TDA Conjunto: Iteradores

V0

Generado por Doxygen 1.8.11

Índice

1. Iterando sobre el conjunto

Versión

Práctica 5 Iteradores. Ver 0.

Autor

Juan F. Huete y Carlos Cano

1.1. Introducción

En la práctica anterior hemos construido el conjunto genérico (se puede particularizar sobre distintos tipos de datos y distintos criterios de comparación). Aunque estaba dotado de un mecanismo para poder iterar sobre las entradas, este mecanismo lo heredaba directamente de la clase vector subyacente, pudiendo provocar errores en la representación (no hay ocultamiento de información).

Por ejemplo, sería posible hacer

```
conjunto<mutacion,less<mutacion> > X;
mutacion aux1,aux2;
aux1.setPos(1234);
aux1.setChr("MT");
conjunto<mutacion,less<mutacion> >::iterator it;
it = X.find(aux1);
if (it!=X.end())
  {aux2.setPos(4321);
   aux2.setChr("1");
   *it = aux2;  //VIOLAMOS EL INVARIANTE DE LA REPRESENTACION
}
```

En este práctica lo que proponemos es dotar a este conjunto de mecanismos de iteración específicos, siguiendo el estándar que se considera para acceder a los elementos de un contenedor, sin necesidad de conocer las particularidades internas de la implementación.

1.2. Generar la Documentación.

Al igual que en la práctica anterior la documentación se entrega mediante un fichero documentacion.pdf, así como mediante un fichero zip que contiene todos los fuentes junto a los archivos necesarios para generar la documentación (en latex y html). Para generar los ficheros html con la documentación de la misma es suficiente con ejecutar desde la línea de comandos

```
doxygen doxPractica.txt
```

Como resultado nos genera dos directorios, uno con la documentación en html y el otro con la documentación en latex.

Se entregan los ficheros de especificación nueva para el TDA conjunto. Estos ficheros incluyen algunas modificaciones que viene dadas por el uso de los iteradores.

- conjunto.h En el nuevo fichero conjunto.h se entrega la nueva especificación de la clase conjunto, donde además se le añade la especificación de tres iteradores. Se os pide implementar los distintos métodos así como el código necesario para demostrar el correcto funcionamiento del mismo.

Pasamos a detallar cada una de las partes de la práctica.

1.3. Iteradores sobre conjunto.

Casi todos los contenedores disponen de una (o varias) clases asociada llamada iterator. Para poder asociar el iterador al contenedor una alternativa es añadir una clase anidada (una clase que se define dentro de la clase contenedora). Ambas clases están estrechamente relacionadas, por lo que es muy usual que se desee que tanto el contenedor como el iterador sean clases amigas. Así, cuando se crea una clase friend anidada es conveniente declarar primero el nombre de la clase y después definir la clase. Así evitamos que se confunda el compilador.

```
template <typename T, class CMP>
class conjunto{
public:
        typedef T value_type;
    typedef unsigned int size_type;

    class iterator;
    class const_iterator;
    class impar_iterator;  // Itera sobre mutaciones con posición impar
    class secure_iterator; // itera asegurando que las posiciones son correctas dentro del
       vector.
    class const_impar_iterator;  // Itera sobre mutaciones con posición impar
    class const_secure_iterator; // itera asegurando que las posiciones son correctas dentro del vector.
   ....

   impar_iterator ibegin();
   const_impar_iterator cibegin( );
   secure_iterator sbegin( );
   const_secure_iterator csbegin( );
   impar_iterator iend();
   const_impar_iterator ciend( );
   secure_iterator send( );
   const_secure_iterator csend( );


   ....
   class impar_iterator {
      //definicion del iterator
      public:
    impar_iterator();
        ....
      private:
         friend class conjunto<T,CMP>; // declaramos conjunto como amigo de la clase
        ....

    }; // end de la clase iterator

   private:
        friend class impar_iterator; // declaramos el iterador como amigo de la clase

}; // end de la clase conjunto
```

