# My Project

Generated by Doxygen 1.8.9.1

Fri Nov 20 2015 11:40:18

# Contents

# Chapter 1

# PRACTICA TEMPLATE

**Version**

v0

**Author**

Juan F. Huete

## 1.1 Introducción

En la práctica anterior se os pidió la implementación del tipo conjunto de crímenes junto con sus iteradores asociados. En esta práctica, el objetivo es seguir avanzando en el uso de las estructuras de datos, particularmente mediante el uso de plantillas (templates) para la definición de tipos de datos genéricos.

Nuestro objetivo es dotar al TDA conjunto de la capacidad de controlar el criterio que se sigue para ubicar los elementos en el mismo. Para ello, es necesario que sobre el tipo de dato que se instancia el conjunto, en nuestro caso crimen se tenga definido una relacion de preorden total, R, esto es:

- Para todo x,y: xRy || yRx
- Para todo x,y,z: Si xRy && yRz entonces xRz

Por tanto R es una relación binaria que toma como entrada dos elementos del mismo tipo y como salida nos devuelve un booleano. Ejemplos de este tipo de relaciones son el operator$<$ (o el operator$>$) que se pueden definir sobre la clase crimen

```
class crimen {
 public:
  crimen();
  ....
  bool operator<(const crimen & y);
 private:
  ....
};

bool crimen::operator<(const crimen & y){
  return (this->ID < y.ID);
}
```

El criterio de ordenación será proporcionado a la hora de definir un conjunto, que será gestionado mediante por un objeto de comparación interno (functor de tipo CMP).

```
template <typename CMP> class conjunto;
```

La expresion comp(a,b), donde comp es un objeto de la clase CMP devuelve true si se considera que a precede b en la relación de preorden. Esta relación será utilizada por el set tanto para decidir cuando un elemento precede a otro en el contenedor como para determinar cuando dos elementos son equivalentes: para determinar cuando dos elementos serán considerados iguales con respecto a la relacion tendremos en cuenta que

- Si (!comp(a,b) && !comp(b,a)) entonces necesariamente a==b.

## 1.2 Uso de templates

Hasta ahora, los crímenes se encuentran almacenados en orden no decreciente de su valor de ID. Este conjunto puede ser de utilidad en muchos casos, sin embargo nos podríamos plantear el ordenar los elementos dentro del conjunto utilizando cualquier otro criterio. Así, podriamos tener

```
#include "conjunto.h"
...
// declaracion de tipos básicos:
conjunto<less<crimen > > X;  // elementos ordenados en orden creciende  (operator< sobre crimen)
conjunto<greater<crimen> > Y; // elementos ordenados en orden decreciente
       (operator> sobre crimen)

// declaracion de tipos más complejos:

conjunto<less<crimen> >::iterator itl;
conjunto<greater<crimen> >::iterator itg;

conjunto<greater<crimen> > Desc(X.begin(),X.end()); //los mismos elementos
       ordenados decrecientemente.

...

if (X.find("1234") == X.end())
   ....
```

Hay que notar que en este ejemplo X e Y representan a tipos distintos, esto es un conjunto ordenado en orden creciente de ID NO SERÁ del mismo tipo que un conjunto ordenado en orden decreciente de ID. De igual forma, itl e itg tampoco serán variables del mismo tipo, por lo que no podríamos hacer entre otras cosas asignaciones como X=Y o itg=itl.

En este caso, para realizar la práctica, el alumno deberá modificar tanto el fichero de especificación, conjunto.h, de forma que la propia especificación indique que trabaja con parámetros plantilla, como los ficheros de implementación (.hxx) de la clase conjunto. Además deberá de modificar los ficheros crimen.h y crimen.hxx para permitir la definición dels distintas relaciones de orden.

De igual forma se debe modificar el fichero principal.cpp de manera que se demuestre el correcto comportamiento del diccionario cuando se instancia bajo distintos criterios de ordenación, en concreto debemos asegurarnos que utilizamos los siguientes criterios de ordenación:

- creciente por id
- decreciente por id
- creciente por fecha
- decreciente por fecha
- creciente por IUCR

Para los dos primeros casos, y teniendo en cuenta que tenemos sobrecargado los operadores relaciones para crimen, es suficiente con utilizar las clases genéricas less<T> y greater<T> definidas en functional ( #include <functional> ). Sin embargo, para el resto de casos debemos implementar los functores que nos permitan realizar la correcta comparación entre crimenes.

## 1.3 functor

Para realizar dichas comparaciones utilizaremos una herramienta potente de C++: un functor (objeto función). Un functor es una clase en C++ que actua como una función. Un functor puede ser llamado puede ser llamado con una sintaxis familiar a la de las funciones en C++, pudiendo devolver valores y aceptar parámetros como una función normal.

Por ejemplo, si queremos crear un functor que compare dos crímenes teniendo en cuenta el orden IUCR, podríamos hacer

```
crimen x,y;
...
ComparacionPorFecha miFunctor;
cout << miFunctor(x,y) << endl;
```

Aunque miFunctor es un objeto, en la llamada miFunctor(x,y) la tratamos como si estuviésemos invocando a una función tomando x e y como parámetros.

