My Project

Generated by Doxygen 1.8.6

Fri Nov 27 2015 12:16:54

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< CMP >

Implementa esta clase, junto con su documentación asociada

Member conjunto< CMP >::conjunto ()

implementar la funcion

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 ComparacionPorFechaCreciente Class Reference

Comparacion con fecha creciente.

`#include <crimen.h>`

Public Member Functions

- bool operator() (const crimen &a, const crimen &b) const

  sobrecarga del operador indexador

Friends

- class **crimen**

4.1.1 Detailed Description

Comparacion con fecha creciente.

4.1.2 Member Function Documentation

4.1.2.1 bool ComparacionPorFechaCreciente::operator() (const crimen & *a*, const crimen & *b*) const

sobrecarga del operador indexador

Parameters

Los	dos crimenes a comparar

Returns

true si cumple la condicion, falso en caso contrario

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

- crimen.h
- crimen.hxx

4.2 ComparacionPorFechaDecreciente Class Reference

Comparacion con fecha decreciente.

`#include <crimen.h>`

Public Member Functions

- bool operator() (const crimen &a, const crimen &b) const
  sobrecarga del operador indexador

Friends

- class **crimen**

4.2.1 Detailed Description

Comparacion con fecha decreciente.

4.2.2 Member Function Documentation

4.2.2.1 bool ComparacionPorFechaDecreciente::operator() (const crimen & *a,* const crimen & *b*) const

sobrecarga del operador indexador

Parameters

Los	dos crimenes a comparar

Returns

true si cumple la condicion, falso en caso contrario

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

- crimen.h
- crimen.hxx

4.3 ComparacionPorIUCRCreciente Class Reference

Comparacion con IUCR creciente.

`#include <crimen.h>`

Public Member Functions

- bool operator() (const crimen &a, const crimen &b) const
  sobrecarga del operador indexador

Friends

- class **crimen**

4.3.1 Detailed Description

Comparacion con IUCR creciente.

4.3.2 Member Function Documentation

4.3.2.1 bool ComparacionPorIUCRCreciente::operator() (const crimen & *a,* const crimen & *b*) const

sobrecarga del operador indexador

Parameters

Los	dos crimenes a comparar

Returns

true si cumple la condicion, falso en caso contrario

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

- crimen.h
- crimen.hxx

4.4 ComparacionPorIUCRDecreciente Class Reference

Comparacion con IUCR decreciente.

`#include <crimen.h>`

Public Member Functions

- bool operator() (const crimen &a, const crimen &b) const

  sobrecarga del operador indexador

Friends

- class **crimen**

4.4.1 Detailed Description

Comparacion con IUCR decreciente.

4.4.2 Member Function Documentation

4.4.2.1 bool ComparacionPorIUCRDecreciente::operator() (const crimen & *a,* const crimen & *b*) const

sobrecarga del operador indexador

Parameters

Los	dos crimenes a comparar

Returns

true si cumple la condicion, falso en caso contrario

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

- crimen.h
- crimen.hxx

4.5 conjunto< CMP > Class Template Reference

Clase conjunto.

`#include <conjunto.h>`

Classes

- class const_iterator

  class const_iterator forward iterador constante sobre el diccionario, Lectura const_iterator ,operator, operator++, operator++(int) operator=, operator==, operator!=*
- 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< CMP > &d)

  constructor de copia
- conjunto (iterator ini, iterator fin)

  Constructor de conjunto que contiene los elementos contenidos en el rango [ini,fin)
- conjunto::const_iterator find (const long int &id) const

  busca un crimen en el conjunto
- conjunto::iterator find (const entrada &c) 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< CMP >::entrada &e)

  Inserta una entrada en el conjunto.