Es importante notar que el tipo asociado al iterador es conjunto<T,CMP>::xxx_iterator Por tanto, para declarar un conjunto y un iterator sobre dicho conjunto debemos hacer

```
conjunto<mutacion,less<mutacion> > C;
conjunto<mutacion,less<mutacion> >::iterator it;
conjunto<mutacion,less<mutacion> >::impar_iterator iit;

for (it = C.begin() ; it!=C.end();++it) //Itera sobre todos los elementos del conjunto.
  cout << *it << endl;

 //iit Itera sobre todos las mutaciones en posiciones impares
for (iit = C.ibegin(); iit!= C.iend();++iit)
  cout << *iit << endl;
```

1.4. begin y end

Para poder iterar sobre los elementos del contenedor, debemos dotarlo de nuevos métodos (que siguiendo en estándar de la Standard Template Library llamaremos begin y end). En sus distintos formatos, begin devuelve un iterator que apunta al primer elemento del contenedor (primer elemento que satisface las condiciones por las que se itera), mientras que end (en sus distintas versiones) por su parte nos devuelve un iterator que apunta «al final» del contenedor. Es importante recordar que la posición final del contenedor no es una posición válida del mismo, esto es, no hay ningún elemento en dicha posición (es conveniente pensar que es la posición siguiente al último elemento del contenedor). Por ello, no es correcto dereferenciar el elemento alojado en dicha posición (∗end()).

Además podemos ver el uso de paréntesis para acceder a los elementos (∗it).getID(). En este caso, si hacemos ∗it.getID, dada la precedencia de los operadores, primero se evaluaría el operador "."

```
(*it).getID() // Correcto
*it.getID() // Incorrecto, primero evalúa it.getID()
```

Además del begin y end que devuelven el iterator, y siguiendo la filosofía del estándar C++11, implementaremos dos métodos, el cbegin y el cend que devuelven los const_iterator

```
conjunto<mutacion,less<mutacion> >::const_iterator cit;
cit = C.cbegin();
```

1.4.1. impar_iterator

En esta práctica debemos destacar el comportamiento de impar_iterator. Dicho iterador nos permitirá iterar sobre todos los elementos que contengan una determinada mutación que está en posiciones impares del cromosoma. Obviamente, este iterador sólo será valido cuando el tipo elemento sobre el que se particulariza el conjunto tenga definido el método getPos() que devuelva un entero. Por tanto, valdría para un conjunto<mutacion,less<mutacion> > pero no para un conjunto<enfermedad,less<enfermedad> >.

```
@brief devolver primera posición del elemento que se encuentra en posiciones impares
@return un iterador que apunta a la primera mutación con posición impar dentro del conjunto,  si no hay
      devuelve iend
conjunto<T;CMP>::impar_iterator
      conjunto<T,CMP>::ibegin( );

@brief devolver final posiciones impares
@return nos debe devolver un iterador que apunta a la posición final del mismo.
conjunto<T;CMP>::impar_iterator conjunto<T,CMP>::iend( );
```

El método iend() puede puede coincidir con el end() del vector<mutaciones>

1.4.2. secure_iterator

Con respecto al secure_iterator, nos permitirá asegurarnos de que el iterador apunte siempre a una posición válida del vector. En caso contrario, el programa aborta. Para su implementación, utilizaremos el método assert (de la biblioteca assert.h). Este chequeo se deberá hacer en todos los métodos que puedan provocar un error, como operator∗, operator++, operator--, etc.

```
#include <assert.h>
....
class secure_iterator{
....
}

....
const T & xxx::secure_iterator::operator*(){
     ....
     assert (posicion correcta); // entre el begin y end del vector al que apunta
     return elemento_en_posicion;
}
```

