CorrectorOrtografico

Practica3

Generado por Doxygen 1.7.5

Lunes, 4 de Noviembre de 2013 09:40:57

Índice general

Capítulo 1

Diccionario

Versión

v0

Autor

Juan F. Huete

1.1. Introducción

En la práctica anterior se os pidió la implementación del tipo diccionario. En esta práctica el objetivo es doble, por un lado permitir al alumno ver los errores cometidos en la práctica anterior y por otro lado seguir avanzando en el uso de las estructuras de datos, particularmente utilizando plantillas (templates) para la definición de tipos de datos genéricos.

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

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

1.2. CORRECCION DE LA PRACTICA 2.

En decsai podeis encontrar los códigos fuentes de las dos versiones del tipo diccionario, junto con un pequeño fichero de prueba que se ha diseñado para testear la validez de los métodos, prueba.cpp. Como antes, la compilación con una u otra versión se hace definiendo la constante DICC_V1 o DICC_V2 a la hora de compilar, esto es,

```
g++ -o ejecV1 -D DICC_V1 fichero.cpp
g++ -o ejecV2 -D DICC_V2 fichero.cpp
```

Los fichero que se entregan son

- diccionario.h Especificación del TDA diccionario.
- diccionarioV1.hxx primera versión del diccionario.
- diccionarioV2.hxx segunda versión del diccionario.
- prueba.cpp fichero de prueba del diccionario

1.2.1. INVARIANTE DE LA REPRESENTACION (Algunos comentarios)

Como os habreis dado cuenta, es IMPOSIBLE que se verifique el invariante de la representación de forma completa ya que hay una "puerta" por la que se permite el acceso a los datos privada de la clase desde fuera de la misma, en concreto es el operator[]

```
int & diccionario::operator[](const string & s)
```

Un ejemplo del porque se viola el invariante de la representación lo podemos tener cuando, si el valor máximo del diccionario es la entrada ("hola",30) y el usuario de la clase, por ejemplo desde el main, ejecuta el comando dic["hola"]=45. Al salir del método, se modifica el valor de la entrada "hola" en el vector desde la función main: al devorver una referencia al entero (int &) asociado a la clave "hola", es desde el propio main cuando se accede a esa dirección y se almacena en la misma el valor 45, por lo que en este momento no se verificaría el invariante de la representación.

Como consecuencia, el invariante de la representación de la clase diccionario sólo podrá hacer referencia a los valores de la clave, ya que sólo desde los métodos de la clase diccionario se puede acceder a la misma. Por tanto esto nos garantiza que no podemos modificarla. Sin embargo, o no es posible imponer restricciones sobre los valores de la definición, pues son modificables desde fuera de la clase.

1.2.2. SE PIDE

El alumno debe comparar su implementación con los distintos métodos entregados, hacer una crítica de los mismos, enviándome sus opiniones en un fichero pdf. En este caso, no es suficiente con indicar el hecho de que la implementaciones sean diferentes, sino en caso de serlo indicar cuáles han sido las dificultades encontradas a la hora de realizar la práctica.

Dicha crítica se debe entregar el lunes 11 de Noviembre, a las 23:59 horas.

1.3. USO DE PLANTILLAS (TEMPLATES)

La segunda parte de la práctica está destinada al manejo de templates por parte del alumno. En concreto se pide dotar al diccionario la capacidad de poder definir un diccionario utilizando cualquier tipo de valores. Así, podriamos tener

```
diccionario<string,int> D1;
diccionario<int, string> D2;
diccionario<string,string> D3;
diccionario<float,string> D4;


...

D1["Hola"]=3;
D3["Hola"]="tres";
D4[3.2]="tres con dos";
...
if (D1.find("Hola")==D1.null()) ....
```

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.

1.3.1. SE PIDE

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

- diccionario.h Especificación del TDA diccionario.
- diccionarioV1.hxx primera versión del diccionario.
- diccionarioV2.hxx segunda versión del diccionario.
- prueba.cpp fichero de prueba del diccionario

Dicha entrega se debe realizar antes de el Lunes 18 de Noviembre, a las 9:00 horas (am).

