Practica4 : Diccionario/Guia telefonos

Generated by Doxygen 1.9.2

Chapter 1

Rep del TDA Guia_Tlf

1.1 Invariante de la representación

El invariante es *para* todo *i* y *j* tal que i<j entonces *e1i* y e1j son distintos

1.2 Función de abstracción

Un objeto válido *rep* del TDA Guia_Tlf representa al valor

{(rep.begin().first,rep.begin().second),(rep.begin()+1.first,rep.begin()+1.second),....,(rep.begin()+n-1.first,rep.↩
begin()+n-1.second)}

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 data< T, U > Struct Template Reference

Tipo elemento que define el diccionario. T es el tipo de dato asociado a una clave que no se repite (DNI p.ej.) y list es una lista de datos (string p.ej) asociados a la clave de tipo T. El diccionario está ordenado de menor a mayor clave.

`#include <diccionario.h>`

Public Attributes

- T clave
- list< U > info_asoci

4.1.1 Detailed Description

template<class T, class U>
struct data< T, U >

Tipo elemento que define el diccionario. T es el tipo de dato asociado a una clave que no se repite (DNI p.ej.) y list es una lista de datos (string p.ej) asociados a la clave de tipo T. El diccionario está ordenado de menor a mayor clave.

Definition at line 21 of file diccionario.h.

4.1.2 Member Data Documentation

4.1.2.1 clave

```
template<class T , class U >
T data< T, U >::clave
```

Definition at line 22 of file diccionario.h.

4.1.2.2 info_asoci

```
template<class T , class U >
list<U> data< T, U >::info_asoci
```

Definition at line 23 of file diccionario.h.

The documentation for this struct was generated from the following file:

- diccionario/include/diccionario.h

4.2 Diccionario< T, U > Class Template Reference

Un diccionario es una lista de datos de los definidos anteriormente. Cuidado porque se manejan listas de listas. Se añaden 2 funciones privadas que hacen más facil la implementación de algunos operadores o funciones de la parte pública. Copiar copia un diccioario en otro y borrar elimina todos los elementos de un diccionario. La implementación de copiar puede hacerse usando iteradores o directamente usando la función assign.

```
#include <diccionario.h>
```

Classes

- class iterator

  clase para iterar sobre la guia

Public Member Functions

- Diccionario ()

  Constructor por defecto.
- Diccionario (const Diccionario &D)

  Constructor de copias.
- ~Diccionario ()

  Destructor.
- Diccionario< T, U > & operator= (const Diccionario< T, U > &D)

  Operador de asignación.
- bool Esta_Clave (const T &p, typename list< data< T, U > >::iterator &it_out)

  Busca la clave p en el diccionario. Si está devuelve un iterador a dónde está clave. Si no está, devuelve end() y deja el iterador de salida apuntando al sitio dónde debería estar la clave.
- void Insertar (const T &clave, const list< U > &info)

  Inserta un nuevo registro en el diccionario. Lo hace a través de la clave e inserta la lista con toda la información asociada a esa clave. Si el diccionario no estuviera ordenado habría que usar la función sort()
- void AddSignificado_Palabra (const U &s, const T &clave)

  Añade una nueva informacion asocida a una clave que está en el diccionario. la nueva información se inserta al final de la lista de información. Si no esta la clave la inserta y añade la informacion asociada.
- bool removeSignificados_Palabra (const T &clave)

  Elimina todos los significados de una palabra a partir de la clave Funcion extra "accidental" en el proceso de remove↩ Palabra.
- bool removePalabra (const T &clave)

  Elimina la palabra y sus definiciones del diccionario.

