TDA Cronologia

3

Antonio Miguel Morillo Chica

Índice general

Capítulo 1

Representación del TDA Cronolofia.

1.1. Invariante de la representación

El invariante es *Cronologia.crono* no repite key. Así mismo la key perteneciende Cronologia.crono ha de conincidir con la key perteneciente a EventoHistorico.eventos.

1.2. Función de abstracción

Un objeto válido *rep* del TDA Cronologia representa un contenedor map de pair de un set de forma que el segundo elemento de Cronologia tiene la misma clave que el map: [[Key] [[SubKey] [[evento1] [evento2] ...]]].

Capítulo 2

Rep del TDA EventoHistorico.

2.1. Invariante de la representación.

El invariante es *eventos* no repite key ni elementos.

2.2. Función de abstracción.

Un objeto válido *rep* del TDA EventoHistorico representa el una asociación entre una fecha y un contenedor de efentos de la siguiente forma: [[Key] [[Evento1][Evento2]]].

Capítulo 3

Índice de clases

3.1. Lista de clases

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

Capítulo 4

Indice de archivos

4.1. Lista de archivos

Lista de todos los archivos documentados y con descripciones breves:

Capítulo 5

Documentación de las clases

5.1. Referencia de la Clase Cronologia::const_iterator

Clase iteradora constante sobre el map. Se mueve de forma constante por el map.

`#include <Cronologia.h>`

Métodos públicos

- const_iterator & operator++ ()
  Sobrecarga del operador ++.
- const_iterator & operator-- ()
  Sobrecarga del operador –.
- const pair< const string, EventoHistorico > & operator* ()
  *Sobrecarga del operador *.*
- bool operator== (const const_iterator &i)
  Sobrecarga del operador ==.
- bool operator!= (const const_iterator &i)
  Sobrecarga del operador !=.

Atributos privados

- map< string, EventoHistorico >::const_iterator **it**

Amigas

- class **Cronologia**

5.1.1. Descripción detallada

Clase iteradora constante sobre el map. Se mueve de forma constante por el map.

Definición en la línea 55 del archivo Cronologia.h.

5.1.2. Documentación de las funciones miembro

5.1.2.1. operatori=()

```
bool Cronologia::const_iterator::operator!= (
            const const_iterator & i )  [inline]
```

Sobrecarga del operador !=.

Definición en la línea 79 del archivo Cronologia.h.

5.1.2.2. operator∗()

```
const pair<const string, EventoHistorico>& Cronologia::const_iterator::operator* ( )  [inline]
```

Sobrecarga del operador ∗.

Devuelve

Devuelve un pair.

Definición en la línea 71 del archivo Cronologia.h.

5.1.2.3. operator++()

```
const_iterator& Cronologia::const_iterator::operator++ ( )  [inline]
```

Sobrecarga del operador ++.

Definición en la línea 62 del archivo Cronologia.h.

5.1.2.4. operator--()

```
const_iterator& Cronologia::const_iterator::operator-- ( )  [inline]
```

Sobrecarga del operador –.

Definición en la línea 66 del archivo Cronologia.h.

5.1.2.5. operator==()

```
bool Cronologia::const_iterator::operator== (
            const const_iterator & i )  [inline]
```

Sobrecarga del operador ==.

Definición en la línea 75 del archivo Cronologia.h.

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

- include/Cronologia.h

5.2. Referencia de la Clase EventoHistorico::const_iterator

Clase iteradora constante sobre el pair. Se mueve de forma constante por el pair.

`#include <EventoHistorico.h>`

Métodos públicos

- const_iterator & operator++ ()
  Sobrecarga del operador ++.
- const_iterator & operator-- ()
  Sobrecarga del operador –.
- const string operator* ()
  *Sobrecarga del operador *.*
- bool operator== (const const_iterator &i)
  Sobrecarga del operador ==.
- bool operator!= (const const_iterator &i)
  Sobrecarga del operador !=.

Atributos privados

- set< string >::const_iterator **it**

Amigas

- class **EventoHistorico**

5.2.1. Descripción detallada

Clase iteradora constante sobre el pair. Se mueve de forma constante por el pair.

