# Tiposabstractosdedatos

3.0

Generado por Doxygen 1.7.5.1

Lunes, 13 de Mayo de 2013 21:25:14

# Índice general

# Capítulo 1

# Índice de clases

## 1.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

# Capítulo 2

# Indice de archivos

## 2.1. Lista de archivos

Lista de todos los archivos documentados y con descripciones breves:

# Capítulo 3

# Documentación de las clases

## 3.1. Referencia de la plantilla de la Clase Arbin< T >

`#include <ArbinConCopias.h>`

### Clases

- class Nodo

### Métodos públicos

- Arbin ()
- Arbin (const Arbin &iz, const T &elem, const Arbin &dr)
- ~Arbin ()
- const T & raiz () const
- Arbin hijoIz () const
- Arbin hijoDr () const
- bool esVacio () const
- unsigned int numNodos () const
- unsigned int talla () const
- unsigned int numHojas () const
- Arbin (const Arbin< T > &other)
- Arbin< T > & operator= (const Arbin< T > &other)
- bool operator== (const Arbin< T > &rhs) const
- Arbin ()
- Arbin (const Arbin &iz, const T &elem, const Arbin &dr)
- ~Arbin ()
- const T & raiz () const
- Arbin hijoIz () const
- Arbin hijoDr () const
- bool esVacio () const

- unsigned int numNodos () const
- unsigned int talla () const
- unsigned int numHojas () const
- Arbin (const Arbin< T > &other)
- Arbin< T > & operator= (const Arbin< T > &other)
- bool operator== (const Arbin< T > &rhs) const

### Métodos públicos estáticos

- template<class T >
  static Arbin< T > construyeYVacia (Arbin< T > &iz, const T &elem, Arbin< T > &dr)

### Métodos protegidos

- Arbin (Nodo *raiz)
- Arbin (Nodo *raiz)

### Atributos protegidos

- Nodo * _ra

### 3.1.1. Descripción detallada

**template<class T>class Arbin< T >**

Implementación dinámica del TAD Arbin utilizando nodos con un puntero al hijo izquierdo y otro al hijo derecho. La implementación permite compartición de estructura, manteniendola bajo control mediante conteo de referencias. La implementación, sin embargo, es bastante artesanal, pues para no complicar el código excesivamente no se ha hecho uso de punteros inteligentes que incrementen y decrementen automáticamente esas referencias.

Las operaciones son:

- ArbolVacio: -> Arbin. Generadora implementada en el constructor sin parámetros.
- Cons: Arbin, Elem, Arbin -> Arbin. Generadora implementada en un constructor con tres parámetros.
- hijoIz, hijoDr: Arbin - -> Arbin. Observadoras que devuelven el hijo izquiero o derecho de un árbol.
- esVacio: Arbin -> Bool. Observadora que devuelve si un árbol binario es vacío.

**Autor**

Marco Antonio Gómez Martín

Implementación dinámica del TAD Arbin utilizando nodos con un puntero al hijo izquierdo y otro al hijo derecho.

Las operaciones son:

- ArbolVacio: -> Arbin. Generadora implementada en el constructor sin parámetros.
- Cons: Arbin, Elem, Arbin -> Arbin. Generadora implementada en un constructor con tres parámetros.
- hijoIz, hijoDr: Arbin - -> Arbin. Observadoras que devuelven el hijo izquiero o derecho de un árbol.
- esVacio: Arbin -> Bool. Observadora que devuelve si un árbol binario es vacío.

**Autor**

Marco Antonio Gómez Martín

### 3.1.2. Documentación del constructor y destructor

#### 3.1.2.1. template<class T > Arbin< T >::Arbin ( ) [inline]

Constructor; operacion ArbolVacio

#### 3.1.2.2. template<class T > Arbin< T >::Arbin ( const Arbin< T > & *iz,* const T & *elem,* const Arbin< T > & *dr* ) [inline]

Constructor; operacion Cons

#### 3.1.2.3. template<class T > Arbin< T >::~Arbin ( ) [inline]

Destructor; elimina la estructura jerárquica de nodos.

#### 3.1.2.4. template<class T > Arbin< T >::Arbin ( const Arbin< T > & *other* ) [inline]

Constructor copia

#### 3.1.2.5. template<class T> Arbin<T>::Arbin ( Nodo * *raiz* ) [inline, protected]

Constructor protegido que crea un árbol a partir de una estructura jerárquica existente. Esa estructura jerárquica SE COMPARTE, por lo que se añade la referencia. Se utiliza en hijoIz e hijoDr.

#### 3.1.2.6. template<class T> Arbin<T>::Arbin ( ) [inline]

Constructor; operacion ArbolVacio

#### 3.1.2.7. template<class T> Arbin<T>::Arbin ( const Arbin<T> & *iz,* const T & *elem,* const Arbin<T> & *dr* ) [inline]

Constructor; operacion Cons

#### 3.1.2.8. template<class T> Arbin<T>::~Arbin ( ) [inline]

Destructor; elimina la estructura jerárquica de nodos.

#### 3.1.2.9. template<class T> Arbin<T>::Arbin ( const Arbin<T> & *other* ) [inline]

Constructor copia

#### 3.1.2.10. template<class T> Arbin<T>::Arbin ( Nodo * *raiz* ) [inline, protected]

Constructor protegido que crea un árbol a partir de una estructura jerárquica de nodos previamente creada. Se utiliza en hijoIz e hijoDr.

### 3.1.3. Documentación de las funciones miembro

#### 3.1.3.1. template<class T> template<class T> static Arbin<T> Arbin<T>::construyeYVacia ( Arbin<T> & *iz,* const T & *elem,* Arbin<T> & *dr* ) [inline, static]

Otra operación generadora (estática) que evita las copias vaciando los árboles que recibe.