- Diccionario operator+ (const Diccionario &d)

  Une dos diccionarios devolviendo la union en un auxiliars.
- Diccionario getElementosEntreClaves (const T &clavea, const T &claveb)

  Devuelve los elementos entre las 2 claves dadas.
- list< U > getInfo_Asoc (const T &p)

  Devuelve la información (una lista) asociada a una clave p. Podrían haberse definido operator[] como data<T,U> & operator[ ](int pos){ return datos.at(pos);} const data<T,U> & operator[ ](int pos)const { return datos.at(pos);}.
- int size () const

  Devuelve el tamaño del diccionario.
- list< data< T, U > >::iterator begin ()

  Funciones begin y end asociadas al diccionario.
- list< data< T, U > >::iterator end ()

  Devuelve el iterador fin del diccionario.
- list< data< T, U > >::const_iterator begin () const

  Devuelve el iterador inicio del diccionario.
- list< data< T, U > >::const_iterator end () const

  Devuelve el iterador fin del diccionario.

4.2.1 Detailed Description

template<class T, class U>
class Diccionario< T, U >

Un diccionario es una lista de datos de los definidos anteriormente. Cuidado porque se manejan listas de listas. Se añaden 2 funciones privadas que hacen más facil la implementación de algunos operadores o funciones de la parte pública. Copiar copia un diccioario en otro y borrar elimina todos los elementos de un diccionario. La implementación de copiar puede hacerse usando iteradores o directamente usando la función assign.

Definition at line 46 of file diccionario.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Diccionario() [1/2]

```
template<class T , class U >
Diccionario< T, U >::Diccionario ( )  [inline]
```

Constructor por defecto.

Definition at line 73 of file diccionario.h.

```
00073 :datos(list<data<T,U> >()){}
```

4.2.2.2 Diccionario() [2/2]

```
template<class T , class U >
Diccionario< T, U >::Diccionario (
            const Diccionario< T, U > & D )  [inline]
```

Constructor de copias.

Parameters

D	Diccionario a copiar

Definition at line 79 of file diccionario.h.

```
00079                                           {
00080                 Copiar(D);
00081         }
```

4.2.2.3 ~Diccionario()

```
template<class T , class U >
Diccionario< T, U >::~Diccionario ( )  [inline]
```

Destructor.

Definition at line 86 of file diccionario.h.

```
00086 {}
```

4.2.3 Member Function Documentation

4.2.3.1 AddSignificado_Palabra()

```
template<class T , class U >
void Diccionario< T, U >::AddSignificado_Palabra (
            const U & s,
            const T & clave )  [inline]
```

Añade una nueva informacion asocida a una clave que está en el diccionario. la nueva información se inserta al final de la lista de información. Si no esta la clave la inserta y añade la informacion asociada.

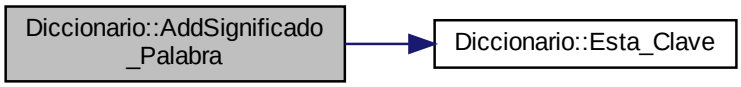
Parameters

s	La definicion a añadir
clave	La palabra a la que queremos añadir la definicion s

Definition at line 166 of file diccionario.h.

```
00166                                                                   {
00167                 typename list<data<T,U> >::iterator it;
00168
00169                 if (!Esta_Clave(clave,it)){
00170                      data<T,U> p;
00171                      p.clave = clave;
00172                      datos.insert(it,p);
00173                 }
00174
00175                  //Insertamos el siginificado al final
00176                     (*it).info_asoci.insert((*it).info_asoci.end(),s);
00177         }
```

Here is the call graph for this function:



4.2.3.2 begin() [1/2]

```
template<class T , class U >
list< data< T, U > >::iterator Diccionario< T, U >::begin ( )  [inline]
```

Funciones begin y end asociadas al diccionario.

Devuelve el iterador inicio del diccionario

Returns

un iterador que apunta al inicio del diccionario

Definition at line 343 of file diccionario.h.

```
00343                                                {
00344                 return datos.begin();
00345         }
```

4.2.3.3 begin() [2/2]

```
template<class T , class U >
list< data< T, U > >::const_iterator Diccionario< T, U >::begin ( ) const  [inline]
```

Devuelve el iterador inicio del diccionario.

Returns

un iterador no modificable que apunta al inicio del diccionario

Definition at line 358 of file diccionario.h.

```
00358                                                      {
00359                 return datos.begin();
00360         }
```

4.2.3.4 end() [1/2]

```
template<class T , class U >
list< data< T, U > >::iterator Diccionario< T, U >::end ( )  [inline]
```

Devuelve el iterador fin del diccionario.