Para crear dicho functor, creamos un objeto que sobrecarga el operador() como sigue

```
class ComparacionPorFecha {
 public:
   bool operator()(const crimen &a, const crimen &b) {
     return (a.getDate() < b.getDate()); // devuelve verdadero si el crimen a precede a b en
       el tiempo
 }
};
```

## 1.4 Generalizando el conjunto.

Para poder extender nuestro conjunto hemos de dotarlo de la capacidad de poder definir el criterio de ordenación. Para ello vamos a considerar un caso simplificado (que no se corresponde exactamente con lo que se pide en la práctica) donde ilustraremos su uso

```
template <typename CMP>
class conjunto {
  public:
    ....
    void insert( const crimen & c);

  private:
    vector<crimen> vc; //donde se almacenan los datos
    CMP comp;
};
```

Como hemos dicho, el nombre del tipo ahora es conjunto<CMP> y no conjunto. Distintas particularizaciones dan lugar a tipos también distintos. Ahora, en el fichero conjunto.hxx debemos de implementar cada uno de los métodos, recordemos que cada uno de ellos pertenece a la clase conjunto<CMP> y por tanto se implementa considerando

```
valorReturn conjunto<CMP>::nombreMetodo( parametros ...)
```

Pasamos a ver la implementación de los métodos:

### 1.4.1 Insert

El método insert asume como prerequisito que el conjunto está ordenado según el criterio dado por CMP, y por tanto debe asegurar que tras insertar un nuevo crimen dicho conjunto siga ordenado. Por ejemplo, podríamos hacer (recordad que en prácticas se pide hacer la búsqueda binaria) algo del tipo

```
void conjunto<CMP>::insert( const crimen & s){
        bool insertado = false;
```

```
        for (int i =0; !insertado && i < v.size(); )
            if (comp(v[i],s) ) i++;
            else {
              v.insert(v.begin()+i,s);
              insertado = true;
            }
        if (!insertado) v.push_back(s);
    }
```

En este caso comp(v[i],s) hace referencia a una comparación genérica entre crímenes definida por la relación de orden con la que se haya particularizado el conjunto. Así si hemos definido

```
conjunto<ComparacionPorFecha> cf;
```

en este caso comp es un objeto de la clase ComparacionPorFecha, y mediante la llamada comp(v[i],s) lo que estamos haciendo es llamar a la "función" que me compara dos crimenes teniendo en cuenta su campo fecha.

Finalmente, debemos tener cuidado a la hora de realizar comparaciones y la semántica de las mismas, por ejemplo, si queremos implementar la búsqueda binaria en un vector<crimen> que está dentro de un conjunto<less<crimen> > podriamos hacer algo como

```
bool conjunto<CMP>::busquedaBinaria (const crimen &d ){
    int sup=vc.size()-1;
    int inf = 0;
    while (sup > inf) {
      medio = (inf+sup)/2;
      if (vc[medio] == d) return true;  // comparamos igualdad entre crimen
      else if (vc[medio] < d) inf = medio+1; // comparamos menor entre crimen
      else sup = medio-1;
    }
    return false;
}
```

En este caso, estaríamos haciendo la llamada a la comparación de igualdad y menor entre crímenes (definida medianta la comparación de su ID) por lo que podría funcionar correctamente el método. Sin embargo, si el conjunto está definido como conjunto<ComparacionPorFecha>, utilizar el mismo código para realizar la búsqueda binaria no funcionaria correctamente: los elementos están ordenados en orden creciente de fecha. De hecho, no tendría sentido utilizar la búsqueda binaria para buscar un ID pues los elementos no se encuentran ordenados según ID en este conjunto<CompararPorFecha>.

El siguiente código nos permitiría utilizar la búsqueda binaria utilizando el criterio utilizado para ordenar los elementos en el conjunto.

```
bool conjunto<CMP>::busquedaBinaria (const crimen &d ){
    int sup=vc.size()-1;
    int inf = 0;
    while (sup > inf) {
      medio = (inf+sup)/2;
      if (!comp(vc[medio],d) && !comp(d,vc[medio]) ) return true;  // comparamos igualdad entre crimen
      else if (comp(vc[medio],d)) inf = medio+1; // comparamos menor entre crimen
      else sup = medio-1;
    }
    return false;
}
```

### 1.4.2 SE PIDE

Con la idea de reducir la parte de codificación, sólo será necesario entregar la implementacion del conjunto y dos de sus iteradores (la entrega del resto de ellos es opcional).

- conjunto<COMP>::iterator
- conjunto<COMP>::const_iterator

Además, al TDA conjunto le incluiremos los siguientes métodos:

- conjunto<CMP>::conjunto(iterator ini, iterator fin); Constructor de conjunto que contiene los elementos contenidos en el rango [ini,fin)

- iterator conjunto<CMP>::find(const crimen & c);
- const_iterator conjunto<CMP>::find(const crimen & c)const; Hace la búsqueda binaria del elemento en el conjunto considerando el orden definido por CMP. Devuelve el iterador que apunta a la posición donde se encuetra el elemento o end() en caso contrario.
- iterator conjunto<CMP>::find(const long int & id);
- const_iterator conjunto<CMP>::find(const long int & id)const; En este caso, como no sabemos cómo están ordenados los elementos será necesario realizar una una búsqueda lineal.
- iterator lower_bound (const entrada & x);
- const_iterator lower_bound (const entrada & x) const; Devuelven un iterador al primer elemento en el contenedor que no precede a x en el conjunto, esto es, es equivalente a x o le sigue según la relacion de orden definida por CMP. Esta función utiliza el functor interno devolviendo un iterador al primer elemento, e, para el que se satisface que comp(e,x) es falso.
- iterator upper_bound (const entrada & x);
- const_iterator lower_bound (const entrada & x) const; Devuelven un iterador al primer elemento que sigue a x según la relacion de orden definida por CMP. Esta función utiliza el functor interno devolviendo un iterador al primer elemento, e, para el que se satisface que comp(x,e) es cierto.