1.5. Representación del iterator

Un iterator de la clase conjunto nos debe permitir el acceso a los datos almacenados en el conjunto propiamente dicho. Una primera alternativa sería representar el iterator como un iterador sobre el vector, directamente como lo hemos considerado en las prácticas anteriores,

```
class conjunto{
   ....
  typedef  vector<value_type>::iterator iterator;
   ...
```

o implementando todo el iterador como podría ser

```
class conjunto{
....
  class iterator {
    ....
     value_type & operator*(); // NO seria correcto
    ......
    private:
      vector<value_type>::iterator it_v; // Puntero a la entrada del vector.
   };
};
```

Sin embargo, con ambas representaciones sería posible violar el invariante de la representación del TDA conjunto. Así, el usuario de la clase podría modificar el contenido de la clave ejecutando

```
conjunto<mutacion,less<mutacion> > X;
mutacion aux1,aux2;
aux1.setPos(1234);
aux1.setChr("MT");
conjunto<mutacion,less<mutacion> >::iterator it

*(X.begin()) = aux1; //VIOLAMOS INVARIANTE

X.find(aux1);
if (it!=X.end())
  {aux2.setPos(4321);
   aux2.setChr("1");
   *it = aux2;  //VIOLAMOS EL INVARIANTE DE LA REPRESENTACION
}
```

Esto nos daría problemas pues estaríamos modificando la clase, y particularmente, al asumir los datos ordenados, el conjunto podría dejar de estar ordenado por lo que no cumpliría el invariante de la representación. Si el conjunto deja de estar ordenado, las operaciones de búsqueda e inserción dejarían de funcionar correctamente.

Para solucionar el problema es necesario que todos los iteradores del conjunto devuelvan una referencia constante a los elementos almacenados en el mismo

```
class conjunto{

....
  class iterator {
    ....
    const value_type & operator*();
    ......
    private:
    ....
   };
....
  class const_iterator {
     .....
     const value_type & operator*();
     ....
     private:
     ....
};
```

1.6. SE PIDE

En concreto se pide implementar los métodos asociados a los iteradores de la clase conjunto.

En este caso, para realizar la práctica, el alumno deberá modificar los ficheros de implementación (.hxx).

De igual forma, debe entregar un fichero principal.cpp que muestre el correcto comportamiento del conjunto cuando se instancia con distintos tipos y el uso de los distintos tipos de iterators y const_iterators. A modo ilustrativo, se entrega el fichero principal.cpp.

1.6.1. A ENTREGAR

El alumno debe entregar los siguientes ficheros, con las correcciones necesarias para poder trabajar

- documentacion.pdf
- ficheros.zip

Dicha entrega tiene como límite el Domingo 4 de Diciembre.

2. Lista de tareas pendientes

Clase conjunto< T, CMP >
Implementa esta clase siguiendo la especificación asociada

3. Índice de clases

3.1. Lista de clases

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

4. Indice de archivos

4.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

5. Documentación de las clases

5.1. Referencia de la plantilla de la Clase conjunto< T, CMP >

Clase conjunto.

```
#include <conjunto.h>
```

Clases

- class impar_iterator
- class iterator
- class secure_iterator

Tipos públicos

- typedef T value_type
- typedef unsigned int size_type

Métodos públicos

- conjunto ()

  constructor primitivo.
- conjunto (const conjunto< T, CMP > &d)

  constructor de copia
- iterator find (const value_type &s)

  busca una entrada en el conjunto
- const_iterator find (const value_type &s) const
- size_type count (const value_type &e) const

  cuenta cuantas entradas coinciden con los parámetros dados.
- pair< iterator, bool > insert (const value_type &val)

  Inserta una entrada en el conjunto.
- iterator erase (const iterator position)

  Borra una entrada en el conjunto . Busca la entrada y si la encuentra la borra.
- size_type erase (const value_type &val)
- void clear ()