Returns

un iterador que apunta al final del diccionario

Definition at line 350 of file diccionario.h.

```
00350                                                {
00351                 return datos.end();
00352           }
```

4.2.3.5 end() [2/2]

```
template<class T , class U >
list< data< T, U > >::const_iterator Diccionario< T, U >::end ( ) const  [inline]
```

Devuelve el iterador fin del diccionario.

Returns

un iterador no modificable que apunta al final del diccionario

Definition at line 366 of file diccionario.h.

```
00366                                                      {
00367                 return datos.end();
00368           }
```

4.2.3.6 Esta_Clave()

```
template<class T , class U >
bool Diccionario< T, U >::Esta_Clave (
                const T & p,
                typename list< data< T, U > >::iterator & it_out )  [inline]
```

Busca la clave p en el diccionario. Si está devuelve un iterador a dónde está clave. Si no está, devuelve end() y deja el iterador de salida apuntando al sitio dónde debería estar la clave.

Parameters

p	La palabra a buscar
it_out	El iterador de salida que apunta al struct 'data' donde se encuentre la palabra pasada por parametro

Returns

true si la encuentra, false si no la encuentra

Definition at line 110 of file diccionario.h.

```
00110                                                                                {
00111
00112               if (datos.size()>0){
00113
00114                   typename list<data<T,U> >::iterator it;
00115
00116                   for (it=datos.begin(); it!=datos.end() ;++it){
00117                     if ((*it).clave==p) {
00118                           it_out=it;
00119                           return true;
00120                     }
00121                     else if ((*it).clave>p){
00122                           it_out=it;
00123                           return false;
00124                     }
00125
00126                   }
00127
00128                   it_out=it;
00129                   return false;
00130               }
00131               else {
00132                     it_out=datos.end();
00133                     return false;
00134               }
00135          }
```

4.2.3.7 getElementosEntreClaves()

```
template<class T , class U >
Diccionario Diccionario< T, U >::getElementosEntreClaves (
            const T & clavea,
            const T & claveb )  [inline]
```

Devuelve los elementos entre las 2 claves dadas.

Parameters

clavea	La clave inferior
claveb	La clave superior

Returns

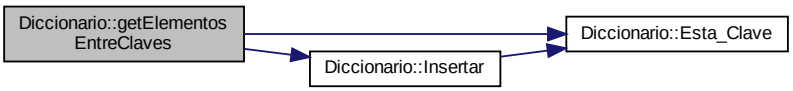
D Un nuevo diccionario con las claves intermedias con sus definiciones

Definition at line 246 of file diccionario.h.

```
00246                                                                                {
00247               Diccionario<T,U> D;
00248               typename list<data<T,U»::iterator ita, itb;
00249               bool terminar = false;
00250               int disa,disb;
00251               if (Esta_Clave(clavea, ita) && Esta_Clave(claveb, itb)){
00252                   disa = distance(datos.begin(), ita);
00253                   disb = distance(datos.begin(), itb);
00254                   if(disa <= disb){ //Si clave_a esta antes o es igual que clave_b
00255                       while (ita != itb){
00256                           D.Insertar((*ita).clave, (*ita).info_asoci);
00257                           ita++;
00258                       }
00259                       D.Insertar((*ita).clave, (*ita).info_asoci); //Añadimos la clave b
00260                   }
```

```
00261                 else{ //Si clave_b esta antes que clave_a
00262                     while (itb != ita){
00263                         D.Insertar((*itb).clave, (*itb).info_asoci);
00264                         itb++;
00265                         //cout « (*itb).clave « " y " « (*itb).clave « endl;
00266                     }
00267                     D.Insertar((*itb).clave, (*itb).info_asoci); //Añadimos la clave b
00268                 }
00269 
00270             }
00271             else
00272                 cerr « "Una de las claves introducidas no existe" « endl;
00273 
00274             return D;
00275         }
```

Here is the call graph for this function:



4.2.3.8 getInfo_Asoc()

```
template<class T , class U >
list< U > Diccionario< T, U >::getInfo_Asoc (
             const T & p )  [inline]
```

Devuelve la información (una lista) asociada a una clave p. Podrían haberse definido operator[] como data<T,U> & operator[ ](int pos){ return datos.at(pos);} const data<T,U> & operator[ ](int pos)const { return datos.at(pos);}.