Definición en la línea 45 del archivo EventoHistorico.h.

5.2.2. Documentación de las funciones miembro

5.2.2.1. operatori=()

```
bool EventoHistorico::const_iterator::operator!= (
            const const_iterator & i )  [inline]
```

Sobrecarga del operador !=.

Definición en la línea 69 del archivo EventoHistorico.h.

5.2.2.2. operator∗()

```
const string EventoHistorico::const_iterator::operator* ( )  [inline]
```

Sobrecarga del operador ∗.

Devuelve

Devuelve un pair.

Definición en la línea 61 del archivo EventoHistorico.h.

5.2.2.3. operator++()

```
const_iterator& EventoHistorico::const_iterator::operator++ ( )  [inline]
```

Sobrecarga del operador ++.

Definición en la línea 52 del archivo EventoHistorico.h.

5.2.2.4. operator--()

```
const_iterator& EventoHistorico::const_iterator::operator-- ( )  [inline]
```

Sobrecarga del operador –.

Definición en la línea 56 del archivo EventoHistorico.h.

5.2.2.5. operator==()

```
bool EventoHistorico::const_iterator::operator== (
            const const_iterator & i )  [inline]
```

Sobrecarga del operador ==.

Definición en la línea 65 del archivo EventoHistorico.h.

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

- include/EventoHistorico.h

5.3. Referencia de la Clase Cronologia

TDA Cronología. Una instancia *c* del tipo de datos abstracto `Cronología` es un objeto contenedor de Eventos↩ Historicos representados como un map de la STL con: [[Key][Value]].

```
#include <Cronologia.h>
```

Clases

- class const_iterator

  Clase iteradora constante sobre el map. Se mueve de forma constante por el map.

- class iterator

  Clase iteradora sobre el map. Se mueve de forma NO constante por el map.

Métodos públicos

- const_iterator cfind (string key) const

  Inicializa el iterador con parámetros. Inicializa el iterador con la posición perteneciente a una key específica, forma constante.

- const_iterator cbegin () const

  Inicializa el iterador, sin parámetros. Iterador constante.

- const_iterator cend () const

  Inicializa el iterador, sin parámetros. Iterador constante.

- iterator find (string key)

  Inicializa el iterador con parámetros. Inicializa el iterador con la posición perteneciente a una key específica, forma constante.

- iterator begin ()

  Inicializa el iterador, sin parámetros. Iterador NO constante.

- iterator end ()

  Inicializa el iterador, sin parámetros. Iterador NO constante.

- EventoHistorico GetEventos (int anio)

  Consulta los EventosHistoricos de un año.

- void insertaEvento (EventoHistorico e)

  Inserta un evento en el map. En la key del objeto que entra por parámetro será la misma que la del map.

Atributos privados

- map< string, EventoHistorico > crono

Amigas

- ostream & operator<< (ostream &os, Cronologia &crono)

  Sobrecarga del operador <<. Operador que se usa para mostrar un objeto de tipo Cronologia.

- istream & operator>> (istream &is, Cronologia &crono)

  Sobrecarga del operador >>. Operador que se usa para leer de un fichero.

5.3.1. Descripción detallada

TDA Cronología. Una instancia *c* del tipo de datos abstracto `Cronología` es un objeto contenedor de Eventos↩ Historicos representados como un map de la STL con: [[Key][Value]].

Los ejemplos de su uso los encontramos en: pruebaronologia.cpp:

```
#include "Cronologia.h"
#include <iostream>

using namespace std;

int main(int argc, char * argv[]){

  if (argc!=2){
      cout«"Dime el nombre del fichero con la cronologia"«endl;
      return 0;
   }

   ifstream f (argv[1]);
   if (!f){
    cout«"No puedo abrir el fichero "«argv[1]«endl;
    return 0;
   }

   Cronologia mi_cronologia;
   f»mi_cronologia; //Cargamos en memoria, en el traductor.
   int anio;
   cout«"Dime un año a consultar: ";
   cin » anio;

   EventoHistorico eventos;
   eventos = mi_cronologia.GetEventos(anio);  //Asumimos que Cronologia::GetEventos() devuelve objeto de
       clase EventoHistorico

   // Recorremos con iterador los acontecimientos para mostrarlos por pantalla
   // Este proceso requiere la definición de un tipo iterator
   // const_iterator en EventoHistorico
   // Y la definición de los métodos begin() y end() en EventoHistorico
   EventoHistorico::const_iterator it;
   cout « anio « ": " « endl;              //Imprimimos el anio
   for (it=eventos.cbegin(); it!=eventos.cend();++it){
     cout«" ---> "« (*it);
     cout«endl;
   }

   return 0;
}
/*
 }   */
```