- bool erase (const long int &id)

  Borra el delito dado un identificacador.
- bool erase (const conjunto< CMP >::entrada &e)

  Borra una crimen con identificador dado por e.getID() en el conjunto.
- iterator lower_bound (const entrada &x)

  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.
- const_iterator lower_bound (const entrada &x) const

  Devuelven un iterador constante 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)

  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. iterator upper_bound (const entrada & x);.
- const_iterator upper_bound (const entrada &x) const

  Devuelven un iterador constante 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. iterator upper_bound (const entrada & x);.
- conjunto< CMP > & operator= (const conjunto< CMP > &org)

  operador de asignación
- unsigned int size () const

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

  Chequea si el conjunto esta vacio.
- conjunto< CMP >::iterator begin ()

  devuelve iterador al inicio del conjunto
- conjunto< CMP >::iterator end ()

  devuelve iterador al final (posición siguiente al último del conjunto
- conjunto< CMP >::const_iterator cbegin ()
- conjunto< CMP >::const_iterator cend ()

  iterador al final

Friends

- class **iterator**
- class **const_iterator**
- ostream & operator<< (ostream &sal, const conjunto< CMP > &D)

  imprime todas las entradas del conjunto

4.5.1 Detailed Description

template<class CMP>class conjunto< CMP >

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

4.5.2 Constructor & Destructor Documentation

4.5.2.1 template<class CMP > conjunto< CMP >::conjunto ( )

constructor primitivo.

Implementacion de la clase conjunto

Todo implementar la funcion

4.5.2.2 template<class CMP > conjunto< CMP >::conjunto (const conjunto< CMP > & *d*)

constructor de copia

Parameters

in	*d*	conjunto a copiar

4.5.2.3 template<class CMP > conjunto< CMP >::conjunto (iterator *ini,* iterator *fin*)

Constructor de conjunto que contiene los elementos contenidos en el rango [ini,fin)

Parameters

ini	iterador inicial, fin iterador final

4.5.3 Member Function Documentation

4.5.3.1 template<class CMP > conjunto< CMP >::const_iterator conjunto< CMP >::cbegin ( )

Returns

Devuelve el const_iterator a la primera posición del conjunto.

Postcondition

no modifica el diccionario

4.5.3.2 template<class CMP > conjunto< CMP >::const_iterator conjunto< CMP >::cend ( )

iterador al final

Returns

Devuelve el iterador constante a la posición final del conjunto.

Postcondition

no modifica el diccionario

4.5.3.3 template<class CMP > bool conjunto< CMP >::empty ( ) const

Chequea si el conjunto esta vacio.

Returns

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

4.5.3.4 template<class CMP > bool conjunto< CMP >::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.

4.5.3.5 template<class CMP > bool conjunto< CMP >::erase (const conjunto< CMP >::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.

4.5.3.6 template<class CMP > conjunto< CMP >::const_iterator conjunto< CMP >::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";
```

4.5.3.7 template<class CMP > conjunto< CMP >::iterator conjunto< CMP >::find (const entrada & *c*) const

busca un crimen en el conjunto

Parameters

crimen	a 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";
```

4.5.3.8 template<class CMP > conjunto< CMP > conjunto< CMP >::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");
```

4.5.3.9 template<class CMP > conjunto< CMP > conjunto< CMP >::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");
```

4.5.3.10 template<class CMP > bool conjunto< CMP >::insert (const conjunto< CMP >::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.

4.5.3.11 template<class CMP> iterator conjunto< CMP >::lower_bound (const entrada & *x*)

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.

Parameters

crimen	a buscar

Returns

iterador al encontrado o al siguiente $*$ si no está.

4.5.3.12 template<class CMP> const_iterator conjunto< CMP >::lower_bound (const entrada & *x*) const

Devuelven un iterador constante 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.

Parameters

crimen	a buscar

Returns

iterador constante el encontrado o al siguiente $*$ si no está.

4.5.3.13 template<class CMP > conjunto< CMP > & conjunto< CMP >::operator= (const conjunto< CMP > & *org*)

operador de asignación

Parameters

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

4.5.3.14 template<class CMP > unsigned int conjunto< CMP >::size ( ) const

numero de entradas en el conjunto

Postcondition

No se modifica el conjunto.