Parameters

p	La palabra de la cual se busca su lista de definiciones

Returns

La lista de definiciones de la palabra si existe, devuelve una vacia en otro caso

Definition at line 285 of file diccionario.h.

```
00285                                                 {
00286                 typename list<data<T,U> >::iterator it;
00287 
00288                 if (!Esta_Clave(p,it)){
00289                  return list<U>();
00290                 }
00291                 else{
00292                  return (*it).info_asoci;
00293                 }
00294         }
```

Here is the call graph for this function:

4.2.3.9 Insertar()

```
template<class T , class U >
void Diccionario< T, U >::Insertar (
            const T & clave,
            const list< U > & info )  [inline]
```

Inserta un nuevo registro en el diccionario. Lo hace a través de la clave e inserta la lista con toda la información asociada a esa clave. Si el diccionario no estuviera ordenado habría que usar la función sort()

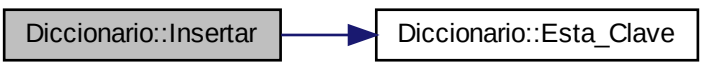
Parameters

clave	La palabra a insertar
info	La lista de definiciones a insertar

Definition at line 144 of file diccionario.h.

```
00144                                                                {
00145
00146               typename list<data<T,U> >::iterator it;
00147
00148               if (!Esta_Clave(clave,it)){
00149                     data<T,U> p;
00150                     p.clave = clave;
00151                     p.info_asoci=info;
00152
00153                     datos.insert(it,p);
00154
00155               }
00156
00157          }
```

Here is the call graph for this function:



4.2.3.10 operator+()

```
template<class T , class U >
Diccionario Diccionario< T, U >::operator+ (
            const Diccionario< T, U > & d )  [inline]
```

Une dos diccionarios devolviendo la union en un auxiliars.

Parameters

d	Diccionario a sumar, añadiendo palabras y definiciones nuevas

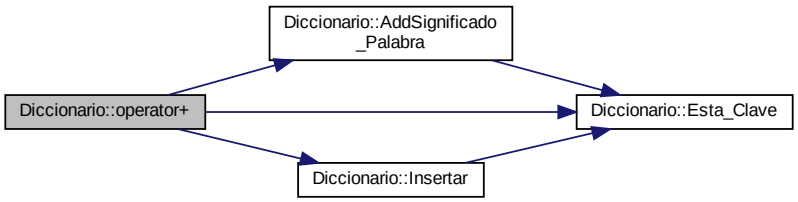
Returns

aux Un diccionario que contiene ambos diccionarios

Definition at line 217 of file diccionario.h.

```
00217                                                   {
00218             Diccionario aux(*this);
00219             typename list<data<T,U»::iterator it2;
00220             typename list<data<T,U»::const_iterator it;;
00221             for (it=d.datos.begin();it!=d.datos.end();++it){
00222                 if(aux.Esta_Clave((*it).clave,it2)){ //mete las definiciones que no existan de la
      palabra
00223                     typename list<U>::const_iterator itinfod, itinfoaux;
00224                     bool existedef = false;
00225                     for (itinfod=(*it).info_asoci.begin();itinfod !=
      (*it).info_asoci.end();++itinfod){
00226                         existedef = false;
00227                         for(itinfoaux=(*it2).info_asoci.begin();itinfoaux != (*it2).info_asoci.end()
      && !existedef;++itinfoaux)
00228                             if((*itinfoaux) == (*itinfod))
00229                                 existedef = true;
00230                         if(!existedef)
00231                             aux.AddSignificado_Palabra((*itinfod),(*it).clave);
00232                     }
00233                 }
00234                 else //Si no existe la palabra, la mete
00235                     aux.Insertar((*it).clave,(*it).info_asoci);
00236             }
00237             return aux;
00238         }
```

Here is the call graph for this function:



4.2.3.11 operator=()

```
template<class T , class U >
Diccionario< T, U > & Diccionario< T, U >::operator= (
            const Diccionario< T, U > & D )  [inline]
```

Operador de asignación.