#### 3.1.3.2. template<class T> bool Arbin<T>::esVacio ( ) const [inline]

Operación observadora que devuelve si el árbol es vacío (no contiene elementos) o no.

esVacio(ArbolVacio) = true esVacio(Cons(iz, elem, dr)) = false

#### 3.1.3.3. template<class T > bool Arbin< T >::esVacio ( ) const [inline]

Operación observadora que devuelve si el árbol es vacío (no contiene elementos) o no.

esVacio(ArbolVacio) = true esVacio(Cons(iz, elem, dr)) = false

#### 3.1.3.4. template<class T > Arbin Arbin< T >::hijoDr ( ) const [inline]

Devuelve un árbol copia del árbol derecho. Es una operación parcial (falla con el árbol vacío).

hijoDr(Cons(iz, elem, dr)) = dr error hijoDr(ArbolVacio)

#### 3.1.3.5. template<class T > Arbin Arbin< T >::hijoDr ( ) const [inline]

Devuelve un árbol copia del árbol derecho. Es una operación parcial (falla con el árbol vacío).

hijoDr(Cons(iz, elem, dr)) = dr error hijoDr(ArbolVacio)

#### 3.1.3.6. template<class T > Arbin Arbin< T >::hijoIz ( ) const [inline]

Devuelve un árbol copia del árbol izquierdo. Es una operación parcial (falla con el árbol vacío).

hijoIz(Cons(iz, elem, dr)) = iz error hijoIz(ArbolVacio)

#### 3.1.3.7. template<class T > Arbin Arbin< T >::hijoIz ( ) const [inline]

Devuelve un árbol copia del árbol izquierdo. Es una operación parcial (falla con el árbol vacío).

hijoIz(Cons(iz, elem, dr)) = iz error hijoIz(ArbolVacio)

#### 3.1.3.8. template<class T > unsigned int Arbin< T >::numHojas ( ) const [inline]

Devuelve el número de hojas de un árbol.

#### 3.1.3.9. template<class T > unsigned int Arbin< T >::numHojas ( ) const [inline]

Devuelve el número de hojas de un árbol.

#### 3.1.3.10. template<class T > unsigned int Arbin< T >::numNodos ( ) const [inline]

Devuelve el número de nodos de un árbol.

#### 3.1.3.11. template<class T > unsigned int Arbin< T >::numNodos ( ) const [inline]

Devuelve el número de nodos de un árbol.

#### 3.1.3.12. template<class T > Arbin<T>& Arbin< T >::operator= ( const Arbin< T > & *other* ) [inline]

Operador de asignacin

#### 3.1.3.13. template<class T > Arbin<T>& Arbin< T >::operator= ( const Arbin< T > & *other* ) [inline]

Operador de asignacin

#### 3.1.3.14. template<class T > bool Arbin< T >::operator== ( const Arbin< T > & *rhs* ) const [inline]

Operador de comparación.

#### 3.1.3.15. template<class T > bool Arbin< T >::operator== ( const Arbin< T > & *rhs* ) const [inline]

Operador de comparación.

#### 3.1.3.16. template<class T > const T& Arbin< T >::raiz ( ) const [inline]

Devuelve el elemento almacenado en la raiz

raiz(Cons(iz, elem, dr)) = elem error raiz(ArbolVacio)

**Devuelve**

Elemento en la raíz.

#### 3.1.3.17. template<class T > const T& Arbin< T >::raiz ( ) const [inline]

Devuelve el elemento almacenado en la raiz

raiz(Cons(iz, elem, dr)) = elem error raiz(ArbolVacio)

**Devuelve**

Elemento en la raíz.

#### 3.1.3.18. template<class T > unsigned int Arbin< T >::talla ( ) const [inline]

Devuelve la talla del árbol.

#### 3.1.3.19. template<class T > unsigned int Arbin< T >::talla ( ) const [inline]

Devuelve la talla del rbol.

### 3.1.4. Documentación de los datos miembro

#### 3.1.4.1. template<class T > Nodo * Arbin< T >::_ra [protected]

Puntero a la raíz de la estructura jerárquica de nodos.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- include/Arbin.h
- include/ArbinConCopias.h

## 3.2. Referencia de la plantilla de la Clase Arbus< Clave, Valor >

`#include <Arbus.h>`

### Clases

- class Iterador

### Métodos públicos

- Arbus ()
- ~Arbus ()
- void inserta (const Clave &clave, const Valor &valor)
- void borra (const Clave &clave)
- const Valor & consulta (const Clave &clave)
- bool esta (const Clave &clave)
- bool esVacio () const
- Iterador principio ()
- Iterador final () const
- Arbus (const Arbus< Clave, Valor > &other)
- Arbus< Clave, Valor > & operator= (const Arbus< Clave, Valor > &other)

### Métodos protegidos

- Arbus (Nodo *raiz)

### 3.2.1. Descripción detallada

**template<class Clave, class Valor>class Arbus< Clave, Valor >**

Implementacinca del TAD Arbus utilizando nodos con un puntero al hijo izquierdo y otro al hijo derecho.

Las operaciones son:

- ArbusVacio: operacineradora que construye un ol de bsqueda vac
- Inserta(clave, valor): generadora que a una nueva pareja (clave, valor) al ol. Si la clave ya estaba se sustituye el valor.
- borra(clave): operacidificadora. Elimina la clave del ol de bsqueda. Si la clave no esta operaci tiene efecto.
- consulta(clave): operaciservadora que devuelve el valor asociado a una clave. Es un error preguntar por una clave que no existe.
- esta(clave): operaciservadora. Sirve para averiguar si se ha introducido una clave en el ol.
- esVacio(): operacion observadora que indica si el ol de bsqueda tiene alguna clave introducida.

**Autor**

Marco Antonio G MartDoxyAuthor

### 3.2.2. Documentación del constructor y destructor

#### 3.2.2.1. template<class Clave , class Valor > Arbus< Clave, Valor >::Arbus ( ) [inline]

Constructor; operacion ArbolVacio

#### 3.2.2.2. template<class Clave , class Valor > Arbus< Clave, Valor >::~Arbus ( ) [inline]

Destructor; elimina la estructura jeruica de nodos.

#### 3.2.2.3. template<class Clave , class Valor > Arbus< Clave, Valor >::Arbus ( const Arbus< Clave, Valor > & *other* ) [inline]

Constructor copia

#### 3.2.2.4. template<class Clave , class Valor > Arbus< Clave, Valor >::Arbus ( Nodo * *raiz* ) [inline, protected]

Constructor protegido que crea un ol a partir de una estructura jeruica de nodos previamente creada. Se utiliza en hijoIz e hijoDr.