Dicha entrega se debe realizar antes del Viernes 27 de Noviembre, a las 23:59 horas.

# Chapter 2

# Todo List

**Class conjunto**

Implementa esta clase, junto con su documentación asociada

**Class crimen**

Implementa esta clase, junto con su documentación asociada

# Chapter 3

# Class Index

## 3.1 Class List

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

# Chapter 4

# Class Documentation

## 4.1 conjunto::arrest_iterator Class Reference

class arrest_iterator forward iterador sobre el conjunto, LECTURA arrest_iterator() ,operator∗(), operator++, operator++(int) operator=, operator==, operator!=

`#include <conjunto.h>`

### Public Member Functions

- **arrest_iterator** (const arrest_iterator &it)
- const conjunto::entrada & **operator∗** () const
- arrest_iterator **operator++** (int)
- arrest_iterator & **operator++** ()
- arrest_iterator **operator--** (int)
- arrest_iterator & **operator--** ()
- bool **operator==** (const arrest_iterator &it)
- bool **operator!=** (const arrest_iterator &it)

### Friends

- class **conjunto**

### 4.1.1 Detailed Description

class arrest_iterator forward iterador sobre el conjunto, LECTURA arrest_iterator() ,operator∗(), operator++, operator++(int) operator=, operator==, operator!=

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

- conjunto.h
- conjunto.hxx

## 4.2 conjunto Class Reference

Clase conjunto.

`#include <conjunto.h>`

## Classes

- class arrest_iterator

  *class arrest_iterator forward iterador sobre el conjunto, LECTURA arrest_iterator() ,operator∗(), operator++, operator++(int) operator=, operator==, operator!=*
- class const_arrest_iterator

  *class iterator forward iterator sobre el conjunto, LECTURA const_arrest_iterator() ,operator∗(), operator++, operator++(int) operator=, operator==, operator!=*
- class const_description_iterator
- class const_iterator

  *class const_iterator forward iterador constante sobre el conjunto, Lectura const_iterator ,operator∗, operator++, operator++(int) operator=, operator==, operator!=*
- class description_iterator

  *class description_iterator forward iterador constante sobre el conjunto, Lectura const_iterator ,operator∗, operator++, operator++(int) operator=, operator==, operator!= esta clase itera sobre todos los elementos que emparejan con una descripcion*
- class iterator

  *class iterator forward iterador sobre el conjunto, LECTURA iterator() ,operator∗(), operator++, operator++(int) operator=, operator==, operator!=*

## Public Types

- typedef crimen entrada

  *entrada permite hacer referencia al elemento almacenados en cada una de las posiciones del conjunto*
- typedef unsigned int size_type

  *size_type numero de elementos en el conjunto*

## Public Member Functions

- conjunto ()

  *constructor primitivo.*
- conjunto (const conjunto &d)

  *constructor de copia*
- conjunto::iterator find (const long int &id)

  *busca un crimen en el conjunto*
- conjunto::const_iterator find (const long int &id) const

  *busca un crimen en el conjunto*
- conjunto findIUCR (const string &iucr) const

  *busca los crimenes con el mismo codigo IUCR*
- conjunto findDESCR (const string &descr) const

  *busca los crimenes que contienen una determinada descripcion*
- bool insert (const conjunto::entrada &e)

  *Inserta una entrada en el conjunto.*
- bool erase (const long int &id)

  *Borra el delito dado un identificacador. Busca la entrada con id en el conjunto y si la encuentra la borra.*
- bool erase (const conjunto::entrada &e)

  *Borra una crimen con identificador dado por e.getID() en el conjunto. Busca la entrada con id en el conjunto (o e.getID() en el segundo caso) y si la encuentra la borra.*
- conjunto & operator= (const conjunto &org)

  *operador de asignación*
- size_type size () const

  *numero de entradas en el conjunto*

- bool empty () const

  *Chequea si el conjunto esta vacio.*
- long int ExisteElemento (const long int &ID) const

  *Busca en nuestro conjunto el elemento con el ID pasado por parámetro.*
- iterator begin ()

  *devuelve iterador al inicio del conjunto*
- iterator end ()

  *devuelve iterador al final (posición siguiente al último del conjunto*
- const_iterator cbegin () const

  *Iterador al principio del conjunto.*
- const_iterator cend () const

  *Iterador al final del conjunto.*
- arrest_iterator abegin ()

  *Iterador al primer arresto del conjunto.*
- arrest_iterator aend ()

  *Iterador al ultimo arresto del conjunto.*
- const_arrest_iterator cabegin ()

  *Iterador al primer arresto del conjunto.*
- const_arrest_iterator caend ()

  *Iterador al ultimo arresto del conjunto.*
- description_iterator dbegin (const string &descr)

  *devolver primera posicion del elemento que empareja con la descripcion descr*
- description_iterator dend ()

  *Devolver la ultima descripcion que empareje a "descr" del conjunto.*
- const_description_iterator cdbegin (const string &descr)

  *devolver primera posicion del elemento que empareja con la descripcion descr*
- const_description_iterator cdend ()

  *Devolver la ultima descripcion que empareje a "descr" del conjunto.*

## Friends

- class **description_iterator**
- class **const_description_iterator**
- class **iterator**
- class **const_iterator**
- class **arrest_iterator**
- class **const_arrest_iterator**
- ostream & operator<< (ostream &sal, const conjunto &D)

  *imprime todas las entradas del conjunto*

### 4.2.1 Detailed Description

Clase conjunto.