Parameters

D	Diccionario a ser asignado

Returns

this el propio objeto

Definition at line 93 of file diccionario.h.

```
00093                                                                    {
00094                 if (this!=&D){
00095                       Borrar();
00096                       Copiar(D);
00097                 }
00098                 return *this;
00099           }
```

4.2.3.12 removePalabra()

```
template<class T , class U >
bool Diccionario< T, U >::removePalabra (
             const T & clave )  [inline]
```

Elimina la palabra y sus definiciones del diccionario.

Parameters

clave	La palabra a eliminar

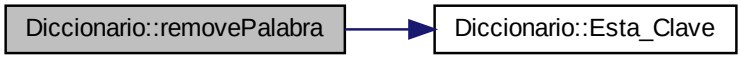
Returns

true si borra la palabra, false si no existe

Definition at line 202 of file diccionario.h.

```
00202                                                {
00203             bool borrada = false;
00204             typename list<data<T,U> >::iterator it;
00205             if(Esta_Clave(clave,it)){
00206                 datos.erase(it);
00207                 borrada = true;
00208             }
00209             return borrada;
00210         }
```

Here is the call graph for this function:



4.2.3.13 removeSignificados_Palabra()

```
template<class T , class U >
bool Diccionario< T, U >::removeSignificados_Palabra (
            const T & clave )  [inline]
```

Elimina todos los significados de una palabra a partir de la clave Funcion extra "accidental" en el proceso de removePalabra.

Parameters

clave	La palabra a la que borrar todas sus definiciones, sin borrar la palabra

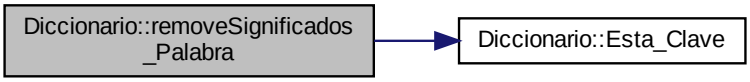
Returns

true si borra las definiciones, false en otro caso

Definition at line 185 of file diccionario.h.

```
00185                                                       {
00186             bool borrados = false;
00187             typename list<data<T,U> >::iterator it;
00188             cout « "Va a borrar las definiciones de " « clave « ":" « endl;
00189                 if (Esta_Clave(clave,it)){
00190                     (*it).info_asoci.clear();
00191                     borrados = true;
00192                 }
00193
00194             return borrados;
00195         }
```

Here is the call graph for this function:



4.2.3.14 size()

```
template<class T , class U >
int Diccionario< T, U >::size ( ) const  [inline]
```

Devuelve el tamaño del diccionario.

Returns

El tamaño del diccionario

Definition at line 302 of file diccionario.h.

```
00302                       {
00303                 return datos.size();
00304           }
```

The documentation for this class was generated from the following file:

- diccionario/include/diccionario.h

4.3 Guia_Tlf Class Reference

T.D.A. Guia_Tlf.

```
#include <guiatlf.h>
```

Classes

- class iterator

  clase para iterar sobre la guia

Public Member Functions

- string & operator[ ] (const string &nombre)

  Acceso a un elemento.
- string gettelefono (const string &nombre)
- pair< map< string, string >::iterator, bool > insert (string nombre, string tlf)

  Insert un nuevo telefono.
- pair< map< string, string >::iterator, bool > insert (pair< string, string > p)

  Insert un nuevo telefono.
- void borrar (const string &nombre)

  Borrar un telefono.
- void borrar (const string &nombre, const string &tlf)

  Borrar un telefono.
- int size () const

  Numero de telefonos.
- unsigned int contabiliza (const string &nombre)

  Contabiliza cuantos telefonos tenemos asociados a un nombre.
- void clear ()

  Limpia la guia.
- Guia_Tlf operator+ (const Guia_Tlf &g)

  Union de guias de telefonos.
- Guia_Tlf operator- (const Guia_Tlf &g)