### 3.2.3. Documentación de las funciones miembro

#### 3.2.3.1. template<class Clave , class Valor > void Arbus< Clave, Valor >::borra ( const Clave & *clave* ) [inline]

Operacidificadora que elimina una clave del ol. Si la clave no exista operaci tiene efecto.
borra(elem, ArbusVacio) = ArbusVacio borra(e, inserta(c, v, arbol)) = inserta(c, v, borra(e, arbol)) si c != e borra(e, inserta(c, v, arbol)) = borra(e, arbol) si c == e

**Parámetros**

| | |
|---|---|
| *clave* | Clave a eliminar. |

#### 3.2.3.2. template<class Clave , class Valor > const Valor& Arbus< Clave, Valor >::consulta ( const Clave & *clave* ) [inline]

Operaciservadora que devuelve el valor asociado a una clave dada.
consulta(e, inserta(c, v, arbol)) = v si e == c consulta(e, inserta(c, v, arbol)) = consulta(e, arbol) si e != c error consulta(ArbusVacio)

**Parámetros**

| | |
|---|---|
| *clave* | Clave por la que se pregunta. |

#### 3.2.3.3. template<class Clave , class Valor > bool Arbus< Clave, Valor >::esta ( const Clave & *clave* ) [inline]

Operaciservadora que permite averiguar si una clave determinada esto en el ol de bsqueda.
esta(e, ArbusVacio) = false esta(e, inserta(c, v, arbol)) = true si e == c esta(e, inserta(c, v, arbol)) = esta(e, arbol) si e != c

**Parámetros**

| | |
|---|---|
| *clave* | Clave por la que se pregunta. |

#### 3.2.3.4. template<class Clave , class Valor > bool Arbus< Clave, Valor >::esVacio ( ) const [inline]

Operaciservadora que devuelve si el ol es vacno contiene elementos) o no.
esVacio(ArbusVacio) = true esVacio(inserta(c, v, arbol)) = false

#### 3.2.3.5. template<class Clave , class Valor > Iterador Arbus< Clave, Valor >::final ( ) const [inline]

**Devuelve**

Devuelve un iterador al final del recorrido (fuera de e).

#### 3.2.3.6. template<class Clave , class Valor > void Arbus< Clave, Valor >::inserta ( const Clave & *clave,* const Valor & *valor* ) [inline]

Operacineradora que a una nueva clave/valor a un ol de bsqueda.

**Parámetros**

| | |
|---|---|
| *clave* | Clave nueva. |
| *valor* | Valor asociado a esa clave. Si la clave ya se habnsertado previamente, sustituimos el valor viejo por el nuevo. |

#### 3.2.3.7. template<class Clave , class Valor > Arbus<Clave, Valor>& Arbus< Clave, Valor >::operator= ( const Arbus< Clave, Valor > & *other* ) [inline]

Operador de asignaci

#### 3.2.3.8. template<class Clave , class Valor > Iterador Arbus< Clave, Valor >::principio ( ) [inline]

Devuelve el iterador al principio de la lista.

**Devuelve**

iterador al principio de la lista; coincidirn final() si la lista estc

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/Arbus.h

## 3.3. Referencia de la plantilla de la Clase Cola< T >

`#include <Cola.h>`

### Métodos públicos

- Cola ()
- ~Cola ()
- void ponDetras (const T &elem)
- void quitaPrim ()
- const T & primero () const
- bool esVacia () const
- int numElems () const
- Cola (const Cola< T > &other)
- Cola< T > & operator= (const Cola< T > &other)
- bool operator== (const Cola< T > &rhs) const

### 3.3.1. Descripción detallada

**template<class T>class Cola< T >**

Implementacil TAD Cola utilizando una lista enlazada.
Las operaciones son:

- ColaVacia: -> Cola. Generadora implementada en el constructor sin partros.
- PonDetras: Cola, Elem -> Cola. Generadora
- quitaPrim: Cola - -> Cola. Modificadora parcial.
- primero: Cola - -> Elem. Observadora parcial.
- esVacia: Cola -> Bool. Observadora.
- numElems: Cola -> Entero. Observadora.

**Autor**

Marco Antonio G MartDoxyAuthor

### 3.3.2. Documentación del constructor y destructor

#### 3.3.2.1. template<class T > Cola< T >::Cola ( ) [inline]

Constructor; operacion ColaVacia

#### 3.3.2.2. template<class T > Cola< T >::~Cola ( ) [inline]

Destructor; elimina la lista enlazada.

#### 3.3.2.3. template<class T > Cola< T >::Cola ( const Cola< T > & *other* ) [inline]

Constructor copia

### 3.3.3. Documentación de las funciones miembro

#### 3.3.3.1. template<class T > bool Cola< T >::esVacia ( ) const [inline]

Devuelve true si la cola no tiene ningn elemento.
esVacia(Cola) = true esVacia(PonDetras(elem, p)) = false

**Devuelve**

true si la cola no tiene ningn elemento.

#### 3.3.3.2. template<class T > int Cola< T >::numElems ( ) const [inline]

Devuelve el nmero de elementos que hay en la cola. numElems(ColaVacia) = 0 numElems(PonDetras(elem, p)) = 1 + numElems(p)

**Devuelve**

Nmero de elementos.

#### 3.3.3.3. template<class T > Cola<T>& Cola< T >::operator= ( const Cola< T > & *other* ) [inline]

Operador de asignaci

#### 3.3.3.4. template<class T > bool Cola< T >::operator== ( const Cola< T > & *rhs* ) const [inline]

Operador de comparaciclass$_{c}ola_{a}a532c750d29cc5b7d552a8aaf074751a$

#### 3.3.3.5. template<class T > void Cola< T >::ponDetras ( const T & *elem* ) [inline]

A un elemento en la parte trasera de la cola. Operacineradora.

**Parámetros**

| | |
|---|---|
| *elem* | Elemento a ar. |

#### 3.3.3.6. template<class T > const T& Cola< T >::primero ( ) const [inline]

Devuelve el primer elemento de la cola. Operaciservadora parcial, que falla si la cola estc
primero(PonDetras(elem, ColaVacia)) = elem primero(PonDetras(elem, xs)) = primero(xs) si !esVacia(xs) error: primero(ColaVacia)

**Devuelve**

El primer elemento de la cola.