Métodos–> conjunto:: conjunto(), insert(), find(), findIUCR(), findDESCR(), erase(), size(), empty()

Tipos–> conjunto::entrada, conjunto::size_type

Descripción

Un conjunto es un contenedor que permite almacenar en orden creciente un conjunto de elementos no repetidos. En nuestro caso el conjunto va a tener un subconjunto restringido de métodos (inserción de elementos, consulta de un elemento, etc). Este conjunto "simulará" un conjunto de la stl, con algunas claras diferencias pues, entre otros, no estará dotado de la capacidad de iterar (recorrer) a través de sus elementos.

Asociado al conjunto, tendremos el tipo

```
conjunto::entrada
```

que permite hacer referencia al elemento almacenados en cada una de las posiciones del conjunto, en nuestro caso delitos (crimenes). Para esta entrada el requisito es que tenga definidos el operador< y operator=

Además encontraremos el tipo

```
conjunto::size_type
```

que permite hacer referencia al número de elementos en el conjunto.

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

Ejemplo de su uso:

```
...
conjunto DatosChicago, agresion;
crimen cr;

conjunto.insert(cr);
...
agresion = conjunto.findDESCR("BATTERY");

if (!agresion.empty()){
 cout <<"Tenemos "<< agresion.size() << " agresiones" << endl;
 cout << agresion << endl;
} else "No hay agresiones en el conjunto" << endl;
...
```

**Todo** Implementa esta clase, junto con su documentación asociada

### 4.2.2 Constructor & Destructor Documentation

#### 4.2.2.1 conjunto::conjunto ( )

constructor primitivo.

Implementacion de la clase conjunto

Implementacion de la clase conjunto

#### 4.2.2.2 conjunto::conjunto ( const conjunto & *d* )

constructor de copia

**Parameters**

| | | |
|---|---|---|
| in | *d* | conjunto a copiar |

### 4.2.3 Member Function Documentation

#### 4.2.3.1 conjunto::arrest_iterator conjunto::abegin ( )

Iterador al primer arresto del conjunto.

**Returns**

Un iterador al primer arresto del conjunto.

#### 4.2.3.2 conjunto::arrest_iterator conjunto::aend ( )

Iterador al ultimo arresto del conjunto.

**Returns**

Devuelve un iterador donde se produce el ultimo crimen con arresto del conjunto.

#### 4.2.3.3 conjunto::const_arrest_iterator conjunto::cabegin ( )

Iterador al primer arresto del conjunto.

**Returns**

Devuelve un const_arrest_iterator al primer arresto.

**Postcondition**

NO modifica el conjunto.

#### 4.2.3.4 conjunto::const_arrest_iterator conjunto::caend ( )

Iterador al ultimo arresto del conjunto.

**Returns**

Devuelve un const_arrest_iterator al ultimo crimen donde se produce un arresto.

**Postcondition**

NO modifica el conjunto.

#### 4.2.3.5 conjunto::const_iterator conjunto::cbegin ( ) const

Iterador al principio del conjunto.

**Returns**

Devuelve el const_iterator a la primera posición del conjunto.

**Postcondition**

no modifica el conjunto.

#### 4.2.3.6 conjunto::const_description_iterator conjunto::cdbegin ( const string & *descr* )

devolver primera posicion del elemento que empareja con la descripcion descr

**Parameters**

| | | |
|---|---|---|
| in | *descr* | descripcion de buscamos |

**Returns**

un iterador que apunta a la primera posicion, el emparejamiento se hace teniendo en cuenta que descr debe ser una subcadena de la descripción del delito.

**Postcondition**

NO modifica el conjunto.

#### 4.2.3.7 conjunto::const_description_iterator conjunto::cdend ( )

Devolver la ultima descripcion que empareje a "descr" del conjunto.

**Returns**

un iterador que apunta a la posicion final con la subcadena "descr".

**Postcondition**

NO modifica el conjunto.

#### 4.2.3.8 conjunto::const_iterator conjunto::cend ( ) const

Iterador al final del conjunto.

**Returns**

Devuelve el iterador constante a la posición final del conjunto.

**Postcondition**

no modifica el conjunto.

#### 4.2.3.9 conjunto::description_iterator conjunto::dbegin ( const string & *descr* )

devolver primera posicion del elemento que empareja con la descripcion descr

**Parameters**

| | | |
|---|---|---|
| in | *descr* | descripcion de buscamos |

**Returns**

un iterador que apunta a la primera posicion, el emparejamiento se hace teniendo en cuenta que descr debe ser una subcadena de la descripción del delito.

#### 4.2.3.10 conjunto::description_iterator conjunto::dend ( )

Devolver la ultima descripcion que empareje a "descr" del conjunto.

**Returns**

un iterador que apunta a la posicion final

#### 4.2.3.11 bool conjunto::empty ( ) const

Chequea si el conjunto esta vacio.

**Returns**

true si size()==0, false en caso contrario.

#### 4.2.3.12 bool conjunto::erase ( const long int & *id* )

Borra el delito dado un identificacador. Busca la entrada con id en el conjunto y si la encuentra la borra.

**Parameters**

| | | |
|---|---|---|
| in | *id* | a borrar |

**Returns**

true si la entrada se ha podido borrar con éxito. False en caso contrario

**Postcondition**

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

#### 4.2.3.13 bool conjunto::erase ( const conjunto::entrada & *e* )

Borra una crimen con identificador dado por e.getID() en el conjunto. Busca la entrada con id en el conjunto (o e.getID() en el segundo caso) y si la encuentra la borra.

