Diccionario<K,D>::iterator

Practica3

Generado por Doxygen 1.7.5

Lunes, 18 de Noviembre de 2013 13:39:28

Índice general

Capítulo 1

Iterando sobre el diccionario

Versión

v0

Autor

Juan F. Huete

1.1. Introducción

En la práctica anterior permitimos parametrizar un diccionario mediante el uso de plantillas (templates). En esta práctica nuestro objetivo es dotar al diccionario de un mecanismo para poder iterar por el conjunto de 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 diccionario) de otros objetos (entradas en el diccionario). 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 como dicho contenedor esta implementado internamente.

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 tgz 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 dox_diccionario
```

Como resultado nos genera dos directorios, uno con la documentación en html y el otro con la documentación en latex. Para generar la documentación en pdf podemos ejecutar

```
cd latex
make
```

Al hacer make se ejecuta una llamada al programa latex (si lo tenemos instalado) que como salida nos genera el fichero refman.pdf

De igual forma que en la práctica anterior, se entrega la práctica de templates corregida, eso si, sólo sobre la versión DICC_V1. Los cambios para la versión V2 son similares, y por tanto en caso de no haber hecho esta parte la podeis implementar de forma fácil.

Además, se entregam los ficheros de especificación nueva sobre el diccionario. Estos ficheros incluyen algunas modificaciones que viene dadas por el uso de los iteradores.

- diccionario.h En el nuevo fichero diccionario.h se entrega la nueva especificación de la clase diccionario, 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.
- diccionarioV1.hxx En este fichero se incluyen la implementación de algunos métodos de la clase diccionario utilizando la versión DICC_V1, además se incluye las cabeceras que hacen referencia al iterator, debiendo de añadir también las que hacen referencia al const_iterator.

En la práctica es OBLIGATORIO entregar la versión en la que el diccionario está ordenado, esto es diccionarioV2.hxx.

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

1.3. Iteradores sobre diccionario.

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 contenedor). Ambas clases están estrechamente relacionadas, por lo que es muy usual que se desee que tanto el contenedor como el iterador sean clases amigas. Así, cuando se crea una clase friend anidada es conveniente declarar primero el nombre de la clase y después definir la clase. Así evitamos que se confunda el compilador.

```
template <typename K, typename D>
class diccionario {
public:
   class iterator;  //declaracion previa

   ....
   iterator begin();  // Podemos declarar el iterator porque ya lo hemos
       declarado previamente

   ....
   class iterator {
      //definicion del iterator
      public:
        iterator();
```

```
        ....
      private:
         friend class diccionario; // declaramos diccionario 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 diccionario
```

Es importante notar que el tipo asociado al iterador es diccionario<K,D>::iterator Por tanto, para declarar un diccionario y un iterator sobre dicho diccionario debemos hacer

```
diccionario<string,int> dic;
diccionario<string,int>::iterator it;

diccionario<int,float> d2;
diccionario<int,float>::const_iterator it2;
```

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(). begin() devuelve un iterator que apunta al principio del contenedor (primer elemento). Sin embargo, 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()).

Para imprimir todos los elementos del diccionario podríamos hacer

```
for (it=dic.begin(); it!=dic.end();++it){
    cout << (*it).first << (*it).second << endl;
}
```

Obviamente, no es posible realizar la asignación it = d2.begin() pues el iterador it está definido para iterar sobre valores de tipo pares de string,int y no sobre entradas int,float. Además podemos ver el uso de paréntesis para acceder a los elementos (∗it).first. - En este caso, si hacemos ∗it.first, dada la precedencia de los operadores, primero se evaluaría it.first.

1.5. Modificación en la especificación de algunos métodos.

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

- typename diccionario<K,D>::iterator diccionario<K,D>::find(const Key & s) ;

En este caso busca una entrada en el diccionario. Si la encuentra devuelve el iterador que apunta a la entrada, en caso contrario devuelve end().

