (p. 14) representara un contenedor que permite almacenar un codigo electronico de producto, en concreto consideraremos la normativa gs1-128

El gs1-128 es un sistema estandar de identificacion mediante codigo de barras utilizado internacionalmente para la identificacion de mercancias en entornos logisticos y no detallistas. Este sistema se utiliza principalmente para la identificacion de unidades de expedicion.

En esta practica consideraremos codigos correctos, aquellos obtenidos segun la siguiente definicion: Una codigo correcto solo puede contener digitos (del 0 al 9) junto con parentesis de apertura y cierre. Por tanto, no debe tener espacios ni delimitadores del tipo comillas, llaves, comas, puntos, etc.

El codigo gs1-128 vendra se representara mediante un string, ejemplos de codigos validos son: (01)18456789012342 (02)18456789012359(37)1234(00)384567890123456782

En la practica NO nos preocuparemos de generar codigos correctos, sino que consideraremos como correcto cualquier codigo que tenga

En el codigo podemos distinguir dos partes, encerrados entre parentesis los identificadores de aplicacion, IA, que son unos prefijos numericos creados para dar significado inequivoco a los elementos de datos estandarizados que se encuentran situados a continuacion (son subcodigos dentro el codigo gs1). Cada prefijo identifica el significado y el formato de los subcadigos que le siguen.

En la actualidad, existen mas de 100 identificadores de aplicacion estandarizados internacionalmente. Por ejemplo,

- 00 Codigo Seriado de la Unidad de Envio (SSCC)
- 01 Codigo de agrupacion
- 02 Codigo del articulo / agrupacion contenido
- 37 Cantidades (va junto al IA 02)
- 10 Numero de lote
- 11 Fecha de fabricacion
- 13 Fecha de envasado
- 15 Fecha de consumo preferente
- 17 Fecha de caducidad etc.

En la practica NO nos preocuparemos de generar codigos correctos, sino que consideraremos como correcto cualquier codigo que tenga el formato (yy)xxxxz(yyy)xxxxxxxz(yy)xxxxxz, donde yyy e xxxxxz son digitos del 0 al 9 representando el IA y el codigo asociado. (El alumno interesado puede consultar como se construyen los codigos enel enlace `http://www.aecoc.es/BAJAR/.php?id_doc=1178&id=GS1%20128.pdf&folder=documento_socio`

Veamos un ejemplo simple, podemos considerar los siguientes codigos

(02)8423(10)0980(11)141215 (02)8423(10)0981(11)141215 (02)8324(10)0982(13)141010 (03)842442(10)211(73)0120(11)12334

El objetivo de la practica es construir un contenedor de codigos gs1, donde el objetivo principal es permitir el acceso lo mas rapido posible a un determinado codigo, asumiendo que cada uno de los codigos que han sido obtenidos (por ejemplo, leidos de un codigo de barras del productos) es correcto.

**Todo** Tareas a realizar: El alumno debera implementar la clase **gs1Set** (p. 14) , junto con el codigo de prueba de los distintos metodos.

### 4.7.2 Member Typedef Documentation

#### 4.7.2.1 typedef gs1Set::size_type

Hace referencia al tipo asociado al numero de elementos en el codigo.

### 4.7.3 Constructor & Destructor Documentation

#### 4.7.3.1 gs1Set::gs1Set ( )

Constructor primitivo crea un **gs1Set** (p. 14) con el caracter '-' en el nodo raiz.

#### 4.7.3.2 gs1Set::gs1Set ( const gs1Set & *x* )

Constructor de copia.

**Parameters**

| | | |
|---|---|---|
| in | *x* | **gs1Set** (p. 14) que se copia |

### 4.7.4 Member Function Documentation

#### 4.7.4.1 gs1Set::const_iterator gs1Set::begin ( ) const

iterador a la primera palabra del conjunto.

Este iterador debe apuntar al nodo en el que se encuentra el ultimo caracter de la primera palabra en el conjunto.