**Parameters**

| | | |
|---|---|---|
| in | *entrada* | con e.getID() que geremos borrar, el resto de los valores no son tenidos en cuenta |

**Returns**

true si la entrada se ha podido borrar con éxito. False en caso contrario

**Postcondition**

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

#### 4.2.3.14 long int conjunto::ExisteElemento ( const long int & *ID* ) const

Busca en nuestro conjunto el elemento con el ID pasado por parámetro.

**Returns**

Posición del elemento buscado. -1 si no lo encuentra

#### 4.2.3.15 conjunto::iterator conjunto::find ( const long int & *id* )

busca un crimen en el conjunto

**Parameters**

| | |
|---|---|
| *id* | identificador del crimen buscar |

**Returns**

Si existe una entrada en el conjunto devuelve un iterador a lo posicion donde está el elemento. Si no se encuentra devuelve end()

**Postcondition**

no modifica el conjunto.

Ejemplo

```
if (C.find(12345)!=C.end() ) cout << "Esta" ;
else cout << "No esta";
```

#### 4.2.3.16 conjunto::const_iterator conjunto::find ( const long int & *id* ) const

busca un crimen en el conjunto

**Parameters**

| | |
|---|---|
| *id* | identificador del crimen buscar |

**Returns**

Si existe una entrada en el conjunto devuelve un iterador a lo posicion donde está el elemento. Si no se encuentra devuelve end()

**Postcondition**

no modifica el conjunto.

Ejemplo

```
if (C.find(12345)!=C.end() ) cout << "Esta" ;
else cout << "No esta";
```

#### 4.2.3.17 conjunto conjunto::findDESCR ( const string & *descr* ) const

busca los crimenes que contienen una determinada descripcion

**Parameters**

| | |
|---|---|
| *descr* | string que representa la descripcion del delito buscar |

**Returns**

Devuelve un conjunto con todos los crimenes que contengan descr en su descripcionR. Si no existe ninguno devuelve el conjunto vacio.

**Postcondition**

no modifica el conjunto.

Uso

```
vector<crimen> C, A;
....
A = C.findDESCR("BATTERY");
```

#### 4.2.3.18 conjunto conjunto::findIUCR ( const string & *iucr* ) const

busca los crimenes con el mismo codigo IUCR

**Parameters**

| | |
|---|---|
| *icur* | identificador del crimen buscar |

**Returns**

Devuelve un conjunto con todos los crimenes con el codigo IUCR. Si no existe ninguno devuelve el conjunto vacio.

**Postcondition**

no modifica el conjunto.

```
Uso
    vector<crimen> C, A;
    ....
    A = C.findIUCR("0460");
```

**Parameters**

| | |
|---|---|
| *iucr* | identificador del crimen buscar |

**Returns**

Devuelve un conjunto con todos los crimenes con el codigo IUCR. Si no existe ninguno devuelve el conjunto vacio.

**Postcondition**

no modifica el conjunto.

```
Uso
    vector<crimen> C, A;
    ....
    A = C.findIUCR("0460");
```

#### 4.2.3.19 bool conjunto::insert ( const conjunto::entrada & *e* )

Inserta una entrada en el conjunto.

**Parameters**

| | |
|---|---|
| *e* | entrada a insertar |

**Returns**

true si la entrada se ha podido insertar con éxito. False en caso contrario

**Postcondition**

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

#### 4.2.3.20 conjunto & conjunto::operator= ( const conjunto & *org* )

operador de asignación

**Parameters**

| | | |
|---|---|---|
| in | *org* | conjunto a copiar. Crea un conjunto duplicado exacto de org. |

#### 4.2.3.21 conjunto::size_type conjunto::size ( ) const

numero de entradas en el conjunto

**Postcondition**

No se modifica el conjunto.

### 4.2.4 Friends And Related Function Documentation

#### 4.2.4.1 ostream& operator<< ( ostream & *sal*, const conjunto & *D* ) [friend]

imprime todas las entradas del conjunto

**Postcondition**

No se modifica el conjunto.

**Todo** implementar esta funcion

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

- conjunto.h
- conjunto.hxx

## 4.3 conjunto::const_arrest_iterator Class Reference

class iterator forward iterator sobre el conjunto, LECTURA const_arrest_iterator() ,operator∗(), operator++, operator++(int) operator=, operator==, operator!=

`#include <conjunto.h>`

### Public Member Functions

- **const_arrest_iterator** (const const_arrest_iterator &it)
- const conjunto::entrada & **operator∗** () const
- const_arrest_iterator **operator++** (int)
- const_arrest_iterator & **operator++** ()
- const_arrest_iterator **operator--** (int)
- const_arrest_iterator & **operator--** ()
- bool **operator==** (const const_arrest_iterator &it)
- bool **operator!=** (const const_arrest_iterator &it)

### Friends

- class **conjunto**

### 4.3.1 Detailed Description

class iterator forward iterator sobre el conjunto, LECTURA const_arrest_iterator() ,operator∗(), operator++, operator++(int) operator=, operator==, operator!=

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

- conjunto.h
- conjunto.hxx

## 4.4 conjunto::const_description_iterator Class Reference

### Public Member Functions

- **const_description_iterator** (const const_description_iterator &it)
- const conjunto::entrada & **operator∗** () const
- const_description_iterator **operator++** (int)
- const_description_iterator & **operator++** ()
- const_description_iterator **operator--** (int)
- const_description_iterator & **operator--** ()
- bool **operator==** (const const_description_iterator &it)
- bool **operator!=** (const const_description_iterator &it)

