ITERADORES EN CONJUNTO

Practica3

Generado por Doxygen 1.8.9.1

Viernes, 23 de Octubre de 2015 11:09:11

Contents

Chapter 1

Iterando sobre el conjunto

Versión

v0

Autor

Juan F. Huete

1.1 Introducción

En la práctica anterior hemos construido el conjunto, pero no estaba doctado de un mecanismo para poder iterar sobre las entradas que tiene almacenadas. Este mecanismo es esencialmente un iterador, que no es otra cosa que un objeto que se mueve a través de un contenedor (en la práctica nuestro conjunto) de conjunto::entrada (entradas en el conjunto). La ventaja del uso del iterador es que nos proporciona un mecanismo estándar 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 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 doxFileP3
```

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

Se entregam 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 del iterator. Se os pide implementar los distintos métodos así como el código necesario para demostrar el correcto funcionamiento del mismo.
- conjunto.hxx En este fichero se incluyen las cabeceras que hacen referencia al los distintos iteratorres, debiendo de añadir también las que hacen referencia al const_iterator.

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
class conjunto {
public:
   class iterator;  //declaraciones previa
   class const_iterator;
   class arrest_iterator; // Iterador sobre los delitos que implicaron un arresto (Arrest==true)
   class const_arrest_iterator;
   class description_iterator;  // Iterador sobre los delitos que concuerdan con la descripcion
   class const_description_iterator;
   ....
   iterator begin();  // Podemos declarar el iterator porque ya lo hemos declarado previamente
   const_iterator cbegin();
   arrest_iterator abegin();
   const_arrest_iterator cabegin( );
   description_iterator dbegin(const string & descrp);
   const_description_iterator cdbegin(const string & descrip);


   ....
   class iterator {
      //definicion del iterator
      public:
        iterator();
        ....
      private:
         friend class conjunto; // declaramos conjunto como amigo de la clase
        ....

    }; // end de la clase iterator
   private:
        friend class 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::iterator Por tanto, para declarar un conjunto y un iterator sobre dicho conjunto debemos hacer

```
conjunto C;
conjunto::iterator it;

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

conjunto::arrest_iterator ait; //Itera sobre todos los crimenes que han tenido un arresto
for (ait = C.abegin(); ait!= C.aend();++ait)
  cout << *it << endl;


conjunto::description_iterator dit, dit2;  //Itera sobre los elementos
       relacionados con BATTERY, i.e., BATTERY esta dentro de la descripcion
for (dit = C.dbegin("BATTERY"); dit!= C.dend();++dit)
  cout << *it << endl;

dit2 = C.dbegin("FINANCIAL");  // Iteraria solo sobre los elementos relacionados con FINANCIAL
```

1.4 begin y end

Para poder iterar sobre los elementos del contenedor, debemos dotarlo de dos 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 crimen 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() // Correncto
*it.getID() // Incorrecto, primero evalua 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::const_iterator c_it = dic.cbegin();
```

1.4.1 debin y end en description_iterator

En esta práctica debemos destacar el comportamiento del description_iterator. Dicho iterador nos permitirá iterar sobre todos los elementos que contengan una determinada subcadena como parte de la descripción. Para ello, debemos indicarle al iterador sobre que subcadena debe iterar, esto lo haremos mediante el método dbegin() (o cdbegin()) que recibirán como parámetro de entrada dicha cadena.

```
@brief devolver primera posicion del elemento que empareja con la descripcion descr
@param[in] descr descripcion de buscamos
@return 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.

conjunto::description_iterator conjunto::dbegin(const string & descr) const;
```

El método end() nos debe devolver un iterador que apunta a la posición final del mismo, en este caso dicho iterador puede coincider con el end() del vector<crimenes>

1.5 Modificación en la especificación de algunos métodos.

Al permitir el uso de iteradores hay métodos de la clase conjunto que ya no tendrían sentido tal y como lo estaban previamente definidos.

- pair< conjunto::entrada, bool > find (const long int &id) const ;

En este caso, lo podemos modificar para que busca una entrada en el conjunto. Si la encuentra devuelve el iterador que apunta a la entrada, en caso contrario devuelve end().

```
  conjunto::iterator  conjunto::find( const Key & s) ;
....
Ejemplo de uso:
conjunto::iterator it;
it = C.find(3456);
if (it == C.end()) cout << "No esta " << endl;
else cout << "Delito "<< (*it) << endl;
```

- conjunto::const_iterator conjunto::find(const long int &id) const;

El comportamiento es similar al anterior pero en este caso devuelve un iterador constante.