#### 4.7.4.2 list<string> gs1Set::codesWithPrefix ( const string & *pr* )

obtiene todos los codigos que tienen la misma secuencia prefijo

**Parameters**

| | | |
|---|---|---|
| in | *pr* | prefijo a buscar |

**Returns**

una lista con todos los codigos epc que contienen el mismo prefijo

#### 4.7.4.3 bool gs1Set::empty ( ) const

Chequea si el **gs1Set** (p. 14) esta vacio (**size()** (p. 18)==0)

#### 4.7.4.4 gs1Set::const_iterator gs1Set::end ( ) const

iterador al fin del conjunto

#### 4.7.4.5 bool gs1Set::erase ( const string & *s* )

elimina el codigo de un **gs1Set** (p. 14)

**Parameters**

| | | |
|---|---|---|
| in | *s* | elemento a borrar. Este elemento puede identificar a un prefijo, por ejemplo el codigo asociado a un producto, por lo que todos los codigos que contienen dicho prefijo seran eliminados. |

**Returns**

el numero de codigos que se han borrado, cero si el borrado no se ha podido realizar con exito

**Postcondition**

el **size()** (p. 18) sera decrementado.

#### 4.7.4.6 gs1Set::const_iterator gs1Set::find ( const string & *s* )

busca un codigo

**Parameters**

| | | |
|---|---|---|
| in | *s* | nombre del codigo (o prefijo) a buscar |

**Returns**

un iterador que apunta al codigo o **end()** (p. 16) si el codigo (prefijo) no existe.

**Parameters**

| | | |
|---|---|---|
| in | *s* | nombre del codigo a buscar |

**Returns**

un iterador que apunta al codigo o **end()** (p. 16) si el codigo no existe.

#### 4.7.4.7 bool gs1Set::insert ( const string & *s* )

Inserta una nuevo codigo dentro del **gs1Set** (p. 14).

**Parameters**

| | | |
|---|---|---|
| in | *s* | elemento a insertar |

**Returns**

bool true si la insercion se ha podido realizar con exito, esto es, el codigo no pertenecia al **gs1Set** (p. 14)

**Postcondition**

el **size()** (p. 18) sera incrementado

#### 4.7.4.8 gs1Set& gs1Set::operator= ( const gs1Set & *org* )

operador de asignacion

**Parameters**

| | | |
|---|---|---|
| in | *org* | **gs1Set** (p. 14) a copiar. Crea un **gs1Set** (p. 14) duplicado exacto a org. |

#### 4.7.4.9 void gs1Set::print ( ) const

imprime todos los codigos almacenados

#### 4.7.4.10 void gs1Set::reading_gs1Set ( )

Lectura de un **gs1Set** (p. 14) por teclado. Se genera el **gs1Set** (p. 14) utilizando un recorrido por nivel.

**Todo** implementar este metodo correctamente OJO ESTE METODO OS SIRVE PARA PODER CONSTRUIR UN ARBOL NO TIENE EN CUENTA EL INVARIANTE DE LA REPRESENTACION AL NO CONSIDERAR EL CAMPO INT DEL NODO!!!!! DEBEIS MODIFICARLO PARA QUE LO HAGA DE FORMA CORRECTA

#### 4.7.4.11 gs1Set::size_type gs1Set::size ( ) const

tamanioo

**Returns**

delvuelve el numero de palabras del **gs1Set** (p. 14)

The documentation for this class was generated from the following files:

- **gs1Set.h**
- **gs1Set.cpp**

## 4.8 tree< T >::inorderiterator Class Reference

`#include <tree.h>`

### Public Member Functions

- **inorderiterator** ()
- **inorderiterator** (**node** n)
- bool **operator!=** (const **inorderiterator** &i) const
- bool **operator==** (const **inorderiterator** &i) const
- T & **operator*** ()
- **inorderiterator** & **operator++** ()

### 4.8.1 Detailed Description

**template<typename T>class tree< T >::inorderiterator**

Clase iterator para recorrer el árbol en Inorder

### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 template<typename T > tree< T >::inorderiterator::inorderiterator ( ) [inline]

#### 4.8.2.2 template<typename T> tree< T >::inorderiterator::inorderiterator ( node *n* )

### 4.8.3 Member Function Documentation

#### 4.8.3.1 template<typename T> bool tree< T >::inorderiterator::operator!= ( const inorderiterator & *i* ) const [inline]