#### 3.3.3.7. template<class T > void Cola< T >::quitaPrim ( ) [inline]

Elimina el primer elemento de la cola. Operacidificadora parcial, que falla si la cola estc
quitaPrim(PonDetras(elem, ColaVacia)) = ColaVacia quitaPrim(PonDetras(elem, xs)) = PonDetras(elem, quitaPrim(xs)) si !esVacia(xs) error: quitaPrim(ColaVacia)
La documentación para esta clase fue generada a partir del siguiente fichero:

- include/Cola.h

## 3.4. Referencia de la plantilla de la Clase DCola< T >

`#include <DCola.h>`

### Métodos públicos

- DCola ()
- ~DCola ()
- void ponDetras (const T &e)
- const T & primero () const
- void quitaPrim ()
- void ponDelante (const T &e)
- const T & ultimo () const

- void quitaUlt ()
- bool esVacia () const
- int numElems () const
- DCola (const DCola< T > &other)
- DCola< T > & operator= (const DCola< T > &other)
- bool operator== (const DCola< T > &rhs) const

### 3.4.1. Descripción detallada

**template<class T>class DCola< T >**

Implementacil TAD Doble Cola utilizando una lista doblemente enlazada circular y con nodo fantasma.
Las operaciones son:

- DColaVacia: -> DCola. Generadora implementada en el constructor sin partros.
- PonDetras: DCola, Elem -> DCola. Generadora
- ponDelante: DCola, Elem -> DCola. Modificadora.
- quitaPrim: DCola - -> DCola. Modificadora parcial
- primero: DCola - -> Elem. Observadora parcial
- quitaUlt: DCola - -> DCola. Modificadora parcial
- ultimo: DCola - -> Elem. Observadora parcial
- esVacia: DCola -> Bool. Observadora

**Autor**

Marco Antonio G MartDoxyAuthor

### 3.4.2. Documentación del constructor y destructor

#### 3.4.2.1. template<class T > DCola< T >::DCola ( ) [inline]

Constructor; operaciColaVacia.

#### 3.4.2.2. template<class T > DCola< T >::~DCola ( ) [inline]

Destructor; elimina la lista doblemente enlazada.

#### 3.4.2.3. template<class T > DCola< T >::DCola ( const DCola< T > & *other* ) [inline]

Constructor copia

### 3.4.3. Documentación de las funciones miembro

#### 3.4.3.1. template<class T > bool DCola< T >::esVacia ( ) const [inline]

Operaciservadora para saber si una doble cola tiene o no elementos.
esVacia(DColaVacia) = true esVacia(ponDetras(x, xs)) = false

**Devuelve**

true si la doble cola no tiene elementos.

#### 3.4.3.2. template<class T> int DCola<T>::numElems ( ) const [inline]

Devuelve el nmero de elementos que hay en la doble cola. numElems(DColaVacia) = 0 numElems(PonDetras(elem, p)) = 1 + numElems(p)

**Devuelve**

Nmero de elementos.

#### 3.4.3.3. template<class T> DCola<T>& DCola<T>::operator= ( const DCola<T> & *other* ) [inline]

Operador de asignaci

#### 3.4.3.4. template<class T> bool DCola<T>::operator== ( const DCola<T> & *rhs* ) const [inline]

Operador de comparaciclass$_{dc}ola_a7eaf1f7f39303c8dc8fc5303e7eb30d1$

#### 3.4.3.5. template<class T> void DCola<T>::ponDelante ( const T & *e* ) [inline]

A un elemento a la parte delantera de una doble cola. Operacidificadora. ponDelante(elem, DColaVacia) = ponDetras(elem, DColaVacia) ponDelante(elem, ponDetras(x, xs)) = ponDetras(x, ponDelante(elem, xs))

**Parámetros**

| | |
|---|---|
| *e* | Elemento que se a |

#### 3.4.3.6. template<class T> void DCola<T>::ponDetras ( const T & *e* ) [inline]

A un elemento por la parte de atre la cola. Es una operacineradora.

#### 3.4.3.7. template<class T> const T& DCola<T>::primero ( ) const [inline]

Devuelve el primer elemento de la cola; es una operaciservadora parcial, pues es un error preguntar por el primer elemento de una doble cola vac primero(PonDetras(elem, DColaVacia)) = elem primero(PonDetras(elem, xs)) = primero(xs) si !esVacia(xs) error: primero(DColaVacia)

#### 3.4.3.8. template<class T> void DCola<T>::quitaPrim ( ) [inline]

Elimina el primer elemento de la doble cola. Operacidificadora parcial, que falla si estc

quitaPrim(PonDetras(elem, DColaVacia)) = DColaVacia quitaPrim(PonDetras(elem, xs)) = PonDetras(elem, quitaPrim(xs)) si !esVacia(xs) error: quitaPrim(DColaVacia)

#### 3.4.3.9. template<class T > void DCola< T >::quitaUlt ( ) [inline]

Elimina el ltimo elemento de la doble cola. Es un error quitar el ltimo de una doble cola vac
quitaUlt(PonDetras(x, xs)) = xs error: quitaUlt(DColaVacia)

#### 3.4.3.10. template<class T > const T& DCola< T >::ultimo ( ) const [inline]

Devuelve el ltimo elemento de la doble cola. Es un error preguntar por el ltimo de una doble cola vac
ultimo(PonDetras(x, xs)) = x error: ultimo(DColaVacia)

**Devuelve**

timo elemento de la cola.

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/DCola.h

## 3.5. Referencia de la Clase ExcepcionTAD

`#include <Excepciones.h>`

### 3.5.1. Descripción detallada

Clase de la que heredan todas las excepciones, y que proporciona el atributo que almacena el mensaje de error.
La documentación para esta clase fue generada a partir del siguiente fichero:

- include/Excepciones.h

## 3.6. Referencia de la Clase Arbus< Clave, Valor >::Iterador

`#include <Arbus.h>`

### Métodos protegidos

- Nodo * primeroInOrden (Nodo *p)

### Amigas

- class **Arbus**

### 3.6.1. Descripción detallada

**template<class Clave, class Valor>class Arbus< Clave, Valor >::Iterador**

Clase interna que implementa un iterador sobre la lista que permite recorrer la lista e incluso alterar el valor de sus elementos.