union_cronologia.cpp:

```
#include "Cronologia.h"
#include <fstream>
#include <iostream>

using namespace std;

// Esta funcion también puede implementarse como método de la clase Cronologia
void Union(const Cronologia & c1, const Cronologia & c2,
      Cronologia &resultado){
  Cronologia::const_iterator it1;
  resultado = c1;
  for (it1 = resultado.cbegin(); it1 != resultado.cend(); ++it1) {
    EventoHistorico nuevo;
    string key = (*it1).first;
    nuevo.aniadeAnio(key);
    Cronologia::const_iterator it2 = c2.cfind(key);
    if (it2 != c2.cend()) {    // existe la key.S
      EventoHistorico::const_iterator it_ev1 = (*it1).second.cbegin();
      EventoHistorico::const_iterator it_ev2 = (*it2).second.cbegin();
      while (it_ev1 != (*it1).second.cend()){
        while (it_ev2 != (*it2).second.cend()) {
          if (*it_ev1 != *it_ev2){
            nuevo.aniadeEvento((*it_ev1));
            nuevo.aniadeEvento((*it_ev2));
```

```
            resultado.insertaEvento(nuevo);
          }
          ++it_ev2;
        }
        ++it_ev1;
      }
    }
  }
}

// Este método también puede implementarse como operator« asociado a la clase Cronología (A vuestra
      elección).
void ImprimeCronologia (const Cronologia &c, ostream &os){
   Cronologia::const_iterator it;
   for (it=c.cbegin(); it!=c.cend();++it){
       os«(*it).first«"#";          //año esta en el key del map
       EventoHistorico::const_iterator it_ev;
       for (it_ev=(*it).second.cbegin(); it_ev!=(*it).second.cend();++it_ev)
        os«(*it_ev)«"#";
   }
}

int main(int argc, char * argv[]){

  if (argc!=3 && argc!=4){
      cout«"Error: debe dar al menos los nombres de dos ficheros con cronologías. "«endl;
      cout«"[Opcional]: un tercer nombre de fichero para guardar la cronología resultante."«endl;
      return 0;
   }

   ifstream f1 (argv[1]);
   if (!f1){
    cout«"No puedo abrir el fichero "«argv[1]«endl;
    return 0;
   }
   ifstream f2 (argv[2]);
   if (!f2){
    cout«"No puedo abrir el fichero "«argv[2]«endl;
    return 0;
   }

   Cronologia c1, c2, cUnion;
   f1 » c1;    // Cargamos los datos de los ficheros en las cronologías.
   f2 » c2;

   Union(c1, c2, cUnion);

   if (argc==3)   //No se dio fichero de salida, imprimimos en cout
      cout « cUnion;
   else{
     ofstream fout(argv[3]);
     if (!fout){
      cout«"No puedo crear el fichero "«argv[3]«endl;
      return 0;
     }
    fout « cUnion;
   }
}
```