#### 4.8.3.2 template<typename T > T & tree< T >::inorderiterator::operator* ( ) [inline]

#### 4.8.3.3 template<typename T > tree< T >::inorderiterator & tree< T >::inorderiterator::operator++ ( )

#### 4.8.3.4 template<typename T> bool tree< T >::inorderiterator::operator== ( const inorderiterator & *i* ) const [inline]

The documentation for this class was generated from the following files:

- **tree.h**
- **tree.hxx**

## 4.9 tree< T >::leveliterator Class Reference

`#include <tree.h>`

### Public Member Functions

- **leveliterator** ()
- **leveliterator** (**node** n)
- bool **operator!=** (const **leveliterator** &i) const
- bool **operator==** (const **leveliterator** &i) const
- T & **operator*** ()
- **leveliterator** & **operator++** ()

### 4.9.1 Detailed Description

template<typename T>class tree< T >::leveliterator

Clase iterator para recorrer el árbol por niveles

### 4.9.2 Constructor & Destructor Documentation

#### 4.9.2.1 template<typename T > tree< T >::leveliterator::leveliterator ( ) [inline]

#### 4.9.2.2 template<typename T> tree< T >::leveliterator::leveliterator ( node *n* )

### 4.9.3 Member Function Documentation

#### 4.9.3.1 template<typename T> bool tree< T >::leveliterator::operator!= ( const leveliterator & *i* ) const [inline]

#### 4.9.3.2 template<typename T > T & tree< T >::leveliterator::operator∗ ( ) [inline]

#### 4.9.3.3 template<typename T > tree< T >::leveliterator & tree< T >::leveliterator::operator++ ( )

#### 4.9.3.4 template<typename T> bool tree< T >::leveliterator::operator== ( const leveliterator & *i* ) const [inline]

The documentation for this class was generated from the following files:

- **tree.h**
- **tree.hxx**

## 4.10 tree< T >::node Class Reference

`#include <tree.h>`

### Public Member Functions

- **node** ()

  *Constructor primitivo.*
- **node** (const **node** &n)

  *Constructor copia.*
- void **setlabel** (const T &e)

  *Modifica la etiqueta.*
- bool **null** () const

  *Devuelve si el nodo es nulo.*
- **node parent** () const

  *Devuelve el padre del nodo receptor.*
- **node left** () const

  *Devuelve el hizo izquierdo del nodo receptor.*
- **node next_sibling** () const

  *Devuelve el hermano derecho del nodo receptor.*
- T & **operator**∗ ()

  *Devuelve la etiqueta del nodo.*
- const T & **operator**∗ () const

  *Devuelve la etiqueta del nodo.*
- **node** & **operator=** (const **node** &n)

  *Operador de asignación.*
- bool **operator==** (const **node** &n) const

  *Operador de comparación de igualdad.*
- bool **operator!=** (const **node** &n) const

  *Operador de comparación de desigualdad.*

### Friends

- class **tree**< **T** >

### 4.10.1 Detailed Description

template<typename T>class tree< T >::node

Descripción

Representa a los nodos del árbol

Descripción

Representa a los nodos del árbol, se usa con arboles constantes

### 4.10.2 Constructor & Destructor Documentation

#### 4.10.2.1 template<typename T > tree< T >::node::node ( ) [inline]

Constructor primitivo.

#### 4.10.2.2 template<typename T> tree< T >::node::node ( const node & *n* ) [inline]

Constructor copia.

### 4.10.3 Member Function Documentation

#### 4.10.3.1 template<typename T > tree< T >::node tree< T >::node::left ( ) const [inline]

Devuelve el hizo izquierdo del nodo receptor.

**Precondition**

El nodo receptor no puede ser nulo

#### 4.10.3.2 template<typename T > tree< T >::node tree< T >::node::next_sibling ( ) const [inline]

Devuelve el hermano derecho del nodo receptor.

**Precondition**

El nodo receptor no puede ser nulo

#### 4.10.3.3 template<typename T > bool tree< T >::node::null ( ) const [inline]

Devuelve si el nodo es nulo.

#### 4.10.3.4 template<typename T > bool tree< T >::node::operator!= ( const node & *n* ) const [inline]

Operador de comparación de desigualdad.