### 3.6.2. Documentación de las funciones miembro

#### 3.6.2.1. template<class Clave , class Valor > Nodo* Arbus< Clave, Valor >::Iterador::primeroInOrden ( Nodo * *p* ) [inline, protected]

Busca el primer elemento en inorden de la estructura jeruica de nodos pasada como partro; va apilando sus ascendientes para poder "ir hacia atr‘˜ cuando sea necesario.

**Parámetros**

| | |
|---|---|
| *p* | Puntero a la rae la subestructura. |

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/Arbus.h

## 3.7. Referencia de la Clase Tabla< C, V >::Iterador

`#include <Tabla.h>`

### Amigas

- class **Tabla**

### 3.7.1. Descripción detallada

**template<class C, class V>class Tabla< C, V >::Iterador**

Clase interna que implementa un iterador sobre el conjunto de pares (clave, valor). Es importante tener en cuenta que el iterador puede devolver el conunto de pares en cualquier orden.
La documentación para esta clase fue generada a partir del siguiente fichero:

- include/Tabla.h

## 3.8. Referencia de la Clase Lista< T >::Iterador

`#include <Lista.h>`

### Amigas

- class **Lista**

### 3.8.1. Descripción detallada

**template<class T>class Lista< T >::Iterador**

Clase interna que implementa un iterador sobre la lista que permite recorrer la lista e incluso alterar el valor de sus elementos.
La documentación para esta clase fue generada a partir del siguiente fichero:

- include/Lista.h

## 3.9. Referencia de la plantilla de la Clase Lista< T >

`#include <Lista.h>`

### Clases

- class Iterador

### Métodos públicos

- Lista ()
- ~Lista ()
- void Cons (const T &elem)
- void ponDr (const T &elem)
- const T & primero () const
- const T & ultimo () const
- void resto ()
- void inicio ()
- bool esVacia () const
- unsigned int numElems () const
- const T & elem (unsigned int idx) const
- Iterador principio ()
- Iterador final () const
- Iterador borra (const Iterador &it)
- void insertar (const T &elem, const Iterador &it)
- Lista (const Lista< T > &other)
- Lista< T > & operator= (const Lista< T > &other)
- bool operator== (const Lista< T > &rhs) const

### 3.9.1. Descripción detallada

**template<class T>class Lista< T >**

Implementacil TAD Pila utilizando vectores dincos.
Las operaciones son:

- ListaVacia: -> Lista. Generadora implementada en el constructor sin partros.
- Cons: Lista, Elem -> Lista. Generadora.
- ponDr: Lista, Elem -> Lista. Modificadora.
- primero: Lista - -> Elem. Observadora parcial
- resto: Lista - -> Lista. Modificadora parcial
- ultimo: Lista - -> Elem. Observadora parcial
- inicio: Lista - -> Lista. Modificadora parcial
- esVacia: Lista -> Bool. Observadora
- numElems: Lista -> Elem. Obervadora.
- elem: Lista, Entero - -> Elem. Observador parcial.

**Autor**

Marco Antonio G MartDoxyAuthor

### 3.9.2. Documentación del constructor y destructor

#### 3.9.2.1. template<class T > Lista< T >::Lista ( ) [inline]

Constructor; operaciistaVacia.

#### 3.9.2.2. template<class T > Lista< T >::~Lista ( ) [inline]

Destructor; elimina la lista doblemente enlazada.

#### 3.9.2.3. template<class T > Lista< T >::Lista ( const Lista< T > & *other* ) [inline]

Constructor copia

### 3.9.3. Documentación de las funciones miembro

#### 3.9.3.1. template<class T > Iterador Lista< T >::borra ( const Iterador & *it* ) [inline]

Permite eliminar de la lista el elemento apuntado por el iterador que se pasa como partro. El iterador recibido DEJA DE SER VLIDO. - En su lugar, deberilizarse el iterador devuelto, que apuntar siguiente elemento al borrado.

**Parámetros**

| | |
|---|---|
| *it* | Iterador colocado en el elemento que se quiere borrar. |

**Devuelve**

Nuevo iterador colocado en el elemento siguiente al borrado (podroincidir con final() si el elemento que se borr el ltimo de la lista).

#### 3.9.3.2. template<class T > void Lista< T >::Cons ( const T & *elem* ) [inline]

A un nuevo elemento en la cabeza de la lista. Operacineradora.

**Parámetros**

| | |
|---|---|
| *elem* | Elemento que se a en la cabecera de la lista. |

#### 3.9.3.3. template<class T > const T& Lista< T >::elem ( unsigned int *idx* ) const [inline]

Devuelve el elemento i-mo de la lista, teniendo en cuenta que el primer elemento (primero()) es el elemento 0 y el ltimo es numElems()-1, es decir idx est [0..numElems()-1]. Operaciservadora parcial que puede fallar si se da un ice incorrecto. El ice es entero sin signo, para

evitar que se puedan pedir elementos negativos.
elem(0, Cons(x, xs)) = x elem(n, Cons(x, xs)) = elem(n-1, xs) si n > 0
error elem(n, xs) si !( 0 <= n < numElems(xs) )

#### 3.9.3.4. template<class T > bool Lista< T >::esVacia ( ) const [inline]

Operaciservadora para saber si una lista tiene o no elementos.
esVacia(ListaVacia) = true esVacia(Cons(x, xs)) = false

**Devuelve**

true si la lista no tiene elementos.

#### 3.9.3.5. template<class T > Iterador Lista< T >::final ( ) const [inline]

**Devuelve**

Devuelve un iterador al final del recorrido (fuera de e).

#### 3.9.3.6. template<class T > void Lista< T >::inicio ( ) [inline]

Elimina el ltimo elemento de la lista. Es un error intentar obtener el inicio de una lista vac
inicio(Cons(x, ListaVacia)) = ListaVacia inicio(Cons(x, xs)) = Cons(x, inicio(xs)) SI !esVacia(xs) error inicio(ListaVacia)

#### 3.9.3.7. template<class T > void Lista< T >::insertar ( const T & *elem,* const Iterador & *it* ) [inline]

Mdo para insertar un elemento en la lista en el punto marcado por el iterador. En concreto, se a _justo antes_ que el elemento actual. Es decir, si it==l.primero(), el elemento insertado se convierte en el primer elemento (y el iterador apuntar segundo). Si it==l.final(), el elemento insertado ser ltimo (e it seguiruntando fuera del recorrido).