```
diccionario<K,D>::iterator  diccionario<K,D>::find( const Key & s) ;
....
Ejemplo de uso:
diccionario<string,int>::iterator it;
it = Dic.find("Hola");
if (it == Dic.end()) cout << "No esta " << endl;
else cout << "Clave "<< (*it).first << " Definicion" << (*it).second;
```

- diccionario<K,D>::const_iterator diccionario<K,D>::find(const Key & s) const;

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

- null ()

En este caso, ya no es necesario de disponer del método null, pues podemos utilizar la comparación anterior para determinar que la entrada no está en el diccionario.

- max_elemen()

Como tenemos acceso a los elementos del diccionario, ya no es necesario el método por lo que lo dejaremos como método externo a la clase, y deberemos de implementarlo haciendo uso de los iteradores.

1.6. Representacion del iterator

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

```
class diccionario{
....
  class iterator {
    ....
    entrada & operator*();
    ......
    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

```
Dic["Hola"] = 24;
...
it = Dic.find("Hola");
```

```
if (it == Dic.end()) cout << "No esta " << endl;
else {
 (*it).first ="PERRO";
}
```

Esto nos daría problemas pues estaríamos modificando la clase, y particularmente en la versión DICC_V2, donde se asumen los datos ordenados, el diccionario podría dejar de cumplir el invariante de la representación. A partir de este momento las operaciones de búsqueda e inserción dejarían de funcionar correctamente.

Para solucionar el problema se incluye una nueva definición de tipo en el diccionario, en concreto se define value_type como un par pair<const Key, Def>. En este caso, la clave es constante y no se puede realizar ninguna modificación a la misma. Inicialmente se os pide trabajar con la versión que devuelva la referencia a la entrada.

1.6.1. Trabajando con value_type

En esta sección se incluye información para poder garantizar que se satisface el invariante de la representación siempre.

En este caso el operator* deberá devolver una referencia a un elemento de tipo value_type

```
class diccionario{
typedef pair<const Key, Def> value_type;

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

Por lo tanto, en la práctica necesitaremos de un mecanismo para transformar un puntero a pair<K,D> a un puntero a pair<const K,D>. Esto lo podemos hacer utilizando un casting, como ilustra el siguiente código (fichero prueba.cpp) .

```
#include <vector>
#include <iostream>
#include <map>

using namespace std;

int main(){

vector<pair<string,int> > V;
pair<string,int> aux("Hola",2);
pair<string,int> * p_e;
pair<const string,int> * p_vt;

V.push_back(aux);
```

```
p_e = &V[0]; // tomamos la direccion del primer elemento del vector
cout << (*p_e).first << (*p_e).second<< endl;
(*p_e).first = "XXXX"; //Correcto

p_vt = (pair<const string,int> *) &V[0];
cout << (*p_vt).first << (*p_vt).second<< endl;
(*p_vt).second = 234;
(*p_vt).first = "XXXX"; //INCORRECTO
}
```

El error que da el compilador es algo del tipo

ejemplo.cpp: En la función ‘int main()’: ejemplo.cpp:24:17: error: pasar ‘const string’ como el argumento ‘this’ de operator=(const char∗)’ descarta a los calificadores.

1.7. SE PIDE

En concreto se pide implementar los métodos asociados a los iteradores de la clase diccionario considerasndo como implmentacion la version DICC_V2. Así, al iterar sobre los elementos los tendríamos ordenados por clave.