4.5.3.15 template<class CMP> iterator conjunto< CMP >::upper_bound (const entrada & *x*)

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. iterator upper_bound (const entrada & x);.

Parameters

crimen	a buscar

Returns

iterador el encontrado o al siguiente $*$ si no está.

4.5.3.16 template<class CMP> const_iterator conjunto< CMP >::upper_bound (const entrada & *x*) const

Devuelven un iterador constante 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. iterator upper_bound (const entrada & x);.

Parameters

crimen	a buscar

Returns

iterador constante el encontrado o al siguiente * si no está.

4.5.4 Friends And Related Function Documentation

4.5.4.1 template<class CMP> ostream& operator<< (ostream & *sal*, const conjunto< CMP > & *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

4.6 conjunto< CMP >::const_iterator Class Reference

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

`#include <conjunto.h>`

Public Member Functions

- const_iterator ()
  constructor defecto const_iterator
- const_iterator (const const_iterator &it)
  constructor copia const_iterator
- const_iterator (const iterator &it)
  Convierte iterator en const_iterator.
- const conjunto< CMP >::entrada & operator* () const
  Sobrecarga del operador asterisco.
- const_iterator operator++ (int)
  Sobrecarga del operador ++. Operador de pre-incremento.
- const_iterator & operator++ ()
  Sobrecarga del operador ++ en modo post-incremento.
- const_iterator operator-- (int)
  Sobrecarga del operador – en modo pre-decremento.
- const_iterator & operator-- ()
  Sobrecarga del operador – en modo post-decremento.
- bool operator== (const const_iterator &it)
  Sobrecarga del operador ==. Operador de comparacion de igualdad.
- bool operator!= (const const_iterator &it)
  Sobrecarga del operador !=. Operador de comparacion de desigualdad.

Friends

- class conjunto< CMP >

  Declaramos esta clase como amiga de conjunto.

4.6.1 Detailed Description

template<class CMP>class conjunto< CMP >::const_iterator

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

4.6.2 Member Function Documentation

4.6.2.1 template<class CMP> bool conjunto< CMP >::const_iterator::operator!= (const const_iterator & *it*)

Sobrecarga del operador !=. Operador de comparacion de desigualdad.

Returns

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

4.6.2.2 template<class CMP> const conjunto< CMP >::entrada & conjunto< CMP >::const_iterator::operator* ( ) const

Sobrecarga del operador asterisco.

Returns

Devuelve un conjunto.

4.6.2.3 template<class CMP> conjunto< CMP >::const_iterator conjunto< CMP >::const_iterator::operator++ (int)

Sobrecarga del operador ++. Operador de pre-incremento.

Returns

Devuelve un iterador conjunto actual y luego incrementa a donde apunta el iterador.

4.6.2.4 template<class CMP> conjunto< CMP >::const_iterator & conjunto< CMP >::const_iterator::operator++ ( )

Sobrecarga del operador ++ en modo post-incremento.

Returns

Devuelve una referencia del iterador a la posición siguiente del conjunto de crimenes.

4.6.2.5 template<class CMP> conjunto< CMP >::const_iterator conjunto< CMP >::const_iterator::operator-- (int)

Sobrecarga del operador – en modo pre-decremento.

Returns

Devuelve un iterador al conjunto actual y luego decrementa a donde apunta el iterador.

4.6.2.6 template<class CMP> conjunto< CMP >::const_iterator & conjunto< CMP >::const_iterator::operator-- ( )

Sobrecarga del operador – en modo post-decremento.

Returns

Devuelve una referencia del iterador a la posición anterior del conjunto de crimenes.

4.6.2.7 template<class CMP> bool conjunto< CMP >::const_iterator::operator== (const const_iterator & *it*)

Sobrecarga del operador ==. Operador de comparacion de igualdad.

Returns

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

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

- conjunto.h
- conjunto.hxx

4.7 crimen Class Reference

Clase crimen.