**Parámetros**

| | |
|---|---|
| *elem* | Valor del elemento a insertar. |
| *it* | Punto en el que insertar el elemento. |

#### 3.9.3.8. template<class T > unsigned int Lista< T >::numElems ( ) const [inline]

Devuelve el nmero de elementos que hay en la lista. numElems(ListaVacia) = 0 numElems(Cons(x, xs)) = 1 + numElems(xs)

**Devuelve**

Nmero de elementos.

#### 3.9.3.9. template<class T > Lista<T>& Lista< T >::operator= ( const Lista< T > & *other* ) [inline]

Operador de asignaci

#### 3.9.3.10. template<class T > bool Lista< T >::operator== ( const Lista< T > & *rhs* ) const [inline]

Operador de comparaciclass$_{l}ista_{a}c7c8b711e925f3ba8e7f3aa6e45a05c9$

#### 3.9.3.11. template<class T > void Lista< T >::ponDr ( const T & *elem* ) [inline]

A un nuevo elemento al final de la lista (a la "derecha"). - Operacidificadora.
ponDr(e, ListaVacia) = Cons(e, ListaVacia) ponDr(e, Cons(x, xs)) = Cons(x, ponDr(e, xs))

#### 3.9.3.12. template<class T > const T& Lista< T >::primero ( ) const [inline]

Devuelve el valor almacenado en la cabecera de la lista. Es un error preguntar por el primero de una lista vac
primero(Cons(x, xs)) = x error primero(ListaVacia)

**Devuelve**

Elemento en la cabecera de la lista.

#### 3.9.3.13. template<class T > Iterador Lista< T >::principio ( ) [inline]

Devuelve el iterador al principio de la lista.

**Devuelve**

iterador al principio de la lista; coincidirn final() si la lista estc

#### 3.9.3.14. template<class T > void Lista< T >::resto ( ) [inline]

Elimina el primer elemento de la lista. Es un error intentar obtener el resto de una lista vac
resto(Cons(x, xs)) = xs error resto(ListaVacia)

#### 3.9.3.15. template<class T > const T& Lista< T >::ultimo ( ) const [inline]

Devuelve el valor almacenado en la ltima posici la lista (a la derecha). Es un error preguntar por el primero de una lista vac
ultimo(Cons(x, xs)) = x SI esVacia(xs) ultimo(Cons(x, xs)) = ultimo(xs) SI !esVacia(xs) error ultimo(ListaVacia)

**Devuelve**

Elemento en la cola de la lista.

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/Lista.h

## 3.10. Referencia de la Clase Arbin< T >::Nodo

`#include <ArbinConCopias.h>`

### 3.10.1. Descripción detallada

**template<class T>class Arbin< T >::Nodo**

Clase nodo que almacena internamente el elemento (de tipo T), y los punteros al hijo izquierdo y al hijo derecho, así como el número de referencias que hay.
Clase nodo que almacena internamente el elemento (de tipo T), y los punteros al hijo izquierdo y al hijo derecho.
La documentación para esta clase fue generada a partir de los siguientes ficheros:

- include/Arbin.h
- include/ArbinConCopias.h

## 3.11. Referencia de la plantilla de la Clase Pila< T >

`#include <Pila.h>`

### Tipos públicos

- enum

### Métodos públicos

- Pila ()
- ~Pila ()
- void apila (const T &elem)
- void desapila ()
- const T & cima () const
- bool esVacia () const
- int numElems () const
- Pila (const Pila< T > &other)
- Pila< T > & operator= (const Pila< T > &other)
- bool operator== (const Pila< T > &rhs) const

### 3.11.1. Descripción detallada

**template<class T>class Pila< T >**

Implementacil TAD Pila utilizando vectores dincos.
Las operaciones son:

- PilaVacia: -> Pila. Generadora implementada en el constructor sin partros.
- apila: Pila, Elem -> Pila. Generadora
- desapila: Pila - -> Pila. Modificadora parcial.
- cima: Pila - -> Elem. Observadora parcial.
- esVacia: Pila -> Bool. Observadora.
- numElems: Pila -> Entero. Observadora.

**Autor**

Marco Antonio G MartDoxyAuthor

### 3.11.2. Documentación de las enumeraciones miembro de la clase

#### 3.11.2.1. template<class T> anonymous enum

Tamaicial del vector dinco.

### 3.11.3. Documentación del constructor y destructor

#### 3.11.3.1. template<class T> Pila< T >::Pila ( ) [inline]

Constructor; operaciilaVacia

#### 3.11.3.2. template<class T> Pila< T >::~Pila ( ) [inline]

Destructor; elimina el vector.

#### 3.11.3.3. template<class T> Pila< T >::Pila ( const Pila< T > & *other* ) [inline]

Constructor copia

### 3.11.4. Documentación de las funciones miembro

#### 3.11.4.1. template<class T> void Pila< T >::apila ( const T & *elem* ) [inline]

Apila un elemento. Operacineradora.

**Parámetros**

| | |
|---|---|
| *elem* | Elemento a apilar. |

#### 3.11.4.2. template<class T> const T& Pila< T >::cima ( ) const [inline]

Devuelve el elemento en la cima de la pila. Operaciservadora parcial, que falla si la pila estc
cima(Apila(elem, p) = elem error: cima(PilaVacia)

**Devuelve**

Elemento en la cima de la pila.

#### 3.11.4.3. template<class T> void Pila< T >::desapila ( ) [inline]

Desapila un elemento. Operacidificadora parcial, que falla si la pila estc
desapila(Apila(elem, p)) = p error: desapila(PilaVacia)

#### 3.11.4.4. template<class T> bool Pila< T >::esVacia ( ) const [inline]

Devuelve true si la pila no tiene ningn elemento.
esVacia(PilaVacia) = true esVacia(Apila(elem, p)) = false
**Devuelve**

true si la pila no tiene ningn elemento.

#### 3.11.4.5. template<class T> int Pila< T >::numElems ( ) const [inline]

Devuelve el nmero de elementos que hay en la pila. numElems(-PilaVacia) = 0 numElems(Apila(elem, p)) = 1 + numElems(p)
**Devuelve**

Nmero de elementos.