```
for (it=dic.begin(); it!=dic.end();++it){
    cout << (*it).first << (*it).second << endl;
}
```

En este caso, para realizar la práctica, el alumno deberá modificar tanto el fichero de especificación, diccionario.h, de forma que la propia especificación indique que trabaja con parámetros plantilla, como los ficheros de implementación (.hxx).

De igual forma se debe modificar el fichero prueba.cpp de manera que se demuestre el correcto comportamiento del diccionario 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

- diccionario.h Especificación del TDA diccionario.
- diccionarioV2.hxx segunda versión del diccionario.
- prueba.cpp fichero de prueba del diccionario

Dicha entrega se debe realizar antes de el Lunes 2 de Diciembre, a las 9:00 horas (am).

Capítulo 2

Lista de tareas pendientes

Miembro diccionario::begin ()
implementa esta función

Miembro diccionario::const_iterator::operator!= (const iterator &it)
implementa esta función

Miembro diccionario::const_iterator::operator== (const iterator &it)
implementa esta función

Miembro diccionario::end ()
implementa esta función

Miembro diccionario::find (const Key &s)
implementa esta función

Miembro diccionario::iterator::iterator (vector< entrada >::iterator it)
implementa esta función

Miembro diccionario::iterator::iterator ()
implementa esta función

Miembro diccionario::iterator::operator* () const
implementa esta función

Miembro diccionario::iterator::operator++ (int)
implementa esta función

Miembro diccionario::iterator::operator++ ()
implementa esta función

Capítulo 3

Índice de clases

3.1. Lista de clases

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

Capítulo 4

Documentación de las clases

4.1. Referencia de la Clase diccionario::const_iterator

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

`#include <diccionario.h>`

Métodos públicos

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

Métodos privados

- **const_iterator** (vector< entrada >::const_iterator it)

Atributos privados

- vector< entradad >::const_iterator **it_v**

Amigas

- class **diccionario**

4.1.1. Descripción detallada

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

4.1.2. Documentación de las funciones miembro

4.1.2.1. bool diccionario::iterator::operator!= (const iterator & *it*)

Tareas pendientes implementa esta función

Definición en la línea 253 del archivo diccionarioV1.hxx.

4.1.2.2. bool diccionario::iterator::operator== (const iterator & *it*)

Tareas pendientes implementa esta función

Definición en la línea 245 del archivo diccionarioV1.hxx.

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

- diccionario.h
- diccionarioV1.hxx

4.2. Referencia de la Clase diccionario

Clases

- 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 diccionario, Lectura y Escritura iterator ,operator, operator++, operator++(int) operator=, operator==, operator!=*

Tipos públicos

- typedef pair< Key, Def > **entrada**
- typedef unsigned int **size_type**
- typedef pair< const Key, Def > **value_type**

Métodos públicos

- iterator begin ()

  entrada nula del diccionario
- const_iterator begin () const

  entrada nula del diccionario
- diccionario ()

  constructor primitivo.
- diccionario (const entrada &nula)

  constructor primitivo.
- diccionario (const diccionario &d)

  constructor de copia
- bool empty () const

  vacia Chequea si el priority_queue esta vacio (size()==0)
- iterator end ()

  entrada nula del diccionario
- const_iterator end () const

  entrada nula del diccionario
- iterator find (const Key &s)

  busca una cadena en el diccionario
- diccionario & operator= (const diccionario &org)

  operador de asignación
- Def & operator[] (const Key &s)

  busca una cadena en el diccionario
- const Def & operator[] (const Key &s) const

  Consulta una entrada en el diccionario.
- size_type size () const

  numero de entradas en el diccionario

Métodos privados

- bool cheq_rep () const

  Chequea el Invariante de la representacion.

Atributos privados

- vector< entrada > **dic**
- int **pos_max**

Amigas

- class **iterator**
- ostream & operator<< (ostream &sal, const diccionario< Key, Def > &D)

  imprime todas las entradas del diccionario

4.2.1. Documentación del constructor y destructor

4.2.1.1. diccionario::diccionario ( )

constructor primitivo.

Postcondición

Definición en la línea 44 del archivo diccionarioV1.hxx.

4.2.1.2. diccionario::diccionario (const entrada & *nula*)

constructor primitivo.

Parámetros

in	*nula*	representa a la entrada nula para el diccionario

Postcondición

define la entrada nula

4.2.1.3. diccionario::diccionario (const **diccionario** & *d*)

constructor de copia

Parámetros

in	*d*	diccionario a copiar

4.2.2. Documentación de las funciones miembro

4.2.2.1. **diccionario**< Key, Def >::**iterator** diccionario::begin ( )

entrada nula del diccionario

Devuelve

Devuelve el iterador a la primera posición del diccionario.

Postcondición

no modifica el diccionario

Tareas pendientes implementa esta función

Definición en la línea 172 del archivo diccionarioV1.hxx.

4.2.2.2. const_iterator diccionario::begin ( ) const

entrada nula del diccionario

Devuelve