filtrado_palabra.cpp:

```
#include "Cronologia.h"
#include <fstream>

using namespace std;

void FiltradoPalabra(const Cronologia &c, string palabra, Cronologia &r){
  string key;
  Cronologia::const_iterator it1;
  EventoHistorico::const_iterator it_ev1, it_ev2;
  EventoHistorico nuevo;
  for (it1 = c.cbegin(); it1 != c.cend(); ++it1) {
    key = (*it1).first;         // fecha a buscar
    nuevo.aniadeAnio(key);
    it_ev1 = (*it1).second.cbegin();
    while (!(*it_ev1).find(palabra)) {
      nuevo.aniadeEvento((*it_ev1));
      r.insertaEvento(nuevo);
      ++it_ev1;
    }
  }
}
```

```
int main(int argc, char const *argv[]) {

  string palabra;
  Cronologia crono;

  if (argc < 0) {
    cout « "Error en el número de argumentos" « endl;
    cout « "Uso: %s <cronologia1.txt> <palabra a buscar <fichero_salida.txt>" « endl;
  }

  if (argc == 2) {
    cout « "Palabra a buscar" « endl;
    cin » palabra;
    cout « "Salida: Pantalla" « endl;
    //ofstream f2 (cout);
  }

  if (argc == 3) {
    palabra = argv[2];
    std::cout « "Salida por pantalla:" « endl;
  }

  if (argc == 4) {
    palabra = argv[2];
    std::cout « "Salida al archivo: " « argv[3] « endl;
  }

  ifstream f1 (argv[1]);

  if (!f1){
    cout « "No puedo abrir el fichero " « argv[1] « endl;
    return 0;
  }

  f1 » crono;
  Cronologia crono_palabra;
  FiltradoPalabra(crono, palabra, crono_palabra);

  if (argc == 3)
    std::cout « crono « '\n';
  else{
    ofstream fout(argv[3]);
    if (!fout){
     cout«"No puedo crear el fichero "«argv[3]«endl;
     return 0;
    }
    fout « crono;
  }
  return 0;
}
```

filtrado_intervalo:

```
#include "Cronologia.h"
#include <fstream>
#include <iostream>

using namespace std;


void FiltradoIntervalo(Cronologia &c, char* a1, char* a2, Cronologia &r) {
  string  key1 = a1, key2 = a2;
  Cronologia::const_iterator it1;
  EventoHistorico::const_iterator it_ev1, it_ev2;
  EventoHistorico nuevo;
  for (it1 = c.find(key1); it1 != c.find(key2); ++it1) {
    nuevo.aniadeAnio(key);
    it_ev1 = (*it1).cbegin();
    while ( it_ev1 != (*it1).cend() ){
      nuevo.aniadeEvento(*it_ev1);
      r.insertaEvento(nuevo);
      ++it_ev1;
    }
  }
}

int main(int argc, char const *argv[]) {

  return 0;
}
```

Autor

Antonio Miguel Morillo Chica

Fecha

10 diciembre 2016

Definición en la línea 34 del archivo Cronologia.h.

5.3.2. Documentación de las funciones miembro

5.3.2.1. begin()

```
iterator Cronologia::begin ( )  [inline]
```

Inicializa el iterador, sin parámetros. Iterador NO constante.

Devuelve

devuelve la posición de inicial.

Definición en la línea 146 del archivo Cronologia.h.

5.3.2.2. cbegin()

```
const_iterator Cronologia::cbegin ( ) const  [inline]
```

Inicializa el iterador, sin parámetros. Iterador constante.

Devuelve

devuelve la posición de inicio.

Definición en la línea 126 del archivo Cronologia.h.

5.3.2.3. cend()

```
const_iterator Cronologia::cend ( ) const  [inline]
```

Inicializa el iterador, sin parámetros. Iterador constante.

Devuelve

devuelve la posición de final.

Definición en la línea 132 del archivo Cronologia.h.

5.3.2.4. cfind()

```
const_iterator Cronologia::cfind (
            string key ) const  [inline]
```

Inicializa el iterador con parámetros. Inicializa el iterador con la posición perteneciente a una key específica, forma constante.

Parámetros

key	año a buscar

Devuelve

devuelve un iterador desde la posición del año.

Definición en la línea 120 del archivo Cronologia.h.

5.3.2.5. end()

```
iterator Cronologia::end ( )  [inline]
```

Inicializa el iterador, sin parámetros. Iterador NO constante.

Devuelve

devuelve la posición final.

Definición en la línea 152 del archivo Cronologia.h.

Hace referencia a GetEventos().