### Friends

- class **conjunto**

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

- conjunto.h
- conjunto.hxx

## 4.5 conjunto::const_iterator Class Reference

class const_iterator forward iterador constante sobre el conjunto, Lectura const_iterator ,operator∗, operator++, operator++(int) operator=, operator==, operator!=

`#include <conjunto.h>`

### Public Member Functions

- **const_iterator** (const const_iterator &it)
- **const_iterator** (const iterator &it)
- const conjunto::entrada & **operator∗** () const
- const_iterator **operator++** (int)
- const_iterator & **operator++** ()
- const_iterator **operator--** (int)
- const_iterator & **operator--** ()
- bool **operator==** (const const_iterator &it)
- bool **operator!=** (const const_iterator &it)

### Friends

- class **conjunto**

### 4.5.1 Detailed Description

class const_iterator forward iterador constante sobre el conjunto, Lectura const_iterator ,operator∗, operator++, operator++(int) operator=, operator==, operator!=

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

- conjunto.h
- conjunto.hxx

## 4.6 crimen Class Reference

Clase crimen, asociada a la definición de un crimen.

`#include <crimen.h>`

### Public Member Functions

- crimen ()

  *Constructor primitivo de la clase.*
- crimen (const crimen &x)

  *Constructor de copia de la clase.*
- void setID (long int &id)

  *Establecer el ID de un crimen.*
- void setCaseNumber (const string &s)

  *Establecer el número del caso de un crimen.*
- void setDate (const fecha &d)

  *Establecer la fecha de un caso.*
- void setArrest (bool a)

  *Establecer si se produce un arresto o no.*
- void setDomestic (bool d)

  *Establecer si es un crimen doméstico.*
- long int getID () const

  *Obtener el ID de un crimen.*
- string getCaseNumber () const

  *Obtener el número del caso.*
- string getDescription () const

  *Obtener la descripción del crimen.*
- bool getArrest () const

  *Obtener si hay arresto.*
- string getIUCR () const

  *Obtener el IUCR del crimen.*
- void setIUCR (string new_IUCR)

  *Asignar un IUCR a un crimen.*
- void setDescription (string new_Descr)

  *Asignar una descripción a un crimen.*
- fecha getDate () const

*Devuelve la fecha del crimen.*

- crimen & operator= (const string &datos)

  *Copia en un crimen los datos de otro pasado como string.*

- crimen & operator= (const crimen &c)

  *Iguala dos crímenes.*

- bool operator== (const crimen &x) const

  *Compara si son iguales dos crímenes.*

- bool operator< (const crimen &x) const

  *Compara si un conjunto es mayor que otro.*

## Friends

- ostream & operator<< (ostream &, const crimen &)

  *Imprime todas las entradas del crimen.*

### 4.6.1 Detailed Description

Clase crimen, asociada a la definición de un crimen.

crimen::crimen, ..... Descripción contiene toda la información asociada a un crimen.

**Todo** Implementa esta clase, junto con su documentación asociada

### 4.6.2 Constructor & Destructor Documentation

#### 4.6.2.1 crimen::crimen ( )

Constructor primitivo de la clase.

Clase crimen, asociada a la definición de un crimen.

crimen::crimen, ..... Descripción contiene toda la información asociada a un crimen.

/**Constructor primitivo de la clase.

**Postcondition**

Se crea un nuevo objeto del tipo crimen.

#### 4.6.2.2 crimen::crimen ( const crimen & *x* )

Constructor de copia de la clase.

**Parameters**

| | | |
|---|---|---|
| in | *c* | crimen a copiar. |
| in | *x* | Crimen del que se copian los datos. |

**Postcondition**

Se crea un nuevo objeto del tipo crimen con los datos del objeto x.

### 4.6.3 Member Function Documentation

#### 4.6.3.1 bool crimen::getArrest ( ) const

Obtener si hay arresto.

**Returns**

bool true si hay arresto, false en caso contrario.

#### 4.6.3.2 string crimen::getCaseNumber ( ) const

Obtener el número del caso.

**Returns**

Devuelve el número del caso.
El número del caso.

#### 4.6.3.3 fecha crimen::getDate ( ) const

Devuelve la fecha del crimen.

**Returns**

Devuelve la fecha del crimen.
La fecha del crimen.

#### 4.6.3.4 string crimen::getDescription ( ) const

Obtener la descripción del crimen.

**Returns**

string con la descripción del caso.

#### 4.6.3.5 long crimen::getID ( ) const

Obtener el ID de un crimen.

**Returns**

Devuelve el ID.
El ID del crimen.

#### 4.6.3.6 string crimen::getIUCR ( ) const

Obtener el IUCR del crimen.

**Returns**

string con el IUCR del caso.

#### 4.6.3.7 bool crimen::operator< ( const crimen & *x* ) const

Compara si un conjunto es mayor que otro.

Compara si un conjunto es menor que otro.

**Parameters**

| | |
|---|---|
| *x* | Crimen a comparar. |

**Returns**

Devuelve true si x es mayor que el que lo llama. False en otro caso.

**Parameters**

| | | |
|---|---|---|
| in | *x* | Crimen a comparar. |

**Returns**

true si x es mayor que el que lo llama. False en otro caso.

#### 4.6.3.8 crimen & crimen::operator= ( const string & *datos* )

Copia en un crimen los datos de otro pasado como string.

**Parameters**

| | | |
|---|---|---|
| in | *string* | con los datos a copiar de otro crimen. |

#### 4.6.3.9 crimen & crimen::operator= ( const crimen & *c* )

Iguala dos crímenes.