**Parameters**

| | |
|---|---|
| *n* | el nodo con el que se compara |

#### 4.10.3.5 template<typename T > T & tree< T >::node::operator*( ) [inline]

Devuelve la etiqueta del nodo.

**Precondition**

Si se usa como consultor, !n.Nulo()

#### 4.10.3.6 template<typename T > const T & tree< T >::node::operator*( ) const [inline]

Devuelve la etiqueta del nodo.

**Precondition**

El nodo receptor no puede ser nulo

#### 4.10.3.7 template<typename T> tree< T >::node & tree< T >::node::operator=( const node & *n* ) [inline]

Operador de asignación.

**Parameters**

| | |
|---|---|
| *n* | el nodo a asignar |

#### 4.10.3.8 template<typename T > bool tree< T >::node::operator==( const node & *n* ) const [inline]

Operador de comparación de igualdad.

**Parameters**

| | |
|---|---|
| *n* | el nodo con el que se compara |

#### 4.10.3.9 template<typename T > tree< T >::node tree< T >::node::parent( ) const [inline]

Devuelve el padre del nodo receptor.

**Precondition**

El nodo receptor no puede ser nulo

#### 4.10.3.10 template<typename T> void tree< T >::node::setlabel( const T & *e* ) [inline]

Modifica la etiqueta.

### 4.10.4 Friends And Related Function Documentation

#### 4.10.4.1 template<typename T> friend class tree< T > [friend]

The documentation for this class was generated from the following files:

- **tree.h**
- **nodetree.hxx**

## 4.11 tree< T >::postorderiterator Class Reference

`#include <tree.h>`

### Public Member Functions

- **postorderiterator** ()
- **postorderiterator** (**node** n)
- bool **operator!=** (const **postorderiterator** &i) const
- bool **operator==** (const **postorderiterator** &i) const
- T & **operator*** ()
- **postorderiterator** & **operator++** ()

### 4.11.1 Detailed Description

**template<typename T>class tree< T >::postorderiterator**

Clase iterator para recorrer el árbol en PostOrden

### 4.11.2 Constructor & Destructor Documentation

#### 4.11.2.1 template<typename T > tree< T >::postorderiterator::postorderiterator ( ) [inline]

#### 4.11.2.2 template<typename T> tree< T >::postorderiterator::postorderiterator ( node *n* )

### 4.11.3 Member Function Documentation

#### 4.11.3.1 template<typename T> bool tree< T >::postorderiterator::operator!= ( const postorderiterator & *i* ) const [inline]

#### 4.11.3.2 template<typename T > T & tree< T >::postorderiterator::operator* ( ) [inline]

#### 4.11.3.3 template<typename T > tree< T >::postorderiterator & tree< T >::postorderiterator::operator++ ( )

#### 4.11.3.4 template<typename T> bool tree< T >::postorderiterator::operator== ( const postorderiterator & *i* ) const [inline]

The documentation for this class was generated from the following files:

- **tree.h**
- **tree.hxx**

## 4.12 tree< T >::preorderiterator Class Reference

`#include <tree.h>`

### Public Member Functions

- **preorderiterator** ()
- **preorderiterator** (**node** n)
- bool **operator!=** (const **preorderiterator** &i) const
- bool **operator==** (const **preorderiterator** &i) const
- T & **operator*** ()
- **preorderiterator** & **operator++** ()
- **preorderiterator operator++** (int)

### 4.12.1 Detailed Description

**template<typename T>class tree< T >::preorderiterator**

Clase iterator para recorrer el árbol en PreOrden

### 4.12.2 Constructor & Destructor Documentation

#### 4.12.2.1 template<typename T > tree< T >::preorderiterator::preorderiterator ( ) [inline]

#### 4.12.2.2 template<typename T> tree< T >::preorderiterator::preorderiterator ( node *n* )

### 4.12.3 Member Function Documentation

#### 4.12.3.1 template<typename T> bool tree< T >::preorderiterator::operator!= ( const preorderiterator & *i* ) const [inline]