5.3.2.6. find()

```
iterator Cronologia::find (
            string key )  [inline]
```

Inicializa el iterador con parámetros. Inicializa el iterador con la posición perteneciente a una key específica, forma constante.

Parámetros

key	año a buscar

Devuelve

devuelve un iterador desde la posición del año.

Definición en la línea 140 del archivo Cronologia.h.

5.3.2.7. GetEventos()

```
EventoHistorico Cronologia::GetEventos (
            int anio )
```

Consulta los EventosHistoricos de un año.

Parámetros

anio	fecha a consultar.

Devuelve

devuelve un objeto de tipo EventoHistorico.

Referenciado por end().

5.3.2.8. insertaEvento()

```
void Cronologia::insertaEvento (
            EventoHistorico e )  [inline]
```

Inserta un evento en el map. En la key del objeto que entra por parámetro será la misma que la del map.

Parámetros

e	EventoHistorico que queremos añadir.

Definición en la línea 165 del archivo Cronologia.h.

Hace referencia a EventoHistorico::getFirst(), operator<< y operator>>.

5.3.3. Documentación de las funciones relacionadas y clases amigas

5.3.3.1. operator<<

```
ostream& operator<< (
            ostream & os,
            Cronologia & crono )  [friend]
```

Sobrecarga del operador <<. Operador que se usa para mostrar un objeto de tipo Cronologia.

Referenciado por insertaEvento().

5.3.3.2. operator>>

```
istream& operator>> (
            istream & is,
            Cronologia & crono )  [friend]
```

Sobrecarga del operador >>. Operador que se usa para leer de un fichero.

Referenciado por insertaEvento().

5.3.4. Documentación de los datos miembro

5.3.4.1. crono

```
map<string, EventoHistorico> Cronologia::crono  [private]
```

map de la stl

Definición en la línea 49 del archivo Cronologia.h.

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

- include/Cronologia.h

5.4. Referencia de la Clase EventoHistorico

TDA EventoHistorico. Una instancia *c* del tipo de datos abstracto `EventoHistorico` es un objeto pair con un contenedor set que almacenan una fecha y una sucesión de eventos alojados en un set.

```
#include <EventoHistorico.h>
```

Clases

- class const_iterator

  Clase iteradora constante sobre el pair. Se mueve de forma constante por el pair.
- class iterator

  Clase iteradora constante sobre el pair. Se mueve de forma NO constante por el pair.

Métodos públicos

- const_iterator cbegin () const

  Inicializa el iterador, sin parámetros. Iterador constante.
- const_iterator cend () const

  Inicializa el iterador, sin parámetros. Iterador constante.
- iterator begin ()

  Inicializa el iterador, sin parámetros. Iterador NO constante.
- iterator end ()

  Inicializa el iterador, sin parámetros. Iterador NO constante.
- string getFirst ()

  Consultor de la key.
- void aniadeAnio (string a)

  Modificador de la clave.
- void aniadeEvento (string e)

  Modificador del contendor.
- void insertaEventoHistorico (EventoHistorico e)

  Inserta un EventoHistorico completo.

Atributos privados

- pair< string, set< string > > eventos

Amigas

- ostream & operator<< (ostream &os, EventoHistorico &e)

  Sobrecarga dle operador <<. El operador << se usa para mostrar un evento. Simplemente itera por el y lo va mostrando.

5.4.1. Descripción detallada

TDA EventoHistorico. Una instancia *c* del tipo de datos abstracto `EventoHistorico` es un objeto pair con un contenedor set que almacenan una fecha y una sucesión de eventos alojados en un set.

Los ejemplos de su uso los encontramos en los mismos archivos que hemos visto para Cronología.

Autor

Antonio Miguel Morillo Chica

Fecha

10 diciembre 2016

Definición en la línea 26 del archivo EventoHistorico.h.

5.4.2. Documentación de las funciones miembro

5.4.2.1. aniadeAnio()

```
void EventoHistorico::aniadeAnio (
            string a )  [inline]
```

Modificador de la clave.

Parámetros

a	clave que queremos asignar a nuestro contenedor.

Definición en la línea 136 del archivo EventoHistorico.h.