  Diferencia de guias de telefonos.
- Guia_Tlf previos (const string &nombre, const string &tlf)

  Obtiene una guia con los nombre previos a uno dado.
- void modificarTlf (string nombre, string tlf)

  Modifica el numero de telefono del nombre pasado como parametro.
- Guia_Tlf TlfsPorLetra (char letra)

  Lista los telefonos de los contactos que empiecen por la letra pasada como paramatro.
- Guia_Tlf TelfsEntreNombres (const string &nombre_1, const string &nombre_2)

  Lista los telefonos que se encuentran entre los 2 nombres dados como parametro.
- iterator begin ()

  Inicializa un iterator al comienzo de la guia.
- iterator end ()

  Inicializa un iterator al final de la guia.

Friends

- ostream & operator<< (ostream &os, Guia_Tlf &g)

  Escritura de la guia de telefonos.
- istream & operator>> (istream &is, Guia_Tlf &g)

  Lectura de la guia de telefonos.

4.3.1 Detailed Description

T.D.A. Guia_Tlf.

Una instancia *c* del tipo de datos abstracto `Guia_Tlf` es un objeto formado por una colección de pares {(e11,e21),(e12,e22),(e13,e23),...,(e1n-1,e2n-1)} ordenados por la el primer elemento del par denominado clave o key. No existen elementos repetidos.

Un ejemplo de su uso:

Author

Rosa Rodríguez

Date

Marzo 2012

Definition at line 43 of file guiatlf.h.

4.3.2 Member Function Documentation

4.3.2.1 begin()

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

Inicializa un iterator al comienzo de la guia.

Definition at line 339 of file guiatlf.h.

```
00339                           {
00340             iterator i;
00341             i.it=datos.begin();
00342             return i;
00343             }
```

4.3.2.2 borrar() [1/2]

```
void Guia_Tlf::borrar (
             const string & nombre )  [inline]
```

Borrar un telefono.

Parameters

nombre	nombre que se quiere borrar

Note

: en caso de que fuese un multimap borraria todos con ese nombre

Definition at line 120 of file guiatlf.h.

```
00120                                                {
00121                 map<string,string>::iterator itlow = datos.lower_bound(nombre);//el primero que tiene
      dicho nombre
00122                 map<string,string>::iterator itupper = datos.upper_bound(nombre);//el primero que ya no
      tiene dicho nombre
00123                 datos.erase(itlow,itupper);//borramos todos aquellos que tiene dicho nombre
00124                 //OTRA ALTERNATIVA
00125                 //pair<map<string,string>::iterator,map<string,string>::iterator>ret;
00126                 //ret = datos.equal_range(nombre
00127                 //datos.erase(ret.first,ret.second);
00128             }
```

4.3.2.3 borrar() [2/2]

```
void Guia_Tlf::borrar (
            const string & nombre,
            const string & tlf )  [inline]
```

Borrar un telefono.

Parameters

nombre	nombre que se quiere borrar y telefono asociado

Note

: esta funcion nos permite borrar solamente aquel que coincida en nombre y tlf

Definition at line 136 of file guiatlf.h.

```
00136                                                          {
00137                     map<string,string>::iterator itlow = datos.lower_bound(nombre);//el primero que
      tiene dicho nombre
00138                 map<string,string>::iterator itupper = datos.upper_bound(nombre);//el primero que ya no
      tiene dicho nombre
00139                 map<string,string>::iterator it;
00140                 bool salir =false;
00141                 for (it=itlow; it!=itupper && !salir;++it){
00142                       if (it->second==tlf){
00143                      datos.erase(it);
00144                      salir =true;
00145                       }
00146                 }
00147
00148             }
```

4.3.2.4 clear()

```
void Guia_Tlf::clear ( )  [inline]
```

Limpia la guia.

Definition at line 170 of file guiatlf.h.

```
00170                       {
00171                   datos.clear();
00172             }
```

4.3.2.5 contabiliza()

```
unsigned int Guia_Tlf::contabiliza (
             const string & nombre )  [inline]
```

Contabiliza cuantos telefonos tenemos asociados a un nombre.