`#include <crimen.h>`

Public Member Functions

- crimen ()
  Constructor por defecto.
- crimen (const crimen &x)
  Constructor de copia.
- void setID (long int &id)
  Set de la variable ID.
- void setCaseNumber (const string &s)
  Set de la variable caseNumber.
- void setDate (const fecha &d)
  Set de la variable date.
- void setIucr (const string &i)
  Set de la variable iucr.
- void setPrimaryType (const string &p)
  Set de la variable primaryType.
- void setDescription (const string &d)
  Set de la variable description.
- void setLocationDescription (const string &l)
  Set de la variable locationDescription.
- void setArrest (bool a)
  Set de la variable arrest.
- void setDomestic (bool d)
  Set de la variable domestica.
- void setLatitude (const double &l)

Set de la variable latitude.

- void setLongitude (const double &l)

  Set de la variable longitude.

- long int getID () const

  Get de la variable ID.

- string getCaseNumber () const

  Get de la variable caseNumber.

- fecha getDate () const

  Get de la variable date.

- string getIucr () const

  Get de la variable iucr.

- string getPrimaryType () const

  Get de la variable primaryType.

- string getDescription () const

  Get de la variable description.

- string getLocationDescription () const

  Get de la variable locationDescription.

- bool getArrest () const

  Get de la variable arrest.

- bool getDomestic () const

  Get de la variable domestic.

- double getLatitude () const

  Get de la variable latitude.

- double getLongitude () const

  Get de la variable longitude.

- crimen & operator= (const crimen &x)

  Asigna un conjunto a otro.

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

  Comprueba si un crimen es igual a otro mediante su ID.

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

  Comprueba si un crimen no es igual a otro mediante su ID.

- bool operator< (const crimen &x) const

  Comprueba si un crimen es menor estricto a otro mediante su ID.

- bool operator> (const crimen &x) const

  Comprueba si un crimen es mayor estricto a otro mediante su ID.

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

  Comprueba si un crimen es menor o igual a otro mediante su ID.

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

  Comprueba si un crimen es mayor o igual a otro mediante su ID.

Friends

- class **ComparacionPorID**
- class **ComparacionPorFecha**
- class **ComparacionPorIUCR**
- ostream & operator<< (ostream &os, const crimen &x)

  imprime todas las entradas del conjunto

4.7.1 Detailed Description

Clase crimen.

Métodos–> crimen:: crimen(), setID(), setCaseNumber(), setDate(), setIucr(), setPrimaryType(), setDescription, setLocationDescription, setArrest(), setDomestic(), setLatitude(), setLongitude(), getID(), getCaseNumber(), getDate(), getIucr(), getPrimaryType(), getDescription(), getLocationDescription(), getArrest(), getDomestic(), getLatitude(), getLongitude()

Descripción

Un crimen es un objeto que posee un compendio de datos relacionados con un crimen.

Estos datos son almacenados en variables, cada una con su set y get propio.

Su implementación precisa de un crimen ya creado, o de inicializar uno a uno sus valores.

Ejemplo de su uso:

```
...
crimen cr;
cr.setID(2345);
cr.setCaseNumber(AFS3423);
//Así sucesivamente
...
```

4.7.2 Constructor & Destructor Documentation

4.7.2.1 crimen::crimen (const crimen & *x*)

Constructor de copia.

Parameters

x	conjunto a copiar.

4.7.3 Member Function Documentation

4.7.3.1 bool crimen::getArrest ( ) const

Get de la variable arrest.

Returns

Devuelve el arrest (bool).

4.7.3.2 string crimen::getCaseNumber ( ) const

Get de la variable caseNumber.

Returns

Devuelve el caseNumber (string).

4.7.3.3 fecha crimen::getDate ( ) const

Get de la variable date.

Returns

Devuelve el date (fecha).

4.7.3.4 string crimen::getDescription ( ) const

Get de la variable description.

Returns

Devuelve el description (string).

4.7.3.5 bool crimen::getDomestic ( ) const

Get de la variable domestic.

Returns

Devuelve el domestic (bool).

4.7.3.6 long int crimen::getID ( ) const

Get de la variable ID.