5.4.2.2. aniadeEvento()

```
void EventoHistorico::aniadeEvento (
            string e )  [inline]
```

Modificador del contendor.

Parámetros

e	evento que se quiere insertar en el set del pair.

Definición en la línea 141 del archivo EventoHistorico.h.

Hace referencia a insertaEventoHistorico() y operator<<.

5.4.2.3. begin()

```
iterator EventoHistorico::begin ( )  [inline]
```

Inicializa el iterador, sin parámetros. Iterador NO constante.

Devuelve

devuelve la posición de inicial.

Definición en la línea 120 del archivo EventoHistorico.h.

5.4.2.4. cbegin()

```
const_iterator EventoHistorico::cbegin ( ) const  [inline]
```

Inicializa el iterador, sin parámetros. Iterador constante.

Devuelve

devuelve la posición de inicio.

Definición en la línea 108 del archivo EventoHistorico.h.

5.4.2.5. cend()

```
const_iterator EventoHistorico::cend ( ) const  [inline]
```

Inicializa el iterador, sin parámetros. Iterador constante.

Devuelve

devuelve la posición de final.

Definición en la línea 114 del archivo EventoHistorico.h.

5.4.2.6. end()

```
iterator EventoHistorico::end ( )  [inline]
```

Inicializa el iterador, sin parámetros. Iterador NO constante.

Devuelve

devuelve la posición final.

Definición en la línea 126 del archivo EventoHistorico.h.

5.4.2.7. getFirst()

```
string EventoHistorico::getFirst ( )  [inline]
```

Consultor de la key.

Devuelve

devuelve la key.

Definición en la línea 131 del archivo EventoHistorico.h.

Referenciado por Cronologia::insertaEvento().

5.4.2.8. insertaEventoHistorico()

```
void EventoHistorico::insertaEventoHistorico (
            EventoHistorico e )
```

Inserta un EventoHistorico completo.

Parámetros

e	evento historico a añadir.

Referenciado por aniadeEvento().

5.4.3. Documentación de las funciones relacionadas y clases amigas

5.4.3.1. operator<<

```
ostream& operator<< (
            ostream & os,
            EventoHistorico & e )  [friend]
```

Sobrecarga dle operador <<. El operador << se usa para mostrar un evento. Simplemente itera por el y lo va mostrando.

Referenciado por aniadeEvento().

5.4.4. Documentación de los datos miembro

5.4.4.1. eventos

```
pair<string, set<string> > EventoHistorico::eventos  [private]
```

pair de la STL

Definición en la línea 39 del archivo EventoHistorico.h.

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

- include/EventoHistorico.h

5.5. Referencia de la Clase EventoHistorico::iterator

Clase iteradora constante sobre el pair. Se mueve de forma NO constante por el pair.

```
#include <EventoHistorico.h>
```

Métodos públicos

- iterator & operator++ ()
  Sobrecarga del operador ++.
- iterator & operator-- ()
  Sobrecarga del operador –.
- string operator* ()
  *Sobrecarga del operador *.*
- bool operator== (const iterator &i)
  Sobrecarga del operador ==.
- bool operator!= (const iterator &i)
  Sobrecarga del operador !=.

Atributos privados

- set< string >::iterator **it**

Amigas

- class **EventoHistorico**

5.5.1. Descripción detallada

Clase iteradora constante sobre el pair. Se mueve de forma NO constante por el pair.

Definición en la línea 76 del archivo EventoHistorico.h.

5.5.2. Documentación de las funciones miembro

5.5.2.1. operatori=()

```
bool EventoHistorico::iterator::operator!= (
            const iterator & i )  [inline]
```

Sobrecarga del operador !=.

Definición en la línea 100 del archivo EventoHistorico.h.

5.5.2.2. operator∗()

```
string EventoHistorico::iterator::operator* ( )  [inline]
```

Sobrecarga del operador ∗.

Devuelve

Devuelve un pair.

Definición en la línea 92 del archivo EventoHistorico.h.

5.5.2.3. operator++()

```
iterator& EventoHistorico::iterator::operator++ ( )  [inline]
```

Sobrecarga del operador ++.