Parameters

nombre	nombre sobre el que queremos consultar

Returns

numero de telefonos asociados a un nombre

Definition at line 163 of file guiatlf.h.

```
00163                                                   {
00164                   return datos.count(nombre);
00165             }
```

4.3.2.6 end()

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

Inicializa un iterator al final de la guia.

Definition at line 347 of file guiatlf.h.

```
00347                           {
00348             iterator i;
00349             i.it=datos.end();
00350             return i;
00351             }
```

4.3.2.7 gettelefono()

```
string Guia_Tlf::gettelefono (
            const string & nombre )  [inline]
```

Definition at line 76 of file guiatlf.h.

```
00076                                                   {
00077                 map<string,string>::iterator it=datos.find(nombre);
00078                 if (it==datos.end())
00079                   return string("");
00080                 else return it->second;
00081           }
```

4.3.2.8 insert() [1/2]

```
pair< map< string, string >::iterator, bool > Guia_Tlf::insert (
            pair< string, string > p )  [inline]
```

Insert un nuevo telefono.

Parameters

p	pair con el nombre y el telefono asociado

Returns

: un pair donde first apunta al nuevo elemento insertado y bool es true si se ha insertado el nuevo tlf o o false en caso contrario

Definition at line 106 of file guiatlf.h.

```
00106                                                                   {
00107
00108                 pair<map<string,string> ::iterator,bool> ret;
00109
00110                 ret=datos.insert(p); //datos.insert(datos.begin(),p); tambien funcionaría
00111                 return ret;
00112
00113           }
```

4.3.2.9 insert() [2/2]

```
pair< map< string, string >::iterator, bool > Guia_Tlf::insert (
            string nombre,
            string tlf )  [inline]
```

Insert un nuevo telefono.

Parameters

nombre	nombre clave del nuevo telefono
tlf	numero de telefono

Returns

: un pair donde first apunta al nuevo elemento insertado y bool es true si se ha insertado el nuevo tlf o o false en caso contrario

Definition at line 90 of file guiatlf.h.

```
00090                                                                                {
00091                   pair<string,string> p (nombre,tlf);
00092                   pair< map<string,string> ::iterator,bool> ret;
00093
00094                   ret=datos.insert(p); //datos.insert(datos.begin(),p); tambien funcionaría
00095                   return ret;
00096
00097             }
```

4.3.2.10 modificarTlf()

```
void Guia_Tlf::modificarTlf (
              string nombre,
              string tlf )  [inline]
```

Modifica el numero de telefono del nombre pasado como parametro.

Parameters

nombre	El nombre de contacto cuyo telefono se quiere modificar
tlf	El nuevo telefono que reemplazara el antiguo

Definition at line 227 of file guiatlf.h.

```
00227                                                    {
00228                   map<string,string>::iterator it=datos.find(nombre);
00229                   if(it != datos.end()){
00230                      pair<string,string> p(it->first,tlf);
00231                      datos.erase(it);
00232                      datos.insert(p);
00233                   }
00234             }
```

4.3.2.11 operator+()

```
Guia_Tlf Guia_Tlf::operator+ (
              const Guia_Tlf & g )  [inline]
```

Union de guias de telefonos.

Parameters

g	guia que se une

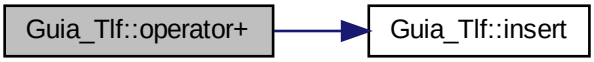
Returns

: una nueva guia resultado de unir el objeto al que apunta this y g

Definition at line 178 of file guiatlf.h.

```
00178                                                {
00179             Guia_Tlf aux(*this);
00180             map<string,string>::const_iterator it;
00181             for (it=g.datos.begin();it!=g.datos.end();++it){
00182                aux.insert(it->first,it->second);
00183             }
00184             return aux;
00185
00186             }
```

Here is the call graph for this function:



4.3.2.12 operator-()

```
Guia_Tlf Guia_Tlf::operator- (
             const Guia_Tlf & g )  [inline]
```

Diferencia de guias de telefonos.