#### 4.12.3.2 template<typename T > T & tree< T >::preorderiterator::operator* ( ) [inline]

#### 4.12.3.3 template<typename T > tree< T >::preorderiterator & tree< T >::preorderiterator::operator++ ( )

#### 4.12.3.4 template<typename T > tree< T >::preorderiterator tree< T >::preorderiterator::operator++ ( int *i* )

#### 4.12.3.5 template<typename T> bool tree< T >::preorderiterator::operator== ( const preorderiterator & *i* ) const [inline]

The documentation for this class was generated from the following files:

- **tree.h**
- **tree.hxx**

## 4.13 tree< T > Class Template Reference

`#include <tree.h>`

### Classes

- class **const_inorderiterator**
- class **const_leveliterator**
- class **const_node**
- class **const_postorderiterator**
- class **const_preorderiterator**

- class **inorderiterator**
- class **leveliterator**
- class **node**
- class **postorderiterator**
- class **preorderiterator**

### Public Types

- typedef unsigned int **size_type**

### Public Member Functions

- **tree** ()

  *Constructor primitivo por defecto.*
- **tree** (const T &e)

  *Constructor primitivo.*
- **tree** (const **tree**< T > &a)

  *Constructor de copia.*
- void **assign_subtree** (const **tree**< T > &a, **node** n)

  *Reemplaza el receptor por una copia de subárbol.*
- ~**tree** ()

  *Destructor.*
- **node setroot** (const T &v)

  *Asigna la raiz al arbol vacío.*
- **tree**< T > & **operator=** (const **tree**< T > &a)

  *Operador de asignación.*
- **node root** () const

  *Obtener el nodo raíz.*
- void **prune_left** (**node** n, **tree**< T > &dest)

  *Podar el subárbol hijo a la izquierda de un nodo.*
- void **prune_right_sibling** (**node** n, **tree**< T > &dest)

  *Podar el subárbol hermano a la derecha de un nodo.*
- **node insert_left** (**node** n, const T &e)

  *Insertar un nodo como hijo a la izquierda de un nodo.*
- **node insert_left** (**node** n, **tree**< T > &rama)

  *Insertar un árbol como subárbol hijo a la izquierda de un nodo.*
- **node insert_right_sibling** (**node** n, const T &e)

  *Insertar un nodo como hermano a la derecha de un nodo.*
- **node insert_right_sibling** (**node** n, **tree**< T > &rama)

  *Insertar un árbol como subárbol hermano a la derecha de un nodo.*
- void **clear** ()

  *Hace nulo un árbol.*
- **size_type size** () const

  *Obtiene el nmero de nodos.*
- bool **empty** () const

  *Comprueba si un árbol esta vacío.*
- bool **null** () const

  *Comprueba si un árbol es nulo.*
- bool **is_root** (**node** n) const

  *Comprueba si un nodo es la raíz.*
- bool **is_internal** (**node** v) const

*Comprueba si un nodo es interior.*

- bool **is_external** (**node** v) const

  *Comprueba si un nodo es exterior.*

- bool **operator==** (const **tree**< T > &a) const

  *Operador de comparación de igualdad.*

- bool **operator!=** (const **tree**< T > &a) const

  *Operador de comparación de desigualdad.*

- **preorderiterator beginPreorder** ()
- **preorderiterator endPreorder** ()
- **const_preorderiterator beginPreorder** () const
- **const_preorderiterator endPreorder** () const
- **inorderiterator beginInorder** ()
- **inorderiterator endInorder** ()
- **const_inorderiterator beginInorder** () const
- **const_inorderiterator endInorder** () const
- **postorderiterator beginPostorder** ()
- **postorderiterator endPostorder** ()
- **const_postorderiterator beginPostorder** () const
- **const_postorderiterator endPostorder** () const
- **leveliterator beginlevel** ()
- **leveliterator endlevel** ()
- **const_leveliterator beginlevel** () const
- **const_leveliterator endlevel** () const

### 4.13.1 Detailed Description

**template<typename T>class tree< T >**

**tree::tree** (p. 26), assign_subtree, setroot, root, ∼tree, =, prune_left, prune_right_sibling,insert_left, insert_right_↩ sibling, clear, size, empty, ==, !=, is_root, internal, external

Representa un árbol general con nodos etiquetados con datos del tipo T.