  Borra todas las entradas del conjunto, dejandolo vacio.
- size_type size () const

  numero de entradas en el conjunto
- bool empty () const

  Chequea si el conjunto esta vacio (size()==0)
- conjunto & operator= (const conjunto &org)

  operador de asignación
- iterator begin ()

  begin del conjunto
- const_iterator cbegin () const
- iterator end ()

  end del conjunto
- const_iterator cend () const
- secure_iterator sbegin ()

  begin del conjunto
- const_secure_iterator csbegin () const
- secure_iterator send ()

  end del conjunto
- const_secure_iterator csend () const
- impar_iterator ibegin ()

  begin del conjunto
- const_impar_iterator cibegin () const
- impar_iterator iend ()

  end del conjunto
- const_impar_iterator ciend () const
- iterator lower_bound (const value_type &val)

  busca primer elemento por debajo ('antes', '<') de los parámetros dados.
- const_iterator lower_bound (const value_type &val) const
- iterator upper_bound (const value_type &val)

  busca primer elemento por encima ('después', '>') de los parámetros dados.
- const_iterator upper_bound (const value_type &val) const

Métodos privados

- bool cheq_rep () const

  Chequea el Invariante de la representacion.

Atributos privados

- vector< value_type > vm
- CMP comp

Amigas

- class impar_iterator
- class secure_iterator
- class iterator

5.1.1. Descripción detallada

template<typename T, class CMP>
class conjunto< T, CMP >

Clase conjunto.

conjunto::conjunto, find, size, Tipos conjunto::value_type, conjunto::size_type Iteradores:iterator, impar_iterator, secure_iterator; Descripción

Un conjunto es un contenedor que permite almacenar en orden creciente un conjunto de elementos no repetidos.

Asociado al conjunto, tendremos el tipo

```
conjunto::value_type
```

que permite hacer referencia al elemento almacenados en cada una de las posiciones del conjunto. Es requisito que el tipo conjunto::value_type tenga definidos los operadores operator< y operator= .

El número de elementos en el conjunto puede variar dinámicamente; la gestión de la memoria es automática.

Tareas pendientes Implementa esta clase siguiendo la especificación asociada

5.1.2. Documentación de los 'Typedef' miembros de la clase

5.1.2.1. template<typename T, class CMP> typedef unsigned int conjunto< T, CMP >::size_type

5.1.2.2. template<typename T, class CMP> typedef T conjunto< T, CMP >::value_type

5.1.3. Documentación del constructor y destructor

5.1.3.1. template<typename T, class CMP> conjunto< T, CMP >::conjunto ( )

constructor primitivo.

5.1.3.2. template<typename T, class CMP> conjunto< T, CMP >::conjunto (const conjunto< T, CMP > & *d*)

constructor de copia

Parámetros

in	*d*	conjunto a copiar

5.1.4. Documentación de las funciones miembro

5.1.4.1. template<typename T, class CMP> iterator conjunto< T, CMP >::begin ( )

begin del conjunto

Devuelve

Devuelve un iterador (o iterador constante, respectivamente) al primer elemento del conjunto. Si no existe devuelve end

5.1.4.2. template<typename T, class CMP> const_iterator conjunto< T, CMP >::cbegin ( ) const

5.1.4.3. template<typename T, class CMP> const_iterator conjunto< T, CMP >::cend ( ) const

5.1.4.4. template<typename T, class CMP> bool conjunto< T, CMP >::cheq_rep ( ) const [private]

Chequea el Invariante de la representacion.

Devuelve

true si el invariante es correcto, falso en caso contrario

5.1.4.5. template<typename T, class CMP> const_impar_iterator conjunto< T, CMP >::cibegin ( ) const

5.1.4.6. template<typename T, class CMP> const_impar_iterator conjunto< T, CMP >::ciend ( ) const

5.1.4.7. template<typename T, class CMP> void conjunto< T, CMP >::clear ( )

Borra todas las entradas del conjunto, dejandolo vacio.

Postcondición

El conjunto se modifica, quedando vacio.