Returns

Devuelve el ID (long int).

4.7.3.7 string crimen::getIucr ( ) const

Get de la variable iucr.

Returns

Devuelve el iucr (string).

4.7.3.8 double crimen::getLatitude ( ) const

Get de la variable latitude.

Returns

Devuelve el latitude (double).

4.7.3.9 string crimen::getLocationDescription ( ) const

Get de la variable locationDescription.

Returns

Devuelve el locationDescription (string).

4.7.3.10 double crimen::getLongitude ( ) const

Get de la variable longitude.

Returns

Devuelve el longitude (double).

4.7.3.11 string crimen::getPrimaryType ( ) const

Get de la variable primaryType.

Returns

Devuelve el primaryType (string).

4.7.3.12 bool crimen::operator!= (const crimen & x) const

Comprueba si un crimen no es igual a otro mediante su ID.

Parameters

x	Crimen con el que compara.

Returns

true si se cumple la condición, false en caso contrario.

Postcondition

No se modifica el conjunto.

4.7.3.13 bool crimen::operator< (const crimen & x) const

Comprueba si un crimen es menor estricto a otro mediante su ID.

Parameters

x	Crimen con el que compara.

Returns

true si se cumple la condición, false en caso contrario.

Postcondition

No se modifica el conjunto.

4.7.3.14 bool crimen::operator<= (const crimen & x) const

Comprueba si un crimen es menor o igual a otro mediante su ID.

Parameters

x	Crimen con el que compara.

Returns

true si se cumple la condición, false en caso contrario.

Postcondition

No se modifica el conjunto.

4.7.3.15 bool crimen::operator== (const crimen & x) const

Comprueba si un crimen es igual a otro mediante su ID.

Parameters

x	Crimen con el que compara.

Returns

true si son iguales, false en caso contrario.

4.7.3.16 bool crimen::operator> (const crimen & *x*) const

Comprueba si un crimen es mayor estricto a otro mediante su ID.

Parameters

x	Crimen con el que compara.

Returns

true si se cumple la condición, false en caso contrario.

Postcondition

No se modifica el conjunto.

4.7.3.17 bool crimen::operator>= (const crimen & *x*) const

Comprueba si un crimen es mayor o igual a otro mediante su ID.

Parameters

x	Crimen con el que compara.

Returns

true si se cumple la condición, false en caso contrario.

Postcondition

No se modifica el conjunto.

4.7.3.18 void crimen::setArrest (bool *a*)

Set de la variable arrest.

Parameters

a	Bool a asignar.

4.7.3.19 void crimen::setCaseNumber (const string & *s*)

Set de la variable caseNumber.

Parameters

s	String a asignar.

4.7.3.20 void crimen::setDate (const fecha & *d*)

Set de la variable date.

Parameters

d	Fecha a asignar.

4.7.3.21 void crimen::setDescription (const string & *d*)

Set de la variable description.

Parameters

d	String a asignar.

4.7.3.22 void crimen::setDomestic (bool *d*)

Set de la variable domestica.

Parameters

d	Bool a asignar.

4.7.3.23 void crimen::setID (long int & *id*)

Set de la variable ID.

Parameters

id	ID a asignar.

4.7.3.24 void crimen::setIucr (const string & *i*)

Set de la variable iucr.

Parameters

i	String a asignar.

4.7.3.25 void crimen::setLatitude (const double & *l*)

Set de la variable latitude.

Parameters

l	Double a asignar.

4.7.3.26 void crimen::setLocationDescription (const string & *l*)

Set de la variable locationDescription.

Parameters

l	String a asignar.

4.7.3.27 void crimen::setLongitude (const double & *l*)

Set de la variable longitude.

Parameters

l	Double a asignar.

4.7.3.28 void crimen::setPrimaryType (const string & *p*)

Set de la variable primaryType.

Parameters

p	String a asignar.

4.7.4 Friends And Related Function Documentation

4.7.4.1 ostream& operator<< (ostream & *os,* const crimen & *x*) [friend]

imprime todas las entradas del conjunto

Postcondition

No se modifica el conjunto.

