commentrgb0.5,0.0,0.0 keywordrgb0.0,0.5,0.0 keywordtypergb0.38,0.25,0.125 keywordflowrgb0.88,0.5,0.0 preprocessorrgb0.5,0.38,0.125 stringliteralrgb0.0,0.125,0.25 charliteralrgb0.0,0.5,0.5 vhdldigitrgb1.0,0.0,1.0 vhdlkeywordrgb0.43,0.0,0.43 vhdllogicrgb1.0,0.0,0.0 vhdlcharrgb0.0,0.0,0.0

darkgray

METEORITOS-V0

Generado por Doxygen 1.8.6

Lunes, 3 de Noviembre de 2014 23:03:25

Contents

Chapter 1

Documentación Práctica

Versión

v0

Autor

Juan F. Huete

1.1 Introducción

En esta practica se pretende avanzar en el uso de las estructuras de datos, para ello comenzaremos con el diseño de dos tipos de datos, el primero será un tipo capaz de guardar la información asociada a un meteorito, mientras que el segundo implicará el diseño de un contenedor básico, capaz de almacenar un gran volumen de meteoritos.

1.2 Conjunto de Datos de Meteoritos

Este conjunto de datos contiene información sobre 45716 meteritos que han caido en la tierra, actualizado a 14 de Mayo de 2013, con los siguientes datos.

- nombre: por ejemplo "Andhara" o "Andover", de tipo string.
- tipo: secuencia de códigos, separados por comas, que describen el tipo de meteorito, por ejemplo "Iron, IIAB".
- masa: masa del meteorito en gramos.
- caido/encontrado: valor binario que indica si ha sido encontrado o sólo sabemos que ha caido (fell/found),
- año: año en que impactó
- coordena de latitud
- coordena de longitud

Cuando alguno de los campos es desconocido, este está en blanco. La base de datos está en formato csv donde los distintos campos estan separados por punto y coma, por ejempo, las primeras líneas del fichero son:

```
name;recclass;mass;fall;year;reclat;reclong
Aachen;L5;21;Fell;1880;50,775;6,08333
Aarhus;H6;720;Fell;1951;56,18333;10,23333
Abee;EH4;107000;Fell;1952;54,21667;-113
Acapulco;Acapulcoite;1914;Fell;1976;16,88333;-99,9
```

1.2.1 Meteorito

En nuestro caso, definiremos un tipo meteorito como un par, donde el campo first representa el nombre del meteorito (string) y el campo second será del tipo defMet, que define al resto de datos asociados que definen el meteorito.

```
typedef pair<nombreM,defM> meteorito;
```

La especificación del tipo, así como su implementación se realizará en los ficheros meteorito.h y meteorito.hxx

1.3 Diccionario

De forma abstracta 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, la clave será el nombre del meteorito y la defición contendrá valores de tipo defM. Como TDA diccionario, lo vamos a dotar de un conjunto restringido de métodos (inserción de elementos, consulta de un elemento por clave). Este diccionario "simulará" un diccionario de la stl, con algunas claras diferencias pues, entre otros, no estará dotado de la capacidad de iterar (recorrer) a través de sus elementos, que se hará en las siguientes prácticas.

Asociado al diccionario, tendremos los tipos tipos

```
diccionario::entrada
diccionario::size_type
```

que permiten hacer referencia al par de elementos almacenados en cada una de las posiciones del diccionario y el número de elementos del mismo, respectivamente. El primer campo de una entrada, first, representa la clave y el segundo campo, second, representa la definición.

1.4 "Se Entrega / Se Pide"

1.4.1 Se entrega

En esta práctica se entrega los fuentes necesarios para generar la documentación de este proyecto así como el código necesario para resolver este problema. En concreto los ficheros que se entregan son:

- documentacion.pdf Documentación de la práctica en pdf.
- dox_diccionario Este fichero contiene el fichero de configuración de doxigen necesario para generar la documentación del proyecto (html y pdf). Para ello, basta con ejecutar desde la línea de comando

  ```
  doxygen dox_diccionario
  ```

  La documentación en html la podemos encontrar en el fichero ./html/index.html, para generar la documentación en latex es suficiente con hacer los siguientes pasos

  ```
  cd latex
  make
  ```

  como resultado tendremos el fichero refman.pdf que incluye toda la documentación generada.