1.6 Representacion del iterator

Un iterator de la clase conjunto nos debe permitir el acceso a los datos almacenados el en conjunto propiamente dicho. Una primera alternativa sería representar el iterator como un iterador sobre el vector, esto es

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

Sin embargo, con esta representación sería posible violar el invariante de la representación, pues el usuario de la clase podría modificar el contenido de la clave ejecutando

```
...
it = C.find(1234);
if (it == Dic.end()) cout << "No esta " << endl;
else {
 (*it).setID(22222);
}
```

Esto nos daría problemas pues estaríamos modificando la clase, y particularmente al asumir los datos ordenados, el conjunto podría dejar de estar ordenado, no cumpliría el invariante de la representación. A partir de este momento 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 almacenado en el mismo

```
class conjunto{

....
  class iterator {
    ....
    const entrada & operator*();
    ......
    private:
    ; // Puntero a la entrada del vector.
   };
....
  class const_iterator {
     .....
     const entrada & operator*();
     ....
     private:
     ....
};
```

1.7 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 se debe modificar el fichero prueba.cpp de manera que se demuestre el correcto comportamiento del conjunto cuando se instancia con distintos tipos.

1.7.1 A ENTREGAR

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

- fecha.h
- fecha.hxx
- crimen.h
- crimen.hxx
- conjunto.h Especificación del TDA conjunto.

- conjunto.hxx segunda versión del conjunto.
- prueba.cpp fichero de prueba del conjunto donde se incluyen los métodos que trabajan sobre meteorito

Dicha entrega tiene como límite el Viernes 13 de Noviembre.

Chapter 2

Lista de tareas pendientes

Clase conjunto

Implementa esta clase, junto con su documentación asociada

Miembro conjunto::conjunto ()

implementar la funcion

Chapter 3

Índice de clases

3.1 Lista de clases

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

conjunto
: Clase conjunto . . . . . **??**

conjunto::const_iterator
: Class const_iterator forward iterador constante sobre el diccionario, Lectura const_iterator ,operator∗, operator++, operator++(int) operator=, operator==, operator!= . . . . . **??**

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

conjunto::iterator
: Class iterator forward iterador sobre el conjunto, LECTURA iterator() ,operator∗(), operator++, operator++(int) operator=, operator==, operator!= . . . . . **??**

Chapter 4

Documentación de las clases

4.1 Referencia de la Clase conjunto

Clase conjunto.

`#include <conjunto.h>`

Clases

- class const_iterator

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

  class description_iterator forward iterador constante sobre el diccionario, 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!=

Tipos públicos

- 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

Métodos públicos

- iterator begin () const

  devuelve iterador al inicio del conjunto
- const_iterator cbegin () const
- const_iterator cend () const

  iterador al final
- conjunto ()

  constructor primitivo.
- conjunto (const conjunto &d)

  constructor de copia

- description_iterator dbegin (const string &descr) const

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

  devolver fin del conjunto
- bool empty () const

  Chequea si el conjunto esta vacio.
- iterator end () const

  devuelve iterador al final (posición siguiente al último del conjunto
- bool erase (const long int &id)

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

  Borra una crimen con identificador dado por e.getID() en el conjunto.
- conjunto::iterator find (const long int &id) const

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

  busca un crimen en el conjunto
- conjunto< conjunto::entrada > findDESCR (const string &descr) const

  busca los crimenes que contienen una determinada descripcion
- conjunto< conjunto::entrada > findIUCR (const string &iucr) const

  busca los crimenes con el mismo codigo IUCR
- bool insert (const conjunto::entrada &e)

  Inserta una entrada en el conjunto.
- conjunto & operator= (const conjunto &org)

  operador de asignación
- size_type size () const

  numero de entradas en el conjunto

Métodos privados

- bool cheq_rep () const

  Chequea el Invariante de la representacion.