T debe tener definidas las operaciones:

- T & operator=(const T & e);
- bool operator!=(const T & e);
- bool operator==(const T & e);

Son mutables. Residen en memoria dinámica.

### 4.13.2 Member Typedef Documentation

#### 4.13.2.1 template<typename T> tree< T >::size_type

Hace referencia al tipo asociado la tamaño del tree

### 4.13.3 Constructor & Destructor Documentation

#### 4.13.3.1 template<typename T > tree< T >::tree ( ) [inline]

Constructor primitivo por defecto.

Crea un árbol nulo.

#### 4.13.3.2 template<typename T> tree< T >::tree ( const T & *e* )

Constructor primitivo.

**Parameters**

| | |
|---|---|
| *e* | Etiqueta para la raíz. |

Crea un árbol con un único nodo etiquetado con e.

#### 4.13.3.3 template<typename T> tree< T >::tree ( const tree< T > & *a* )

Constructor de copia.

**Parameters**

| | |
|---|---|
| | a árbol que se copia. |

Crea un árbol duplicado exacto de a.

#### 4.13.3.4 template<typename T > tree< T >::~tree ( ) [inline]

Destructor.

Destruye el receptor liberando los recursos que ocupaba.

### 4.13.4 Member Function Documentation

#### 4.13.4.1 template<typename T> void tree< T >::assign_subtree ( const tree< T > & *a,* node *n* )

Reemplaza el receptor por una copia de subárbol.

**Parameters**

| | |
|---|---|
| *a* | árbol desde el que se copia. |
| *n* | nodo raíz del subárbol que se copia. |

El receptor se hace nulo y después se le asigna una copia del subárbol de a cuya raíz es n.

#### 4.13.4.2 template<typename T > tree< T >::inorderiterator tree< T >::beginInorder ( ) [inline]

#### 4.13.4.3 template<typename T > tree< T >::const_inorderiterator tree< T >::beginInorder ( ) const [inline]

#### 4.13.4.4 template<typename T > tree< T >::leveliterator tree< T >::beginlevel ( ) [inline]

#### 4.13.4.5 template<typename T > tree< T >::const_leveliterator tree< T >::beginlevel ( ) const [inline]

#### 4.13.4.6 template<typename T > tree< T >::postorderiterator tree< T >::beginPostorder ( ) [inline]

#### 4.13.4.7 template<typename T > tree< T >::const_postorderiterator tree< T >::beginPostorder ( ) const [inline]

#### 4.13.4.8 template<typename T > tree< T >::preorderiterator tree< T >::beginPreorder ( ) [inline]

#### 4.13.4.9 template<typename T > tree< T >::const_preorderiterator tree< T >::beginPreorder ( ) const [inline]

#### 4.13.4.10 template<typename T > void tree< T >::clear ( )

Hace nulo un árbol.

Destruye todos los nodos del árbol receptor y lo hace un árbol nulo.

#### 4.13.4.11 template<typename T > bool tree< T >::empty ( ) const [inline]

Comprueba si un árbol esta vacío.

**Returns**

true, si el receptor es un árbol vacío. false, en otro caso.

#### 4.13.4.12 template<typename T > tree< T >::inorderiterator tree< T >::endInorder ( ) [inline]

#### 4.13.4.13 template<typename T > tree< T >::const_inorderiterator tree< T >::endInorder ( ) const [inline]

#### 4.13.4.14 template<typename T > tree< T >::leveliterator tree< T >::endlevel ( ) [inline]

#### 4.13.4.15 template<typename T > tree< T >::const_leveliterator tree< T >::endlevel ( ) const [inline]

#### 4.13.4.16 template<typename T > tree< T >::postorderiterator tree< T >::endPostorder ( ) [inline]

#### 4.13.4.17 template<typename T > tree< T >::const_postorderiterator tree< T >::endPostorder ( ) const [inline]

#### 4.13.4.18 template<typename T > tree< T >::preorderiterator tree< T >::endPreorder ( ) [inline]

#### 4.13.4.19 template<typename T > tree< T >::const_preorderiterator tree< T >::endPreorder ( ) const [inline]

#### 4.13.4.20 template<typename T> node tree< T >::insert_left ( node *n,* const T & *e* )

Insertar un nodo como hijo a la izquierda de un nodo.