#### 3.11.4.6. template<class T> Pila<T>& Pila< T >::operator= ( const Pila< T > & *other* ) [inline]

Operador de asignaci

#### 3.11.4.7. template<class T> bool Pila< T >::operator== ( const Pila< T > & *rhs* ) const [inline]

Operador de comparaci
La documentación para esta clase fue generada a partir del siguiente fichero:

- include/Pila.h

## 3.12. Referencia de la plantilla de la Clase PilaLE< T >

`#include <PilaLE.h>`

### Métodos públicos

- PilaLE ()
- ~PilaLE ()
- void apila (const T &elem)
- void desapila ()
- const T & cima () const
- bool esVacia () const

- int numElems () const
- PilaLE (const PilaLE< T > &other)
- PilaLE< T > & operator= (const PilaLE< T > &other)
- bool operator== (const PilaLE< T > &rhs) const

### 3.12.1. Descripción detallada

**template<class T>class PilaLE< T >**

Implementacil TAD Pila utilizando vectores dincos.
Las operaciones son:

- PilaVacia: -> Pila. Generadora implementada en el constructor sin partros.
- apila: Pila, Elem -> Pila. Generadora
- desapila: Pila - -> Pila. Modificadora parcial.
- cima: Pila - -> Elem. Observadora parcial.
- esVacia: Pila -> Bool. Observadora.
- numElems: Pila -> Entero. Observadora.

**Autor**

Marco Antonio G MartDoxyAuthor

### 3.12.2. Documentación del constructor y destructor

#### 3.12.2.1. template<class T > PilaLE< T >::PilaLE ( ) [inline]

Constructor; operaciilaVacia

#### 3.12.2.2. template<class T > PilaLE< T >::~PilaLE ( ) [inline]

Destructor; elimina la lista enlazada.

#### 3.12.2.3. template<class T > PilaLE< T >::PilaLE ( const PilaLE< T > & *other* ) [inline]

Constructor copia

### 3.12.3. Documentación de las funciones miembro

#### 3.12.3.1. template<class T > void PilaLE< T >::apila ( const T & *elem* ) [inline]

Apila un elemento. Operacineradora.

**Parámetros**

| | |
|---|---|
| *elem* | Elemento a apilar. |

#### 3.12.3.2. template<class T > const T& PilaLE< T >::cima ( ) const [inline]

Devuelve el elemento en la cima de la pila. Operaciservadora parcial, que falla si la pila estc
cima(Apila(elem, p) = elem error: cima(PilaVacia)

**Devuelve**

Elemento en la cima de la pila.

#### 3.12.3.3. template<class T > void PilaLE< T >::desapila ( ) [inline]

Desapila un elemento. Operacidificadora parcial, que falla si la pila estc
desapila(Apila(elem, p) = p error: desapila(PilaVacia)

#### 3.12.3.4. template<class T > bool PilaLE< T >::esVacia ( ) const [inline]

Devuelve true si la pila no tiene ningn elemento.
esVacia(PilaVacia) = true esVacia(Apila(elem, p)) = false

**Devuelve**

true si la pila no tiene ningn elemento.

#### 3.12.3.5. template<class T > int PilaLE< T >::numElems ( ) const [inline]

Devuelve el nmero de elementos que hay en la pila. numElems(PilaVacia) = 0 numElems(Apila(elem, p)) = 1 + numElems(p)

**Devuelve**

Nmero de elementos.

#### 3.12.3.6. template<class T > PilaLE<T>& PilaLE< T >::operator= ( const PilaLE< T > & *other* ) [inline]

Operador de asignaci

#### 3.12.3.7. template<class T > bool PilaLE< T >::operator== ( const PilaLE< T > & *rhs* ) const [inline]

Operador de comparaci
La documentación para esta clase fue generada a partir del siguiente fichero:

- include/PilaLE.h

## 3.13. Referencia de la plantilla de la Clase - Tabla< C, V >

`#include <Tabla.h>`

### Clases

- class Iterador

### Métodos públicos

- Tabla ()
- ~Tabla ()
- void inserta (const C &clave, const V &valor)
- void borra (const C &clave)
- bool esta (const C &clave)
- const V & consulta (const C &clave)
- bool esVacia ()
- Iterador principio ()
- Iterador final () const
- Tabla (const Tabla< C, V > &other)
- Tabla< C, V > & operator= (const Tabla< C, V > &other)

### Atributos públicos estáticos

- static const int TAM_INICIAL = 10

### 3.13.1. Descripción detallada

**template<class C, class V>class Tabla< C, V >**

Implementacil TAD Tabla usando una tabla hash abierta.
Las operaciones pblicas son:

- TablaVacia: -> Tabla. Generadora (constructor).
- inserta: Tabla, Clave, Valor -> Tabla. Generadora.
- borra: Tabla, Clave -> Tabla. Modificadora.
- esta: Tabla, Clave -> Bool. Observadora.
- consulta: Tabla, Clave - -> Valor. Observadora parcial.
- esVacia: Tabla -> Bool. Observadora.

**Autor**

Antonio Shez Ruiz-Granados

### 3.13.2. Documentación del constructor y destructor

#### 3.13.2.1. template<class C , class V > Tabla< C, V >::Tabla ( ) [inline]

Constructor por defecto. Crea una tabla con TAM_INICIAL posiciones.

#### 3.13.2.2. template<class C , class V > Tabla< C, V >::~Tabla ( ) [inline]

Destructor.

#### 3.13.2.3. template<class C , class V > Tabla< C, V >::Tabla ( const Tabla< C, V > & *other* ) [inline]

Constructor por copia.

**Parámetros**

| | |
|---|---|
| *other* | tabla que se quiere copiar. |

### 3.13.3. Documentación de las funciones miembro

#### 3.13.3.1. template<class C , class V > void Tabla< C, V >::borra ( const C & *clave* ) [inline]

Elimina el elemento de la tabla con la clave dada. Si no existingn elemento con dicha clave, la tabla no se modifica.