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

- crimen.h
- crimen.hxx

4.8 fecha Class Reference

Clase fecha.

`#include <fecha.h>`

Public Member Functions

- fecha ()
  Constructor por defecto.
- fecha (const fecha &x)
  Constructor de copia.
- fecha (const string &s)
  Constructor a partir de un string.
- int getSec () const
  Get de la variable sec.
- int getMin () const
  Get de la variable min.

- int getHour () const

  Get de la variable hour.
- int getDay () const

  Get de la variable mday.
- int getMon () const

  Get de la variable mon.
- int getYear () const

  Get de la variable year.
- fecha & operator= (const fecha &f)

  Operador de asignacion.
- fecha & operator= (const string &s)

  Operador de asignacion.
- string toString () const

  Transforma un objeto fecha a string.
- bool operator== (const fecha &f) const

  Comprueba si una fecha es igual a otra.
- bool operator< (const fecha &f) const

  Comprueba si una fecha es menor que otra.
- bool operator> (const fecha &f) const

  Comprueba si una fecha es mayor que otra.
- bool operator<= (const fecha &f) const

  Comprueba si una fecha es menor o igual que otra.
- bool operator>= (const fecha &f) const

  Comprueba si una fecha es mayor o igual que otra.
- bool operator!= (const fecha &f) const

  Comprueba si una fecha es distinta a otra.

Friends

- ostream & **operator**<< (ostream &os, const fecha &f)

4.8.1 Detailed Description

Clase fecha.

Métodos–> fecha:: fecha(), getSec(), getMin(), getHour(), getDay(), getMon(), getYear()

Descripción

Esta clase contiene toda la información asociada a una fecha con el formato mm/dd/aaaa hh:mm:ss AM/PM

Ejemplo de su uso:

```
...
string cad1 = "01/01/2000 00:00:00 AM";
fecha date1(cad1);
string cad2 = "01/09/2015 01:00:00 AM";
fecha date2(cad2);
if(date1 <= date2)
{
    cout << "date1 menor o igual que date2" << endl;
}
else
{
    cout << "date2 menor que date1" << endl;
}
...
```

4.8.2 Constructor & Destructor Documentation

4.8.2.1 fecha::fecha ( )

Constructor por defecto.

fichero de implementacion de la clase fecha

4.8.2.2 fecha::fecha (const fecha & *x*)

Constructor de copia.

Parameters

x	Conjunto a copiar.

4.8.2.3 fecha::fecha (const string & *s*)

Constructor a partir de un string.

Parameters

s	Parametro con formato mm/dd/aaaa hh:mm:ss AM/PM

4.8.3 Member Function Documentation

4.8.3.1 int fecha::getDay ( ) const

Get de la variable mday.

Returns

Devuelve la variable mday

4.8.3.2 int fecha::getHour ( ) const

Get de la variable hour.

Returns

Devuelve la variable hour

4.8.3.3 int fecha::getMin ( ) const

Get de la variable min.

Returns

Devuelve la variable min

4.8.3.4 int fecha::getMon ( ) const

Get de la variable mon.

Returns

Devuelve la variable mon

4.8.3.5 int fecha::getSec ( ) const

Get de la variable sec.

Returns

Devuelve la variable sec

4.8.3.6 int fecha::getYear ( ) const

Get de la variable year.

Returns

Devuelve la variable year

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

Comprueba si una fecha es distinta a otra.

Parameters

f	Fecha con la que compara.

Returns

Devuelve true si es distinta, false en caso contrario.

4.8.3.8 bool fecha::operator< (const fecha & *f*) const

Comprueba si una fecha es menor que otra.

Parameters

f	Fecha con la que compara.

Returns

Devuelve true si es menor, false en caso contrario.

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

Comprueba si una fecha es menor o igual que otra.

Parameters

f	Fecha con la que compara.

Returns

Devuelve true si es menor o igual, false en caso contrario.