**Parameters**

| | |
|---|---|
| *n* | nodo del receptor. n != nodo_nulo. |
| *e* | etiqueta del nuevo nodo. |

Inserta un nuevo nodo con etiqueta e como hijo a la izquierda, el anterior hijo más a la izquierda queda como hermano a la derecha del recién insertado

#### 4.13.4.21 template<typename T> node tree< T >::insert_left ( node *n,* tree< T > & *rama* )

Insertar un árbol como subárbol hijo a la izquierda de un nodo.

**Parameters**

| | |
|---|---|
| *n* | nodo del receptor. n != nodo_nulo. |
| *rama* | subárbol que se inserta. Es MODIFICADO. |

Si rama no es un árbol vacío: Inserta rama como hijo a la izquierda de n, el anterior hijo más a la izquierda queda como hermana a la derecha del recién insertado. y rama se hace árbol nulo. En caso contrario no se hace nada

#### 4.13.4.22 template<typename T> node tree< T >::insert_right_sibling ( node *n,* const T & *e* )

Insertar un nodo como hermano a la derecha de un nodo.

**Parameters**

| | |
|---|---|
| *n* | nodo del receptor. !n.Nulo(). |
| *e* | etiqueta del nuevo nodo. |

Inserta un nuevo nodo con etiqueta e como hermano a la derecha, el anterior hermano a la derecha de n queda como hermano a la derecha del nodo insertado

#### 4.13.4.23 template<typename T> node tree< T >::insert_right_sibling ( node *n,* tree< T > & *rama* )

Insertar un árbol como subárbol hermano a la derecha de un nodo.

**Parameters**

| | |
|---|---|
| *n* | nodo del receptor. !n.Nulo(). |
| *rama* | subárbol que se inserta. Es MODIFICADO. |

Si rama no es un árbol vacío: Asigna el valor de rama como nuevo subárbol hermano a la derecha, el anteriot hermano a la derecha de n queda como hermano a la derecha del nodo insertado y rama se hace árbol nulo. En caso contrario no se hace nada

#### 4.13.4.24 template<typename T> bool tree< T >::is_external ( node *v* ) const [inline]

Comprueba si un nodo es exterior.

**Parameters**

| | |
|---|---|
| *v* | nodo que se evala. |

**Returns**

true, si n es exterior. false, en otro caso.

#### 4.13.4.25 template<typename T> bool tree< T >::is_internal ( node *v* ) const [inline]

Comprueba si un nodo es interior.

**Parameters**

| | |
|---|---|
| *v* | nodo que se evala. |

**Returns**

true, si n es interior. false, en otro caso.

#### 4.13.4.26 template<typename T> bool tree< T >::is_root ( node *n* ) const [inline]

Comprueba si un nodo es la raíz.

**Parameters**

| | |
|---|---|
| *n* | nodo que se evala. |

**Returns**

true, si n es la raíz del receptor. false, en otro caso.

#### 4.13.4.27 template<typename T> bool tree< T >::null ( ) const

Comprueba si un árbol es nulo.

**Returns**

true, si el receptor es un árbol nulo. false, en otro caso.

#### 4.13.4.28 template<typename T> bool tree< T >::operator!= ( const tree< T > & *a* ) const [inline]

Operador de comparación de desigualdad.

**Parameters**

| | |
|---|---|
| *a* | árbol con que se compara el receptor. |

**Returns**

true, si el receptor no es igual, en estructura o etiquetas a a. false, en otro caso.

#### 4.13.4.29 template<typename T> tree< T > & tree< T >::operator= ( const tree< T > & *a* )

Operador de asignación.

**Parameters**

| | |
|---|---|
| *a* | árbol que se asigna. |

Destruye el contenido previo del receptor y le asigna un duplicado de a.

#### 4.13.4.30 template<typename T> bool tree< T >::operator== ( const tree< T > & *a* ) const [inline]

Operador de comparación de igualdad.

**Parameters**

| | |
|---|---|
| *a* | árbol con que se compara el receptor. |

**Returns**

true, si el receptor es igual, en estructura y etiquetas a a. false, en otro caso.