5.1.4.8. template<typename T, class CMP> size_type conjunto< T, CMP >::count (const value_type & *e*) const

cuenta cuantas entradas coinciden con los parámetros dados.

Parámetros

in	*e*	entrada.

Devuelve

Como el conjunto de mutaciones no puede tener entradas repetidas, devuelve 1 (si se encuentra la entrada) o 0 (si no se encuentra).

Postcondición

no modifica el conjunto.

5.1.4.9. template<typename T, class CMP> const_secure_iterator conjunto< T, CMP >::csbegin ( ) const

5.1.4.10. template<typename T, class CMP> const_secure_iterator conjunto< T, CMP >::csend ( ) const

5.1.4.11. template<typename T, class CMP> bool conjunto< T, CMP >::empty ( ) const

Chequea si el conjunto esta vacio (size()==0)

Postcondición

No se modifica el conjunto.

5.1.4.12. template<typename T, class CMP> iterator conjunto< T, CMP >::end ( )

end del conjunto

Devuelve

Devuelve un iterador (o iterador constante, respectivamente) al final del conjunto (posicion siguiente al ultimo.

Postcondición

no modifica el conjunto.

5.1.4.13. template<typename T, class CMP> iterator conjunto< T, CMP >::erase (const iterator *position*)

Borra una entrada en el conjunto . Busca la entrada y si la encuentra la borra.

Parámetros

in	*val*	entrada a borrar.
in	*position*	itarador que apunta a la entrada que geremos borrar

Devuelve

devuelve la posicion siguiente al elemento borrado (para la version con iterador) o el numero de elementos borrados

Postcondición

Si esta en el conjunto su tamaño se decrementa en 1.

5.1.4.14. template<typename T, class CMP> size_type conjunto< T, CMP >::erase (const value_type & *val*)

5.1.4.15. template<typename T, class CMP> iterator conjunto< T, CMP >::find (const value_type & *s*)

busca una entrada en el conjunto

Parámetros

in	*s*	entrada a buscar.

Devuelve

Si existe una entrada en el conjunto con ese valor devuelve el iterador a su posicion, en caso contrario devuelve iterador al final de conjunto

Postcondición

no modifica el conjunto.

5.1.4.16. template<typename T, class CMP> const_iterator conjunto< T, CMP >::find (const value_type & *s*) const

5.1.4.17. template<typename T, class CMP> impar_iterator conjunto< T, CMP >::ibegin ( )

begin del conjunto

Devuelve

Devuelve un iterador impar (o iterador impar constante, respectivamente) al primer elemento (de posición impar) del conjunto. Si no existe devuelve end

5.1.4.18. template<typename T, class CMP> impar_iterator conjunto< T, CMP >::iend ( )

end del conjunto

Devuelve

Devuelve un iterador impar (o iterador impar constante, respectivamente) al final del conjunto (posicion siguiente al ultimo.

5.1.4.19. template<typename T, class CMP> pair<iterator,bool> conjunto< T, CMP >::insert (const value_type & *val*)

Inserta una entrada en el conjunto.

Parámetros

val	entrada a insertar

Devuelve

un par donde el segundo campo vale true si la entrada se ha podido insertar con éxito, esto es, no existe una mutación con igual valor en el conjunto. False en caso contrario. El primer campo del par devuelve un iterador al elemento insertado, o end() si no fue posible la insercion

Postcondición

Si e no esta en el conjunto, el size() sera incrementado en 1.

5.1.4.20. template<typename T, class CMP> iterator conjunto< T, CMP >::lower_bound (const value_type & *val*)

busca primer elemento por debajo ('antes', '<') de los parámetros dados.

Parámetros

in	*val*	entrada.

Devuelve

Devuelve un iterador al primer elemento que cumple que "elemento<e" es falso, esto es, el primer elemento que es mayor o igual que val Si no existe devuelve end

Postcondición

no modifica el conjunto.