4.8.3.10 fecha & fecha::operator= (const fecha & *f*)

Operador de asignacion.

Parameters

f	Fecha a copiar. Crea una fecha duplicada exacta a f.

4.8.3.11 fecha & fecha::operator= (const string & *s*)

Operador de asignacion.

Parameters

s	String a copiar. Crea una fecha duplicada a partir de s.

4.8.3.12 bool fecha::operator== (const fecha & *f*) const

Comprueba si una fecha es igual a otra.

Parameters

f	Fecha con la que compara.

Returns

Devuelve true si es igual, false en caso contrario.

4.8.3.13 bool fecha::operator> (const fecha & *f*) const

Comprueba si una fecha es mayor que otra.

Parameters

f	Fecha con la que compara.

Returns

Devuelve true si es mayor, false en caso contrario.

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

Comprueba si una fecha es mayor o igual que otra.

Parameters

f	Fecha con la que compara.

Returns

Devuelve true si es mayor o igual, false en caso contrario.

4.8.3.15 string fecha::toString ( ) const

Transforma un objeto fecha a string.

Returns

Un string con formato mm/dd/aaaa hh:mm:ss AM/PM.

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

- fecha.h
- fecha.hxx

4.9 conjunto< CMP >::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 ()
  constructor defecto iterator
- iterator (const iterator &it)
  constructor copia iterator
- const conjunto< CMP >::entrada & operator* () const
  Sobrecarga del operador asterisco.
- iterator operator++ (int)
  Sobrecarga del operador ++. Operador de pre-incremento.
- iterator & operator++ ()
  Sobrecarga del operador ++. Operador de post-incremento.
- iterator operator-- (int)
  Sobrecarga del operador –. Operador de pre-decremento.
- iterator & operator-- ()
  Sobrecarga del operador –. Operador de post-decremento.
- bool operator== (const iterator &it)
  Sobrecarga del operador ==. Operador de comparacion de igualdad.
- bool operator!= (const iterator &it)
  Sobrecarga del operador !=. Operador de comparacion de desigualdad.

Friends

- class conjunto< CMP >
  Declaramos esta clase como amiga de conjunto.

4.9.1 Detailed Description

template<class CMP>class conjunto< CMP >::iterator

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

4.9.2 Member Function Documentation

4.9.2.1 template<class CMP> bool conjunto< CMP >::iterator::operator!= (const iterator & *it*)

Sobrecarga del operador !=. Operador de comparacion de desigualdad.

Returns

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

4.9.2.2 template<class CMP> const conjunto< CMP >::entrada & conjunto< CMP >::iterator::operator* ( ) const

Sobrecarga del operador asterisco.

Returns

Devuelve el un conjunto.

4.9.2.3 template<class CMP> conjunto< CMP >::iterator conjunto< CMP >::iterator::operator++ (int)

Sobrecarga del operador ++. Operador de pre-incremento.

Returns

Devuelve un iterador conjunto actual y luego incrementa a donde apunta el iterador.

4.9.2.4 template<class CMP> conjunto< CMP >::iterator & conjunto< CMP >::iterator::operator++ ( )

Sobrecarga del operador ++. Operador de post-incremento.

Returns

Devuelve un iterador conjunto siguiente.

4.9.2.5 template<class CMP> conjunto< CMP >::iterator conjunto< CMP >::iterator::operator-- (int)

Sobrecarga del operador –. Operador de pre-decremento.

Returns

Devuelve un iterador conjunto actual y luego decrementa a donde apunta el iterador.

4.9.2.6 template<class CMP> conjunto< CMP >::iterator & conjunto< CMP >::iterator::operator-- ( )

Sobrecarga del operador –. Operador de post-decremento.

Returns

Devuelve un iterador conjunto anterior.

4.9.2.7 template<class CMP> bool conjunto< CMP >::iterator::operator== (const iterator & *it*)

Sobrecarga del operador ==. Operador de comparacion de igualdad.

Returns

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

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

- conjunto.h
- conjunto.hxx