#### 4.13.4.31 template<typename T> void tree< T >::prune_left ( node *n,* tree< T > & *dest* )

Podar el subárbol hijo a la izquierda de un nodo.

**Parameters**

| | |
|---|---|
| *n* | nodo del receptor. n != nodo_nulo. |
| *dest* | subárbol hijo a la izquierda de n. Es MODIFICADO. |

Desconecta el subárbol hijo a la izquierda de n, que pasa a ser el árbol que era su hermano a la derecha, si lo tuviera. El subárbol anterior se devuelve sobre dest.

#### 4.13.4.32 template<typename T> void tree< T >::prune_right_sibling ( node *n,* tree< T > & *dest* )

Podar el subárbol hermano a la derecha de un nodo.

**Parameters**

| | |
|---|---|
| *n* | nodo del receptor. n != nodo_nulo. |
| *dest* | subárbol hermano a la derecha de n. Es MODIFICADO. |

Desconecta el subárbol hermano a la derecha de n, que pasa a ser el árbol que era su hermano a la derecha, si lo tuviera. El subárbol anterior se devuelve sobre dest.

#### 4.13.4.33 template<typename T> tree< T >::node tree< T >::root ( ) const [inline]

Obtener el nodo raíz.

**Returns**

nodo raíz del receptor.

#### 4.13.4.34 template<typename T> tree< T >::node tree< T >::setroot ( const T & *v* )

Asigna la raiz al arbol vacío.

**Parameters**

| | |
|---|---|
| *v* | el valor a almacenar en la raiz. |

**Precondition**

el receptor es el árbol nulo.

#### 4.13.4.35 template<typename T> tree< T >::size_type tree< T >::size ( ) const [inline]

Obtiene el nmero de nodos.

**Returns**

nmero de nodos del receptor.

The documentation for this class was generated from the following files:

- **tree.h**
- **tree.hxx**

# Chapter 5

# File Documentation

## 5.1 generadorCodigos.cpp File Reference

```
#include <iostream>
#include <random>
#include <string>
#include <map>
```

### Functions

- int **main** (int argc, char ∗argv[])

### 5.1.1 Function Documentation

#### 5.1.1.1 int main ( int *argc,* char ∗ *argv[]* )

## 5.2 gs1Set.cpp File Reference

```
#include "gs1Set.h"
#include <vector>
```

## 5.3 gs1Set.h File Reference

```
#include <iostream>
#include <string>
#include "tree.h"
```

### Classes

- class **gs1Set**
- class **gs1Set::const_iterator**

## 5.4 nodetree.hxx File Reference

```
#include <cassert>
```

## 5.5 prueba_gs1.cpp File Reference

```
#include <iostream>
#include <set>
#include <unordered_set>
#include "gs1Set.h"
#include <fstream>
```

### Functions

- template<typename T >
  void **load** (T &contenedor, const string &s)

  *Carga el fichero en memoria.*
- int **main** (int argc, char *argv[])

### 5.5.1 Function Documentation

#### 5.5.1.1 template<typename T > void load ( T & *contenedor,* const string & *s* )

Carga el fichero en memoria.

**Parameters**

| | |
|---|---|
| *contenedor* | contenedor de salida |
| *s* | nombre del fichero |

**Precondition**

T debe tener el método insert(const string &)

#### 5.5.1.2 int main ( int *argc,* char * *argv[]* )

## 5.6 tree.h File Reference

TDA tree.

```
#include <queue>
#include <iostream>
#include <stack>
#include <list>
#include "tree.hxx"
#include "nodetree.hxx"
```

### Classes

- class **tree**< **T** >
- class **tree**< **T** >**::preorderiterator**
- class **tree**< **T** >**::const_preorderiterator**
- class **tree**< **T** >**::inorderiterator**
- class **tree**< **T** >**::const_inorderiterator**
- class **tree**< **T** >**::postorderiterator**
- class **tree**< **T** >**::const_postorderiterator**
- class **tree**< **T** >**::leveliterator**
- class **tree**< **T** >**::const_leveliterator**
- class **tree**< **T** >**::node**
- class **tree**< **T** >**::const_node**

#### 5.6.1 Detailed Description

TDA tree.

## 5.7 tree.hxx File Reference

```
#include <cassert>
#include <iostream>
```