- diccionario.h Especificación del TDA diccionario.
- diccionario.hxx plantilla de fichero donde debemos implementar el diccionario.
- meteorito.h Plantilla para la especificación del TDA meteorito
- principal.cpp fichero donde se incluye el main del programa. En este caso, se toma como entrada el fichero de datos "meteorites_all.csv" y se debe cargar en el diccionario.

1.4.2 Se Pide

Se pide implementar el código asociado tanto para el TDA meteorito (en los ficheros meteoritos.h y meteorito.hxx) como el tipo de dato diccionario. En este último caso se considerarán dos posibles representaciones basadas en el tipo de dato vector de la stl, analizando la eficiencia de las mismas, teniendo en cuenta las operaciones de insercion y búsqueda. La primera implementación se entregará en un fichero denominado diccionarioV1.hxx y la segunda en un fichero denominado diccionarioV2.hxx

Para compilar con la primera implementación habrá que hacer

- g++ -D DICC_V1 -o correctorV1 principal.cpp

Para compilar con la segunda implementación se tendrá que utilizar

- g++ -D DICC_V2 -o correctorV2 principal.cpp

Por tanto, los alumnos deberán subir a la plataforma las dos implementaciones así como un análisis de la eficiencia de las mismas en los siguientes ficheros

- diccionarioV1.hxx
- diccionarioV2.hxx
- AnalisisCompartivo.pdf Dicho análisis valorará por un lado el tiempo dedicado a la insercion de toda la colección de meteoritos en un diccionario y por otro el tiempo necesario para realizar una búsqueda de todos los meteoritos en el mismo.

1.5 Fecha Límite de Entrega: Martes 28 de Octubre a las 23:59 horas.

1.6 Representaciones

El alumno deberá realizar dos implementaciones distintas del diccionario, utilizando como base el TDA vector de la STL, en la primera de ellas los elementos se almacenarán sin tener en cuenta el valor de la clave mientras que en la segunda debemos garantizar que los elementos se encuentran ordenados por dicho valor.

1.7 Primera Representación:

1.7.1 Función de Abstracción :

Función de Abstracción: AF: Rep => Abs

dado D=(vector<entrada> dic) ==> Diccionario Dic;

Un objeto abstracto, Dic, representando una colección de pares (clave,def) se instancia en la clase diccionario como un vector de entradas, definidas como diccionario::entrada. Dada una entrada, x, en D, x.first representa a una clave válida y x.second representa su definición.

1.7.2 Invariante de la Representación:

Propiedades que debe cumplir cualquier objeto

```
Dic.size() == D.dic.size();


Para todo i, 0 <= i < D.dic.size() se cumple
        D.dic[i].first != "";
        D.dic[i].fisrt != D.dic[j].first, para todo j!=i.
```

1.8 Segunda Representación:

En este caso, la representación que se utiliza es un vector ordenado de entradas, teniendo en cuenta el valor de la clave.

1.8.1 Función de Abstracción :

Función de Abstracción: AF: Rep => Abs

```
dado D=(vector<entrada>  dic) ==> Diccionario Dic;
```

Un objeto abstracto, Dic, representando una colección ORDENADA de pares (clave,def), se instancia en la clase diccionario como un vector de entradas, definidas como diccionario::entrada. Dada una entrada, x, en D, x.first representa a una clave válida y x.second representa su definición.(

1.8.2 Invariante de la Representación:

Propiedades que debe cumplir cualquier objeto

```
Dic.size() == D.dic.size();

Para todo i, 0 <= i < D.dic.size() se cumple
        D.dic[i].first != "";
Para todo i, 0 <= i < D.dic.size()-1 se cumple
        D.dic[i].first< D.dic[i+1].first
```