5.1.4.21. template<typename T, class CMP> const_iterator conjunto< T, CMP >::lower_bound (const value_type & *val*) const

5.1.4.22. template<typename T, class CMP> conjunto& conjunto< T, CMP >::operator= (const conjunto< T, CMP > & *org*)

operador de asignación

Parámetros

in	*org*	conjunto a copiar.

Devuelve

Crea y devuelve un conjunto duplicado exacto de org.

5.1.4.23. template<typename T, class CMP> secure_iterator conjunto< T, CMP >::sbegin ( )

begin del conjunto

Devuelve

Devuelve un iterador seguro (o iterador seguro constante, respectivamente) al primer elemento del conjunto. Si no existe devuelve end

5.1.4.24. template<typename T, class CMP> secure_iterator conjunto< T, CMP >::send ( )

end del conjunto

Devuelve

Devuelve un iterador seguro (o iterador seguro constante, respectivamente) al final del conjunto (posicion siguiente al ultimo.

Postcondición

no modifica el conjunto.

5.1.4.25. template<typename T, class CMP> size_type conjunto< T, CMP >::size ( ) const

numero de entradas en el conjunto

Postcondición

No se modifica el conjunto.

Devuelve

numero de entradas en el conjunto

5.1.4.26. template<typename T, class CMP> iterator conjunto< T, CMP >::upper_bound (const value_type & *val*)

busca primer elemento por encima ('después', '>') de los parámetros dados.

Parámetros

in	*val*	entrada. Devuelve un iterador al primer elemento que cumple que "elemento>e", esto es, el primer elemento ESTRICTAMENTE mayor que val Si no existe devuelve end

Postcondición

no modifica el conjunto.

5.1.4.27. template<typename T, class CMP> const_iterator conjunto< T, CMP >::upper_bound (const value_type & *val*) const

5.1.5. Documentación de las funciones relacionadas y clases amigas

5.1.5.1. template<typename T, class CMP> friend class impar_iterator [friend]

5.1.5.2. template<typename T, class CMP> friend class iterator [friend]

5.1.5.3. template<typename T, class CMP> friend class secure_iterator [friend]

5.1.6. Documentación de los datos miembro

5.1.6.1. template<typename T, class CMP> CMP conjunto< T, CMP >::comp [private]

5.1.6.2. template<typename T, class CMP> vector<value_type> conjunto< T, CMP >::vm [private]

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

- conjunto.h

5.2. Referencia de la Clase conjunto< T, CMP >::impar_iterator

```
#include <conjunto.h>
```

Métodos públicos

- impar_iterator ()
- impar_iterator (const impar_iterator &x)
- const T & operator∗ ()
- impar_iterator & operator++ ()
- impar_iterator operator++ (int i)
- bool operator== (const impar_iterator &x) const
- bool operator!= (const impar_iterator &x) const
- impar_iterator & operator= (const impar_iterator &x)

Atributos públicos

- vector< T >::iterator it
- vector< T > ∗ elvector

Amigas

- class conjunto< T, CMP >

5.2.1. Documentación del constructor y destructor

5.2.1.1. template<typename T, class CMP> conjunto< T, CMP >::impar_iterator::impar_iterator ( )

5.2.1.2. template<typename T, class CMP> conjunto< T, CMP >::impar_iterator::impar_iterator (const impar_iterator & *x*)

5.2.2. Documentación de las funciones miembro

5.2.2.1. template<typename T, class CMP> bool conjunto< T, CMP >::impar_iterator::operator!= (const impar_iterator & *x*) const

5.2.2.2. template<typename T, class CMP> const T& conjunto< T, CMP >::impar_iterator::operator* ( )

5.2.2.3. template<typename T, class CMP> impar_iterator& conjunto< T, CMP >::impar_iterator::operator++ ( )

5.2.2.4. template<typename T, class CMP> impar_iterator conjunto< T, CMP >::impar_iterator::operator++ (int *i*)