**Parámetros**

| | |
|---|---|
| *clave* | clave del elemento a eliminar. |

#### 3.13.3.2. template<class C , class V > const V& Tabla< C, V >::consulta ( const C & *clave* ) [inline]

Devuelve el valor asociado a la clave dada. Si la tabla no contiene esa clave lanza una excepci

**Parámetros**

| | |
|---|---|
| *clave* | clave del elemento a buscar. |

**Devuelve**

valor asociado a dicha clave.

**Excepciones**

| | |
|---|---|
| *EClaveInexistente* | si la clave no existe en la tabla. |

#### 3.13.3.3. template<class C , class V > bool Tabla< C, V >::esta ( const C & *clave* ) [inline]

Comprueba si la tabla contiene algn elemento con la clave dada.

**Parámetros**

| | |
|---|---|
| *clave* | clave a buscar. |

**Devuelve**

si existe algn elemento con esa clave.

#### 3.13.3.4. template<class C , class V > bool Tabla< C, V >::esVacia ( ) [inline]

Indica si la tabla estc es decir, si no contiene ningn elemento.

**Devuelve**

si la tabla estc

#### 3.13.3.5. template<class C , class V > Iterador Tabla< C, V >::final ( ) const [inline]

Devuelve un iterador al final del recorrido (apunta mlll ltimo elemento de la tabla).

**Devuelve**

iterador al final del recorrido.

#### 3.13.3.6. template<class C , class V > void Tabla< C, V >::inserta ( const C & *clave,* const V & *valor* ) [inline]

Inserta un nuevo par (clave, valor) en la tabla. Si ya existn elemento con esa clave, se actualiza su valor.

**Parámetros**

| | |
|---|---|
| *clave* | clave del nuevo elemento. |
| *valor* | valor del nuevo elemento. |

#### 3.13.3.7. template<class C , class V > Tabla<C,V>& Tabla< C, V >::operator= ( const Tabla< C, V > & *other* ) [inline]

Operador de asignaci

**Parámetros**

| | |
|---|---|
| *other* | tabla que se quiere copiar. |

**Devuelve**

referencia a este mismo objeto (∗this).

#### 3.13.3.8. template<class C , class V > Iterador Tabla< C, V >::principio ( ) [inline]

Devuelve un iterador al primer par (clave, valor) de la tabla. El iterador devuelto coincidirn final() si la tabla estc

**Devuelve**

iterador al primer par (clave, valor) de la tabla.

### 3.13.4. Documentación de los datos miembro

#### 3.13.4.1. template<class C , class V > const int Tabla< C, V >::TAM_INICIAL = 10 [static]

Tamaicial de la tabla.

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/Tabla.h

# Capítulo 4

# Documentación de archivos

## 4.1. Referencia del Archivo include/Arbin.h

```
#include "Excepciones.h"      #include "-
Lista.h" #include "Cola.h"
```

### Clases

- class Arbin< T >
- class Arbin< T >::Nodo

#### 4.1.1. Descripción detallada

Implementacin dinmica del TAD Arbol Binario.
Estructura de Datos y Algoritmos Facultad de Informtica - Universidad Complutense de Madrid
(c) Marco Antonio Gmez Martn, 2012

## 4.2. Referencia del Archivo include/Arbus.h

```
#include "Excepciones.h"      #include "-
Lista.h" #include "Pila.h"
```

### Clases

- class Arbus< Clave, Valor >
- class Arbus< Clave, Valor >::Iterador

#### 4.2.1. Descripción detallada

Implementacinca del TAD Arbol de Bsqueda.
Estructura de Datos y Algoritmos Facultad de Informca - Universidad Complutense de Madrid

(c) Marco Antonio G Mart 2012

## 4.3. Referencia del Archivo include/Cola.h

```
#include "Excepciones.h"
```

### Clases

- class Cola< T >

#### 4.3.1. Descripción detallada

Implementacil TAD Cola utilizando una lista enlazada de nodos.
Estructura de Datos y Algoritmos Facultad de Informca - Universidad Complutense de Madrid
(c) Marco Antonio G Mart 2012

## 4.4. Referencia del Archivo include/DCola.h

```
#include "Excepciones.h" #include <cassert> ×
```

### Clases

- class DCola< T >

#### 4.4.1. Descripción detallada

Implementacil TAD doble cola, utilizando una lista doblemente enlazada con nodo fantasma o cabecera.
Estructura de Datos y Algoritmos Facultad de Informca - Universidad Complutense de Madrid
(c) Marco Antonio G Mart 2012

## 4.5. Referencia del Archivo include/Hash.h

```
#include <string>
```

### Funciones

- template<class C >
  unsigned int hash (const C &clave)

### 4.5.1. Descripción detallada

Declaracimplementaci funciones de localizacira tipos bcos y funcinca que confn la existencia del mdo mdo hash de las clases.
Estructura de Datos y Algoritmos Facultad de Informca - Universidad Complutense de Madrid
(c) Antonio Shez Ruiz-Granados, 2012

### 4.5.2. Documentación de las funciones

#### 4.5.2.1. template<class C > unsigned int hash ( const C & *clave* )

Funcish genca para clases que implementen un mdo publico hash.

## 4.6. Referencia del Archivo include/Lista.h

`#include "Excepciones.h"` `#include <cassert>` ×

### Clases

- class Lista< T >
- class Lista< T >::Iterador

### 4.6.1. Descripción detallada

Implementacil TAD lista, utilizando una lista doblemente enlazada.
Estructura de Datos y Algoritmos Facultad de Informca - Universidad Complutense de Madrid
(c) Marco Antonio G Mart 2012

## 4.7. Referencia del Archivo include/Pila.h

`#include "Excepciones.h"`

### Clases

- class Pila< T >

### 4.7.1. Descripción detallada

Implementacil TAD Pila utilizando un vector dinco cuyo tama creciendo si es necesario.
Estructura de Datos y Algoritmos Facultad de Informca - Universidad Complutense de Madrid

(c) Marco Antonio G Mart 2012

## 4.8. Referencia del Archivo include/PilaLE.h

```
#include "Excepciones.h"
```

### Clases

- class PilaLE< T >

### 4.8.1. Descripción detallada

Implementacil TAD Pila utilizando una lista enlazada de nodos.
Estructura de Datos y Algoritmos Facultad de Informca - Universidad Complutense de Madrid
(c) Marco Antonio G Mart 2012