Capítulo 2

Lista de tareas pendientes

Miembro diccionario::cheq_rep () const

implementa esta función

implementa esta función

Miembro diccionario::diccionario ()

Chequear esta función

implementa esta función

Miembro diccionario::diccionario (const entrada &nula)

implementa esta función

implementa esta función

Miembro diccionario::diccionario (const diccionario &d)

implementa esta función

implementa esta función

Miembro diccionario::empty () const

implementa esta función

implementa esta función

Miembro diccionario::find (const string &s) const

implementa esta función

implementa esta función

Miembro diccionario::max_element () const

implementa esta función

implementa esta función

Miembro diccionario::operator= (const diccionario &org)

implementa esta función

implementa esta función

Miembro diccionario::operator[] (const string &s) const

implementa esta función

implementa esta función

Miembro diccionario::operator[] (const string &s)

implementa esta función

implementa esta función

Miembro diccionario::size () const

implementa esta funcion.

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 corrector

Métodos públicos

- std::string correct (const std::string &word)

  determina la palabra corregida
- void load (const std::string &filename)

  lectura del fichero de entrenamiento

Métodos privados

- void edits (const std::string &word, std::vector< std::string > &result)

  Genera todas las posibles modificaciones tras una "edicion" de la cadena word.
- void known (std::vector< std::string > &results, diccionario &candidates)

  busca ocurrencias en un diccionario

Atributos privados

- diccionario **dictionary**

4.1.1. Documentación de las funciones miembro

4.1.1.1. std::string corrector::correct (const std::string & *word*) [inline]

determina la palabra corregida

Parámetros

in	*word*	palabra origen

Devuelve

palabra que se sugiere como correccion.

Definición en la línea 97 del archivo corrector.h.

4.1.1.2. void corrector::edits (const std::string & *word,* std::vector< std::string > & *result*) [inline, private]

Genera todas las posibles modificaciones tras una "edicion" de la cadena word.

Parámetros

in	*word*	cadena de entrada
out	*result*	vector con las posibles palabras que se obtienen al realizar borrados, transposiciones, alteraciones o inserciones en la cadena word.

Definición en la línea 36 del archivo corrector.h.

4.1.1.3. void corrector::known (std::vector< std::string > & *results,* diccionario & *candidates*) [inline, private]

busca ocurrencias en un diccionario

Parámetros

in	*results*	conjunto de palabras a buscar
in,out	*candidates*	conjunto de palabras en results cuyas entradas tambien se encuentra en el diccionario

Definición en la línea 53 del archivo corrector.h.

4.1.1.4. void corrector::load (const std::string & *filename*) [inline]

lectura del fichero de entrenamiento

Parámetros

in	*filename*	nombre del fichero a leer

Definición en la línea 65 del archivo corrector.h.

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

- corrector.h

4.2. Referencia de la Clase diccionario

Clase diccionario.

`#include <diccionario.h>`

Tipos públicos

- typedef pair< string, int > **entrada**
- typedef unsigned int **size_type**

Métodos públicos

- 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)
- const entrada & find (const string &s) const

  busca una cadena en el diccionario
- const string & max_element () const

  devuelve una referencia constante a la entrada con un mayor numero de ocurrencias en el diccionario.
- const entrada & null () const

  entrada nula del diccionario
- diccionario & operator= (const diccionario &org)

  operador de asignación
- int & operator[] (const string &s)

  Consulta/Inserta una entrada en el diccionario.
- const int & operator[] (const string &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

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

  imprime todas las entradas del diccionario

4.2.1. Descripción detallada

Clase diccionario.

diccionario:: diccionario, null, find, operator[], size, max_element Tipos diccionario-::entrada, diccionario::size_type Descripción

Un diccionario es un contenedor que permite almacenar un conjunto de pares de elementos, el primero será la clave que deber ser única y el segundo la definición. En nuestro caso el diccionario va a tener un subconjunto restringido de métodos (inserción de elementos, consulta de un elemento por clave, además de la consulta del elemento con mayor valor en la definición). Este diccionario "simulará" un diccionario de la stl, con algunas claras diferencias pue, entre otros, no estará dotado de la capacidad de iterar (recorrer) a través de sus elementos.