Devuelve el iterador a la primera posición del diccionario.

Postcondición

no modifica el diccionario

4.2.2.3. bool diccionario::cheq_rep ( ) const [private]

Chequea el Invariante de la representacion.

Devuelve

true si el invariante es correcto, falso en caso contrario

Definición en la línea 142 del archivo diccionarioV1.hxx.

4.2.2.4. diccionario< Key, Def >::iterator diccionario::end ( )

entrada nula del diccionario

Devuelve

Devuelve el iterador a la ultima posición del diccionario.

Postcondición

no modifica el diccionario

Tareas pendientes implementa esta función

Definición en la línea 181 del archivo diccionarioV1.hxx.

4.2.2.5. const_iterator diccionario::end ( ) const

entrada nula del diccionario

Devuelve

Devuelve el iterador a la ultima posición del diccionario.

Postcondición

no modifica el diccionario

4.2.2.6. diccionario< Key, Def >::iterator diccionario::find (const Key & *s*)

busca una cadena en el diccionario

Parámetros

s	cadena a buscar

Devuelve

un iterador que apunta a la entrada en el diccionario. Si la palabra s no se encuentra devuelve end()

Postcondición

no modifica el diccionario.

Tareas pendientes implementa esta función

Definición en la línea 71 del archivo diccionarioV1.hxx.

4.2.2.7. diccionario< Key, Def > & diccionario::operator= (const diccionario & *org*)

operador de asignación

Parámetros

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

Definición en la línea 117 del archivo diccionarioV1.hxx.

4.2.2.8. Def & diccionario::operator[] (const Key & *s*)

busca una cadena en el diccionario

Parámetros

s	cadena a buscar

Devuelve

un const_iterador que apunta a la entrada en el diccionario. Si la palabra s no se encuentra devuelve end()

Postcondición

no modifica el diccionario. const_iterator diccionario<K,D>::find(const Key & s) const;

/**Consulta/Inserta una entrada en el diccionario.

Busca la cadena s en el diccionario, si la encuentra devuelve una referencia al numero de ocurrencias de la misma en caso contrario la inserta, con frecuencia cero, devolviendo una referencia a este valor.

Parámetros

in	*s*	cadena a insertar
out	*int*	& referencia a la definicion asociada a la entrada

Postcondición

Si s no esta en el diccionario, el size() sera incrementado en 1.

Definición en la línea 78 del archivo diccionarioV1.hxx.

4.2.2.9. const Def & diccionario::operator[ ] (const Key & *s*) const

Consulta una entrada en el diccionario.

Busca la cadena s en el diccionario, si la encuentra devuelve una referencia constante al numero de ocurrencias de la misma, si no la encuentra da un mensaje de error.

Parámetros

in	*s*	cadena a insertar
out	*int*	& referencia constante a la definicion asociada a la entrada

Postcondición

No se modifica el diccionario.

Definición en la línea 102 del archivo diccionarioV1.hxx.

4.2.2.10. diccionario< Key, Def >::size_type diccionario::size ( ) const

numero de entradas en el diccionario

Postcondición

No se modifica el diccionario.

Definición en la línea 127 del archivo diccionarioV1.hxx.

4.2.3. Documentación de las funciones relacionadas y clases amigas

4.2.3.1. ostream& operator<< (ostream & *sal,* const diccionario< Key, Def > & *D*) [friend]

imprime todas las entradas del diccionario

Postcondición

No se modifica el diccionario.

Tareas pendientes implementar esta funcion

Definición en la línea 160 del archivo diccionarioV1.hxx.

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

- diccionario.h
- diccionarioV1.hxx

4.3. Referencia de la Clase diccionario::iterator

class iterator forward iterador sobre el diccionario, Lectura y Escritura iterator ,operator*, operator++, operator++(int) operator=, operator==, operator!=

`#include <diccionario.h>`

Métodos públicos

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

Métodos privados

- iterator (vector< entrada >::iterator it)

Atributos privados

- vector< entrada >::iterator **it_v**

Amigas

- class **diccionario**

4.3.1. Descripción detallada

class iterator forward iterador sobre el diccionario, Lectura y Escritura iterator ,operator*, operator++, operator++(int) operator=, operator==, operator!=

4.3.2. Documentación del constructor y destructor

4.3.2.1. diccionario::iterator::iterator ( )

Tareas pendientes implementa esta función

Definición en la línea 189 del archivo diccionarioV1.hxx.

4.3.2.2. diccionario::iterator::iterator (vector< entrada >::iterator *it*) [private]

Tareas pendientes implementa esta función

Definición en la línea 213 del archivo diccionarioV1.hxx.

4.3.3. Documentación de las funciones miembro

4.3.3.1. diccionario< Key, Def >::entrada & diccionario::iterator::operator* ( ) const

Tareas pendientes implementa esta función

Definición en la línea 220 del archivo diccionarioV1.hxx.

4.3.3.2. diccionario< Key, Def >::iterator diccionario::iterator::operator++ (int *a*)

Tareas pendientes implementa esta función

Definición en la línea 228 del archivo diccionarioV1.hxx.

4.3.3.3. diccionario< Key, Def >::iterator & diccionario::iterator::operator++ ( )

Tareas pendientes implementa esta función

Definición en la línea 236 del archivo diccionarioV1.hxx.

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

- diccionario.h
- diccionarioV1.hxx