Definición en la línea 83 del archivo EventoHistorico.h.

5.5.2.4. operator--()

```
iterator& EventoHistorico::iterator::operator-- ( )  [inline]
```

Sobrecarga del operador –.

Definición en la línea 87 del archivo EventoHistorico.h.

5.5.2.5. operator==()

```
bool EventoHistorico::iterator::operator== (
            const iterator & i )  [inline]
```

Sobrecarga del operador ==.

Definición en la línea 96 del archivo EventoHistorico.h.

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

- include/EventoHistorico.h

5.6. Referencia de la Clase Cronologia::iterator

Clase iteradora sobre el map. Se mueve de forma NO constante por el map.

`#include <Cronologia.h>`

Métodos públicos

- iterator & operator++ ()
  Sobrecarga del operador ++.
- iterator & operator-- ()
  Sobrecarga del operador –.
- pair< const string, EventoHistorico > & operator* ()
  *Sobrecarga del operador *.*
- bool operator== (const iterator &i)
  Sobrecarga del operador ==.
- bool operator!= (const iterator &i)
  Sobrecarga del operador !=.

Atributos privados

- map< string, EventoHistorico >::iterator **it**

Amigas

- class **Cronologia**

5.6.1. Descripción detallada

Clase iteradora sobre el map. Se mueve de forma NO constante por el map.

Definición en la línea 86 del archivo Cronologia.h.

5.6.2. Documentación de las funciones miembro

5.6.2.1. operatori=()

```
bool Cronologia::iterator::operator!= (
            const iterator & i )  [inline]
```

Sobrecarga del operador !=.

Definición en la línea 110 del archivo Cronologia.h.

5.6.2.2. operator*()

```
pair<const string, EventoHistorico>& Cronologia::iterator::operator* ( )  [inline]
```

Sobrecarga del operador *.

Devuelve

Devuelve un pair.

Definición en la línea 102 del archivo Cronologia.h.

5.6.2.3. operator++()

```
iterator& Cronologia::iterator::operator++ ( )  [inline]
```

Sobrecarga del operador ++.

Definición en la línea 93 del archivo Cronologia.h.

5.6.2.4. operator--()

```
iterator& Cronologia::iterator::operator-- ( )  [inline]
```

Sobrecarga del operador –.

Definición en la línea 97 del archivo Cronologia.h.

5.6.2.5. operator==()

```
bool Cronologia::iterator::operator== (
            const iterator & i )  [inline]
```

Sobrecarga del operador ==.

Definición en la línea 106 del archivo Cronologia.h.

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

- include/Cronologia.h

Capítulo 6

Documentación de archivos

6.1. Referencia del Archivo include/Cronologia.h

Fichero cabecera del TDA Cronologia.

```
#include <iostream>
#include <string>
#include <map>
#include "EventoHistorico.h"
```

Clases

- class Cronologia

  TDA Cronología. Una instancia c del tipo de datos abstracto `Cronología` es un objeto contenedor de Eventos↩ Historicos representados como un map de la STL con: [[Key][Value]].
- class Cronologia::const_iterator

  Clase iteradora constante sobre el map. Se mueve de forma constante por el map.
- class Cronologia::iterator

  Clase iteradora sobre el map. Se mueve de forma NO constante por el map.

6.1.1. Descripción detallada

Fichero cabecera del TDA Cronologia.

6.2. Referencia del Archivo include/EventoHistorico.h

Fichero cabecera del TDA EventoHistorico.

```
#include <iostream>
#include <string>
#include <set>
#include <utility>
```

Clases

- class EventoHistorico

  TDA EventoHistorico. Una instancia c del tipo de datos abstracto `EventoHistorico` es un objeto pair con un contenedor set que almacenan una fecha y una sucesión de eventos alojados en un set.
- class EventoHistorico::const_iterator

  Clase iteradora constante sobre el pair. Se mueve de forma constante por el pair.
- class EventoHistorico::iterator

  Clase iteradora constante sobre el pair. Se mueve de forma NO constante por el pair.

6.2.1. Descripción detallada

Fichero cabecera del TDA EventoHistorico.

Índice alfabético