5.2.2.5. template<typename T, class CMP> impar_iterator& conjunto< T, CMP >::impar_iterator::operator= (const impar_iterator & *x*)

5.2.2.6. template<typename T, class CMP> bool conjunto< T, CMP >::impar_iterator::operator== (const impar_iterator & *x*) const

5.2.3. Documentación de las funciones relacionadas y clases amigas

5.2.3.1. template<typename T, class CMP> friend class conjunto< T, CMP > [friend]

5.2.4. Documentación de los datos miembro

5.2.4.1. template<typename T, class CMP> vector<T>* conjunto< T, CMP >::impar_iterator::elvector

5.2.4.2. template<typename T, class CMP> vector<T>::iterator conjunto< T, CMP >::impar_iterator::it

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

- conjunto.h

5.3. Referencia de la Clase conjunto< T, CMP >::iterator

```
#include <conjunto.h>
```

Métodos públicos

- iterator ()
- iterator (const iterator &x)
- iterator (const secure_iterator &x)
- const T & operator∗ ()
- iterator & operator++ ()
- iterator operator++ (int i)
- iterator & operator-- ()
- iterator operator-- (int i)
- bool operator== (const iterator &x) const
- bool operator!= (const iterator &x) const
- iterator & operator= (const iterator &x)

Atributos públicos

- vector< T >::iterator it
- vector< T > ∗ elvector

Amigas

- class conjunto< T, CMP >

5.3.1. Documentación del constructor y destructor

5.3.1.1. template<typename T, class CMP> conjunto< T, CMP >::iterator::iterator ( )

5.3.1.2. template<typename T, class CMP> conjunto< T, CMP >::iterator::iterator (const iterator & *x*)

5.3.1.3. template<typename T, class CMP> conjunto< T, CMP >::iterator::iterator (const secure_iterator & *x*)

5.3.2. Documentación de las funciones miembro

5.3.2.1. template<typename T, class CMP> bool conjunto< T, CMP >::iterator::operator!= (const iterator & *x*) const

5.3.2.2. template<typename T, class CMP> const T& conjunto< T, CMP >::iterator::operator∗ ( )

5.3.2.3. template<typename T, class CMP> iterator& conjunto< T, CMP >::iterator::operator++ ( )

5.3.2.4. template<typename T, class CMP> iterator conjunto< T, CMP >::iterator::operator++ (int *i*)

5.3.2.5. template<typename T, class CMP> iterator& conjunto< T, CMP >::iterator::operator-- ( )

5.3.2.6. template<typename T, class CMP> iterator conjunto< T, CMP >::iterator::operator-- (int *i*)

5.3.2.7. template<typename T, class CMP> iterator& conjunto< T, CMP >::iterator::operator= (const iterator & *x*)

5.3.2.8. template<typename T, class CMP> bool conjunto< T, CMP >::iterator::operator== (const iterator & *x*) const

5.3.3. Documentación de las funciones relacionadas y clases amigas

5.3.3.1. template<typename T, class CMP> friend class conjunto< T, CMP > [friend]

5.3.4. Documentación de los datos miembro

5.3.4.1. template<typename T, class CMP> vector<T>∗ conjunto< T, CMP >::iterator::elvector

5.3.4.2. template<typename T, class CMP> vector<T>::iterator conjunto< T, CMP >::iterator::it

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

- conjunto.h

5.4. Referencia de la Clase conjunto< T, CMP >::secure_iterator

`#include <conjunto.h>`

Métodos públicos

- secure_iterator ()
- secure_iterator (const secure_iterator &x)
- const T & operator* ()
- secure_iterator & operator++ ()
- secure_iterator operator++ (int i)
- secure_iterator & operator-- ()
- secure_iterator operator-- (int i)
- bool operator== (const secure_iterator &x) const
- bool operator!= (const secure_iterator &x) const
- secure_iterator & operator= (const secure_iterator &x)