**Parameters**

| | |
|---|---|
| *c* | Crimen a copiar en el que lo llama. |

**Returns**

Devuelve una copia del crimen.

**Parameters**

| | | |
|---|---|---|
| in | *c* | Crimen a copiar en el que lo llama. |

**Returns**

Una copia del crimen.

#### 4.6.3.10 bool crimen::operator== ( const crimen & *x* ) const

Compara si son iguales dos crímenes.

**Parameters**

| | |
|---|---|
| *x* | Crimen a comparar. |

**Returns**

Devuelve true si son iguales y false si no lo son.

**Parameters**

| | | |
|---|---|---|
| in | *x* | Crimen a comparar. |

**Returns**

true si son iguales y false si no lo son.

#### 4.6.3.11 void crimen::setArrest ( bool *a* )

Establecer si se produce un arresto o no.

**Parameters**

| | | |
|---|---|---|
| | *a* | Se produce arresto -> True / No se produce arresto -> False |
| in | *a* | Se produce arresto -> True / No se produce arresto -> False |

#### 4.6.3.12 void crimen::setCaseNumber ( const string & *s* )

Establecer el número del caso de un crimen.

**Parameters**

| | | |
|---|---|---|
| | *in* | s Número del caso. |
| in | *s* | Número del caso. |

#### 4.6.3.13 void crimen::setDate ( const fecha & *d* )

Establecer la fecha de un caso.

**Parameters**

| | | |
|---|---|---|
| | *d* | Fecha a establecer. |
| in | *d* | Fecha a establecer. |

#### 4.6.3.14 void crimen::setDescription ( string *new_Descr* )

Asignar una descripción a un crimen.

**Parameters**

| | | |
|---|---|---|
| in | *string* | con la descripción a asignar. |

#### 4.6.3.15 void crimen::setDomestic ( bool *d* )

Establecer si es un crimen doméstico.

**Parameters**

| | | |
|---|---|---|
| | *d* | Es crimen doméstico -> True / No es crimen doméstico -> False |
| in | *d* | Es crimen doméstico -> True / No es crimen doméstico -> False |

#### 4.6.3.16 void crimen::setID ( long int & *id* )

Establecer el ID de un crimen.

**Parameters**

| | | |
|---|---|---|
| in | *id* | ID a establecer. |

#### 4.6.3.17 void crimen::setIUCR ( string *new_IUCR* )

Asignar un IUCR a un crimen.

**Parameters**

| | | |
|---|---|---|
| in | *string* | con el IUCR a asignar. |

### 4.6.4 Friends And Related Function Documentation

#### 4.6.4.1 ostream& operator<< ( ostream & , const crimen & *x* ) [friend]

Imprime todas las entradas del crimen.

**Postcondition**

No se modifica el crimen original.

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

- crimen.h
- crimen.hxx

## 4.7 conjunto::description_iterator Class Reference

class description_iterator forward iterador constante sobre el conjunto, Lectura const_iterator ,operator*, operator++, operator++(int) operator=, operator==, operator!= esta clase itera sobre todos los elementos que emparejan con una descripcion

`#include <conjunto.h>`

### Public Member Functions

- **description_iterator** (const description_iterator &it)
- const conjunto::entrada & **operator*** () const
- description_iterator **operator++** (int)
- description_iterator & **operator++** ()
- description_iterator **operator--** (int)
- description_iterator & **operator--** ()
- bool **operator==** (const description_iterator &it)
- bool **operator!=** (const description_iterator &it)

### Friends

- class **conjunto**

### 4.7.1 Detailed Description

class description_iterator forward iterador constante sobre el conjunto, Lectura const_iterator ,operator*, operator++, operator++(int) operator=, operator==, operator!= esta clase itera sobre todos los elementos que emparejan con una descripcion

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

- conjunto.h
- conjunto.hxx

## 4.8 fecha Class Reference

### Public Member Functions

- fecha ()
  *Constructor primitivo de la clase.*
- fecha (const string &s)
  *Constructor de copia de la clase.*
- fecha & operator= (const fecha &f)
  *operador de asignación*
- fecha & operator= (const string &s)
  *operador de asignación*
- string toString () const
  *Muestra el valor de sus atributos.*
- string DarFormato (int param) const
  *Imprime datos en formato correcto (horas, minutos, etc)*
- bool operator== (const fecha &f) const
  *Compara si son iguales dos fechas.*
- bool operator< (const fecha &f) const
  *Compara si una fecha es mayor que otra.*
- bool operator> (const fecha &f) const
  *Compara si una fecha es menor que otra.*
- bool operator<= (const fecha &f) const
  *Compara si una fecha es mayor o igual que otra.*
- bool operator>= (const fecha &f) const
  *Compara si una fecha es menor o igual que otra.*
- bool operator!= (const fecha &f) const
  *Es un comparador de desigualdad.*

### Friends

- ostream & operator<< (ostream &os, const fecha &f)
  *Imprime todas las entradas de las fechas.*

### 4.8.1 Constructor & Destructor Documentation

#### 4.8.1.1 fecha::fecha ( )

Constructor primitivo de la clase.

fichero de implementacion de la clase fecha

Constructor sin parametros de la clase.

**Postcondition**

Se crea un nuevo objeto fecha con parametros por defecto.

#### 4.8.1.2 fecha::fecha ( const string & *x* )

Constructor de copia de la clase.

**Parameters**

| | | |
|---|---|---|
| in | *s* | string a copiar: |
| in | *x* | fecha del que se copian los datos. |

**Postcondition**

Se crea un nuevo objeto del tipo fehca con los datos del objeteo x.