Asociado al diccionario, tendremos el tipo

```
diccionario::entrada
```

que permite hacer referencia al par de elementos almacenados en cada una de las posiciones del diccionario. Así, el primer campo de una entrada, first, representa la clave y el segundo campo, second, representa la definición. En nuestra aplicación concreta, la clave será un string representando una palabra válida del diccionario y el segundo campo es un entero que hace referencia a la frecuencia de ocurrencia de la palabra en el lenguaje.

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

4.2.2. Documentación del constructor y destructor

4.2.2.1. diccionario::diccionario ( )

constructor primitivo.

Postcondición

define la entrada nula como el par ("",-1)

Tareas pendientes Chequear esta función

Tareas pendientes implementa esta función

Definición en la línea 46 del archivo diccionarioV1.hxx.

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

constructor primitivo.

Parámetros

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

Postcondición

define la entrada nula

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

Definición en la línea 54 del archivo diccionarioV1.hxx.

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

constructor de copia

Parámetros

in	*d*	diccionario a copiar

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

Definición en la línea 62 del archivo diccionarioV1.hxx.

4.2.3. Documentación de las funciones miembro

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

Chequea el Invariante de la representacion.

Devuelve

true si el invariante es correcto, falso en caso contrario

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

Definición en la línea 157 del archivo diccionarioV1.hxx.

4.2.3.2. bool diccionario::empty ( ) const

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

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

Definición en la línea 136 del archivo diccionarioV1.hxx.

4.2.3.3. const diccionario::entrada & diccionario::find (const string & *s*) const

busca una cadena en el diccionario

Parámetros

s	cadena a buscar

Devuelve

una copia de la entrada en el diccionario. Si la palabra s no se encuentra devuelve null()

Postcondición

no modifica el diccionario.

```
Uso
if (D.fin("hola")!=D.null()) cout << "Esta" ;
else cout << "No esta";
```

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

Definición en la línea 75 del archivo diccionarioV1.hxx.

4.2.3.4. const string & diccionario::max_element ( ) const

devuelve una referencia constante a la entrada con un mayor numero de ocurrencias en el diccionario.

Postcondición

No se modifica el diccionario.

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

Definición en la línea 143 del archivo diccionarioV1.hxx.

4.2.3.5. const diccionario::entrada & diccionario::null ( ) const

entrada nula del diccionario

Devuelve

Devuelve la entrada nula del diccionario.

Postcondición

no modifica el diccionario

Definición en la línea 68 del archivo diccionarioV1.hxx.

4.2.3.6. diccionario & diccionario::operator= (const diccionario & *org*)

operador de asignación

Parámetros

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

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

Definición en la línea 118 del archivo diccionarioV1.hxx.

4.2.3.7. int & diccionario::operator[] (const string & *s*)

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.

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

Definición en la línea 84 del archivo diccionarioV1.hxx.

4.2.3.8. const int & diccionario::operator[] (const string & *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.

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

Definición en la línea 103 del archivo diccionarioV1.hxx.

4.2.3.9. diccionario::size_type diccionario::size ( ) const

numero de entradas en el diccionario

Postcondición

No se modifica el diccionario.

Tareas pendientes implementa esta funcion.

Definición en la línea 129 del archivo diccionarioV1.hxx.

4.2.4. Documentación de las funciones relacionadas y clases amigas

4.2.4.1. ostream& operator<< (ostream & *sal,* const diccionario & *D*) [friend]

imprime todas las entradas del diccionario

Postcondición

No se modifica el diccionario.

Tareas pendientes implementar esta funcion

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

Definición en la línea 173 del archivo diccionarioV1.hxx.

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

- diccionario.h
- diccionarioV1.hxx
- diccionarioV2.hxx