Atributos privados

- vector< crimen > **vc**

Amigas

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

  imprime todas las entradas del conjunto

4.1.1 Descripción detallada

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;
...
```

Tareas pendientes Implementa esta clase, junto con su documentación asociada

4.1.2 Documentación del constructor y destructor

4.1.2.1 conjunto::conjunto ( )

constructor primitivo.

Implementacion de la clase conjunto

Tareas pendientes implementar la funcion

4.1.2.2 conjunto::conjunto (const conjunto & *d*)

constructor de copia

Parámetros

in	*d*	conjunto a copiar

4.1.3 Documentación de las funciones miembro

4.1.3.1 const_iterator conjunto::cbegin ( ) const

Devuelve

Devuelve el const_iterator a la primera posición del conjunto.

Postcondición

no modifica el diccionario

4.1.3.2 const_iterator conjunto::cend ( ) const

iterador al final

Devuelve

Devuelve el iterador constante a la posición final del conjunto.

Postcondición

no modifica el diccionario

4.1.3.3 bool conjunto::cheq_rep ( ) const [private]

Chequea el Invariante de la representacion.

Invariante

IR: rep ==> bool

- Para todo i, 0 <= i < vc.size() se cumple vc[i].ID > 0;
- Para todo i, 0 <= i <= D.dic.size()-1 se cumple vc[i].ID< vc[i+1].ID

Devuelve

true si el invariante es correcto, falso en caso contrario

4.1.3.4 description_iterator conjunto::dbegin (const string & *descr*) const

devolver primera posicion del elemento que empareja con la descripcion descr

Parámetros

in	*descr*	descripcion de buscamos

Devuelve

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.

4.1.3.5 description_iterator conjunto::dend ( ) const

devolver fin del conjunto

Devuelve

un iterador que apunta a la posicion final

4.1.3.6 bool conjunto::empty ( ) const

Chequea si el conjunto esta vacio.

Devuelve

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

4.1.3.7 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

Parámetros

in	*id*	a borrar

Devuelve

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

Postcondición

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

4.1.3.8 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

Parámetros

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

Devuelve

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

Postcondición

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

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

busca un crimen en el conjunto

Parámetros

id	identificador del crimen buscar

Devuelve

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

Postcondición

no modifica el conjunto.

Ejemplo

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

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

busca un crimen en el conjunto

Parámetros

id	identificador del crimen buscar

Devuelve

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

Postcondición

no modifica el conjunto.

Ejemplo

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

4.1.3.11 conjunto<conjunto::entrada> conjunto::findDESCR (const string & *descr*) const

busca los crimenes que contienen una determinada descripcion

Parámetros

descr	string que representa la descripcion del delito buscar

Devuelve

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

Postcondición

no modifica el conjunto.

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

4.1.3.12 conjunto<conjunto::entrada> conjunto::findIUCR (const string & *iucr*) const

busca los crimenes con el mismo codigo IUCR

Parámetros

icur	identificador del crimen buscar

Devuelve

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

Postcondición

no modifica el conjunto.

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

4.1.3.13 bool conjunto::insert (const conjunto::entrada & *e*)

Inserta una entrada en el conjunto.

Parámetros

e	entrada a insertar

Devuelve

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

Postcondición

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

4.1.3.14 conjunto& conjunto::operator= (const conjunto & *org*)

operador de asignación

Parámetros

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

4.1.3.15 size_type conjunto::size ( ) const

numero de entradas en el conjunto

Postcondición

No se modifica el conjunto.

4.1.4 Documentación de las funciones relacionadas y clases amigas

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

imprime todas las entradas del conjunto

Postcondición

No se modifica el conjunto.

Tareas pendientes implementar esta funcion

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

- conjunto.h
- conjunto.hxx

4.2 Referencia de la Clase conjunto::const_iterator

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

`#include <conjunto.h>`

Métodos públicos

- **const_iterator** (const const_iterator &it)
- const_iterator (const iterator &it)

  Convierte iterator en const_iterator.
- bool **operator!=** (const 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)

Atributos privados

- vector< entrada >::const_iterator **c_itv**

Amigas

- class **diccionario**

4.2.1 Descripción detallada

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

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

- conjunto.h

4.3 Referencia de la Clase conjunto::description_iterator

class description_iterator forward iterador constante sobre el diccionario, 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>`

Métodos públicos

- **description_iterator** (const description_iterator &it)
- bool **operator!=** (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)

Atributos privados

- vector< entrada >::const_iterator **c_itv**
- string **descr**

Amigas

- class **diccionario**

4.3.1 Descripción detallada

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

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

- conjunto.h

4.4 Referencia de la Clase conjunto::iterator

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

`#include <conjunto.h>`

Métodos privados

- iterator ()

  constructor defecto iterator
- iterator (const iterator &it) = i.itv

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

Atributos privados

- vector< entradas >::iterator **itv**

Amigas

- class **conjunto**

4.4.1 Descripción detallada

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

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

- conjunto.h
- conjunto.hxx