### 4.8.2 Member Function Documentation

#### 4.8.2.1 string fecha::DarFormato ( int *param* ) const

Imprime datos en formato correcto (horas, minutos, etc)

**Parameters**

| | | |
|---|---|---|
| in | *param* | entero que se comprueba y/o formatea |

#### 4.8.2.2 bool fecha::operator!= ( const fecha & *f* ) const

Es un comparador de desigualdad.

Compara si una fecha.

**Parameters**

| | | |
|---|---|---|
| in | *f* | Fecha a comparar. |

**Returns**

Devuelve true cuando f es distinta del que la llama.False en otro caso.

**Parameters**

| | | |
|---|---|---|
| in | *f* | fecha a comparar. |

**Returns**

true si es mayor.

#### 4.8.2.3 bool fecha::operator< ( const fecha & *f* ) const

Compara si una fecha es mayor que otra.

Compara si una fecha es menor que otra.

**Parameters**

| | | |
|---|---|---|
| in | *f* | Fecha a comparar. |

**Returns**

Devuelve true si f es mayor que el que lo llama. False en otro caso.

**Parameters**

| | | |
|---|---|---|
| in | *f* | fecha a comparar. |

**Returns**

true si es menor.

#### 4.8.2.4 bool fecha::operator<= ( const fecha & *f* ) const

Compara si una fecha es mayor o igual que otra.

Compara si una fecha es menor o igual que otra.

**Parameters**

| | | |
|---|---|---|
| in | *f* | Fecha a comparar. |

**Returns**

Devuelve true si f es mayor o igual que el que lo llama.False en otro caso.

**Parameters**

| | | |
|---|---|---|
| in | *f* | fecha a comprar. |

**Returns**

true si es menor.

#### 4.8.2.5 fecha & fecha::operator= ( const fecha & *f* )

operador de asignación

Iguala dos fechas.

**Parameters**

| | | |
|---|---|---|
| in | *f* | fecha a copiar. |

**Postcondition**

Crea una fecha duplicada exacta de f

**Parameters**

| | | |
|---|---|---|
| in | *f* | fecha a copiar en el que lo llama. |

**Returns**

Una copia de la fecha.

#### 4.8.2.6 fecha & fecha::operator= ( const string & *s* )

operador de asignación

Iguala dos fechas.

**Parameters**

| | | |
|---|---|---|
| in | *s* | string a copiar. |
| in | *s* | fecha a copiar en el que lo llama. |

**Returns**

Una copia de la fecha.

#### 4.8.2.7 bool fecha::operator== ( const fecha & *f* ) const

Compara si son iguales dos fechas.

**Parameters**

| | | |
|---|---|---|
| in | *f* | Fecha a comparar. |

**Returns**

Devuelve true si son iguales y false si no lo son.

**Parameters**

| | | |
|---|---|---|
| in | *f* | fecha a comparar. |

**Returns**

true si son iguales y false si no lo son.

#### 4.8.2.8 bool fecha::operator> ( const fecha & *f* ) const

Compara si una fecha es menor que otra.

Compara si una fecha es mayor que otra.

**Parameters**

| | | |
|---|---|---|
| in | *f* | Fecha a comparar. |

**Returns**

Devuelve true si f es menor que el que lo llama. False en otro caso.

**Parameters**

| | | |
|---|---|---|
| in | *f* | fecha a comprar. |

**Returns**

true si es mayor.

#### 4.8.2.9 bool fecha::operator>= ( const fecha & *f* ) const

Compara si una fecha es menor o igual que otra.

Compara si una fecha es mayor que otra.

**Parameters**

| | | |
|---|---|---|
| in | *f* | Fecha a comparar. |

**Returns**

Devuelve true si f es menor o igual que el que lo llama.False en otro caso.

**Parameters**

| | | |
|---|---|---|
| in | *f* | fecha a comprar. |

**Returns**

true si es mayor.

#### 4.8.2.10 string fecha::toString ( ) const

Muestra el valor de sus atributos.

Copiar los datos de una fecha a un string.

**Returns**

Devuelve la fecha convertida a un string

**Postcondition**

Se crea un objeto string con los datos de fecha

### 4.8.3 Friends And Related Function Documentation

#### 4.8.3.1 ostream& operator<< ( ostream & *os,* const fecha & *f* ) [friend]

Imprime todas las entradas de las fechas.

**Postcondition**

No se modifica la fecha original

**Parameters**

| | | |
|---|---|---|
| in | *os* | flujo |
| in | *f* | fecha a mostrar. |

**Returns**

La fecha por salida estandar.

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

- fecha.h
- fecha.hxx

## 4.9 conjunto::iterator Class Reference

class iterator forward iterador sobre el conjunto, LECTURA iterator() ,operator∗(), operator++, operator++(int) operator=, operator==, operator!=

`#include <conjunto.h>`

### Public Member Functions

- **iterator** (const iterator &it)
- const conjunto::entrada & **operator∗** () const
- iterator **operator++** (int)
- iterator & **operator++** ()
- iterator **operator--** (int)
- iterator & **operator--** ()
- bool **operator==** (const iterator &it)
- bool **operator!=** (const iterator &it)

### Friends

- class **conjunto**

### 4.9.1 Detailed Description

class iterator forward iterador sobre el conjunto, LECTURA iterator() ,operator∗(), operator++, operator++(int) operator=, operator==, operator!=

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

- conjunto.h
- conjunto.hxx

# Index