Atributos públicos

- vector< T >::iterator it
- vector< T > * elvector

Amigas

- class conjunto< T, CMP >

5.4.1. Documentación del constructor y destructor

5.4.1.1. template<typename T, class CMP> conjunto< T, CMP >::secure_iterator::secure_iterator ( )

5.4.1.2. template<typename T, class CMP> conjunto< T, CMP >::secure_iterator::secure_iterator (const secure_iterator & *x*)

5.4.2. Documentación de las funciones miembro

5.4.2.1. template<typename T, class CMP> bool conjunto< T, CMP >::secure_iterator::operator!= (const secure_iterator & *x*) const

5.4.2.2. template<typename T, class CMP> const T& conjunto< T, CMP >::secure_iterator::operator* ( )

5.4.2.3. template<typename T, class CMP> secure_iterator& conjunto< T, CMP >::secure_iterator::operator++ ( )

5.4.2.4. template<typename T, class CMP> secure_iterator conjunto< T, CMP >::secure_iterator::operator++ (int *i*)

5.4.2.5. template<typename T, class CMP> secure_iterator& conjunto< T, CMP >::secure_iterator::operator-- ( )

5.4.2.6. template<typename T, class CMP> secure_iterator conjunto< T, CMP >::secure_iterator::operator-- (int *i*)

5.4.2.7. template<typename T, class CMP> secure_iterator& conjunto< T, CMP >::secure_iterator::operator= (const secure_iterator & *x*)

5.4.2.8. template<typename T, class CMP> bool conjunto< T, CMP >::secure_iterator::operator== (const secure_iterator & *x*) const

5.4.3. Documentación de las funciones relacionadas y clases amigas

5.4.3.1. template<typename T, class CMP> friend class conjunto< T, CMP > [friend]

5.4.4. Documentación de los datos miembro

5.4.4.1. template<typename T, class CMP> vector<T>* conjunto< T, CMP >::secure_iterator::elvector

5.4.4.2. template<typename T, class CMP> vector<T>::iterator conjunto< T, CMP >::secure_iterator::it

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

- conjunto.h

6. Documentación de archivos

6.1. Referencia del Archivo conjunto.h

```
#include <string>
#include <vector>
#include <iostream>
#include <assert.h>
#include "conjunto.hxx"
```

Clases

- class conjunto< T, CMP >

  Clase conjunto.
- class conjunto< T, CMP >::iterator
- class conjunto< T, CMP >::secure_iterator
- class conjunto< T, CMP >::impar_iterator

Funciones

- template<typename T , typename CMP >
  ostream & operator<< (ostream &sal, const conjunto< T, CMP > &C)

  imprime todas las entradas del conjunto

6.1.1. Documentación de las funciones

6.1.1.1. template<typename T , typename CMP > ostream& operator<< (ostream & *sal,* const conjunto< T, CMP > & *C*)

imprime todas las entradas del conjunto

Postcondición

No se modifica el conjunto. Implementar tambien esta funcion

6.2. Referencia del Archivo documentacion.dox

6.3. Referencia del Archivo principal.cpp

```
#include "mutacion.h"
#include <iostream>
#include <fstream>
#include "conjunto.h"
```

Funciones

- template<class CMP >
  bool load (conjunto< mutacion, CMP > &cm, const string &s)
  lee un fichero de mutaciones, linea a linea
- int main (int argc, char *argv[ ])

6.3.1. Documentación de las funciones

6.3.1.1. template<class CMP > bool load (conjunto< mutacion, CMP > & *cm,* const string & *s*)

lee un fichero de mutaciones, linea a linea

Parámetros

in	*s*	nombre del fichero
in,out	*cm*	objeto tipo conjunto sobre el que se almacenan las mutaciones

Devuelve

true si la lectura ha sido correcta, false en caso contrario

6.3.1.2. int main (int *argc,* char * *argv[ ]*)

Índice alfabético