Chapter 2

Lista de tareas pendientes

Miembro diccionario::cheq_rep () const
implementa esta función

Miembro diccionario::diccionario ()
implementa esta función

Miembro diccionario::diccionario (const diccionario &d)
implementa esta función

Miembro diccionario::empty () const
implementa esta función

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

Miembro diccionario::insert (const entrada &e)
implementa esta función

Miembro diccionario::operator= (const diccionario &org)
implementa esta función

Miembro diccionario::operator[] (const nombreM &s) const
implementa esta función

Miembro diccionario::operator[] (const nombreM &s)
implementa esta función

Chapter 3

Índice de clases

3.1 Lista de clases

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

Chapter 4

Documentación de las clases

4.1 Referencia de la Clase defM

Métodos públicos

- **defM** (const defM &x)
- vector< string > **getCodes** () const
- bool **getFall** () const
- double **getLat** () const
- double **getLong** () const
- double **getMas** () const
- string **getYear** () const
- bool **setCode** (const string &s)
- void **setFall** (bool f)
- void **setLat** (const double &lat)
- void **setLong** (const double &longi)
- void **setMas** (const double &m)
- void **setYear** (const string &y)

Atributos privados

- bool **caidoEncontrado**
- double **latitud**
- double **longitud**
- double **masa**
- nombreM **nombreMeteo**
- vector< string > **tipo**
- string **year**

Amigas

- ostream & **operator**<< (ostream &, const defM &)

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

- meteorito.h

4.2 Referencia de la Clase diccionario

Clase diccionario.

`#include <diccionario.h>`

Tipos públicos

- typedef pair< nombreM, defM > **entrada**
- typedef unsigned int **size_type**

Métodos públicos

- diccionario ()

  constructor primitivo.
- diccionario (const diccionario &d)

  constructor de copia
- bool empty () const

  vacia Chequea si el diccionario esta vacio (size()==0)
- pair< entrada, bool > find (const nombreM &s) const

  busca una meteorito en el diccionario
- bool insert (const entrada &e)

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

  operador de asignación
- defM & operator[] (const nombreM &s)

  Consulta/Inserta una entrada en el diccionario.
- const defM & operator[] (const nombreM &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**

Amigas

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

  imprime todas las entradas del diccionario

4.2.1 Descripción detallada

Clase diccionario.

diccionario:: diccionario, find, operator[], size, 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 implementa esta función

4.2.2.2 diccionario::diccionario (const diccionario & *d*)

constructor de copia

Parámetros

>p0.10	>p0.15	p0.678
`in` *d* diccionario a copiar

Tareas pendientes implementa esta función

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

4.2.3.2 bool diccionario::empty ( ) const

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

Tareas pendientes implementa esta función

4.2.3.3 pair< diccionario::entrada, bool > & diccionario::find (const nombreM & *s*) const

busca una meteorito en el diccionario

Parámetros

>p0.15	p0.805
s	cadena a buscar

Devuelve

una copia de la entrada en el diccionario. Si no se encuentra devuelve la entrada con la definicion por defecto

Postcondición

no modifica el diccionario.

Uso

```
if (D.find("aaa").second ==true) cout << "Esta" ;
else cout << "No esta";
```

Tareas pendientes implementa esta función

4.2.3.4 bool diccionario::insert (const entrada & *e*)

Inserta una entrada en el diccionario.

Parámetros

>p0.15	p0.805
e	entrada a insertar

Devuelve

true si la entrada se ha podido insertar con éxito, esto es, no existe un meteorito con igual nombre en el diccionario. False en caso contrario

Postcondición

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

Tareas pendientes implementa esta función

4.2.3.5 diccionario & diccionario::operator= (const diccionario & *org*)

operador de asignación

Parámetros

>p0.10	>p0.15	p0.678
`in` *org*	diccionario a copiar. Crea un diccionario duplicado exacto de org.

Tareas pendientes implementa esta función

4.2.3.6 defM & diccionario::operator[] (const nombreM & *s*)

Consulta/Inserta una entrada en el diccionario.

Busca la cadena s en el diccionario, si la encuentra devuelve una referencia a la definición de la misma en caso contrario la inserta, con una definición por defecto, devolviendo una referencia a este valor.

Parámetros

>p0.10	>p0.15	p0.678
`in` *s*	cadena a insertar
`out` *defM*	& referencia a la definicion asociada a la entrada, nos permite modificar la definición

Postcondición

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

Tareas pendientes implementa esta función

4.2.3.7 const defM & diccionario::operator[] (const nombreM & *s*) const

Consulta una entrada en el diccionario.

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

Parámetros

>p0.10	>p0.15	p0.678
`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

4.2.3.8 diccionario::size_type diccionario::size ( ) const

numero de entradas en el diccionario

Postcondición

No se modifica el diccionario.

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

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

- diccionario.h
- diccionario.hxx