Parameters

g	guia a restar

Returns

: una nueva guia resultado de la diferencia del objeto al que apunta this y g

Definition at line 193 of file guiatlf.h.

```
00193                                                {
00194             Guia_Tlf aux(*this);
00195             map<string,string>::const_iterator it;
00196             for (it=g.datos.begin();it!=g.datos.end();++it){
00197                aux.borrar(it->first,it->second);
00198             }
00199             return aux;
00200
00201             }
```

Here is the call graph for this function:

4.3.2.13 operator[]()

```
string & Guia_Tlf::operator[] (
            const string & nombre )  [inline]
```

Acceso a un elemento.

Parameters

nombre	nombre del elemento elemento acceder

Returns

devuelve el valor asociado a un nombre, es decir el teléfono

Definition at line 71 of file guiatlf.h.

```
00071                                                   {
00072                   return datos[nombre];
00073             }
```

4.3.2.14 previos()

```
Guia_Tlf Guia_Tlf::previos (
            const string & nombre,
            const string & tlf )  [inline]
```

Obtiene una guia con los nombre previos a uno dado.

Parameters

nombre	nombre delimitador
tlf	telefono asociado a nombre

Returns

nueva guia sin nombres mayores que *nombre*

Definition at line 208 of file guiatlf.h.

```
00208                                                                    {
00209             map<string,string>::value_compare vc=datos.value_comp();
00210             //map<string,string>::key_compare vc=datos.key_comp();
00211             Guia_Tlf aux;
00212             pair<string,string>p(nombre,tlf);
00213             map<string,string>::iterator it=datos.begin();
00214             while (vc(*it,p)){
00215                   aux.insert(*it++);
00216
00217             }
00218             return aux;
00219
00220              }
```

Here is the call graph for this function:

4.3.2.15 size()

```
int Guia_Tlf::size ( ) const  [inline]
```

Numero de telefonos.

Returns

el numero de telefonos asociados

Definition at line 153 of file guiatlf.h.

```
00153                          {
00154                   return datos.size();
00155             }
```

4.3.2.16 TelfsEntreNombres()

```
Guia_Tlf Guia_Tlf::TelfsEntreNombres (
             const string & nombre_1,
             const string & nombre_2 )  [inline]
```

Lista los telefonos que se encuentran entre los 2 nombres dados como parametro.

Parameters

nombre_1	El nombre a partir empieza a listar
nombre_2	El nombre hasta donde se listan los telefonos

Returns

tlfs Los telefonos encontrados

Definition at line 258 of file guiatlf.h.

```
00258                                                                                {
00259                 Guia_Tlf tlfs;
00260                 map<string,string>::iterator itlow = datos.find(nombre_1);
00261                 map<string,string>::iterator itupper = datos.find(nombre_2);
00262
00263                 int contador = 1,
00264                     dis1 = distance(datos.begin(),itlow),
00265                     dis2 = distance(datos.begin(), itupper);
00266                 if(itlow != datos.end() && dis1 <= dis2){
00267                   tlfs.insert(to_string(contador++) + ".",itlow->second);
00268                   while(dis1 < dis2){
00269                      itlow++; dis1++;
00270                      tlfs.insert(to_string(contador++) + ".",itlow->second);
00271                   }
00272                 }
00273                 return tlfs;
00274             }
```

Here is the call graph for this function:

4.3.2.17 TlfsPorLetra()

```
Guia_Tlf Guia_Tlf::TlfsPorLetra (
             char letra )  [inline]
```

Lista los telefonos de los contactos que empiecen por la letra pasada como paramatro.

Parameters

letra	La letra a partir del cual se buscaran los telefonos

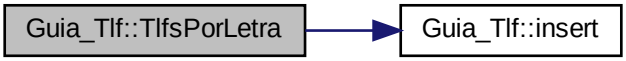
Returns

tlfs Una guia con los telefonos encontrados

Definition at line 241 of file guiatlf.h.

```
00241                                           {
00242                 Guia_Tlf tlfs;
00243                 int contador = 1; //Para enumerar los telefonos del map
00244                 map<string,string>::iterator it;
00245                 for(it = datos.begin(); it != datos.end(); ++it)
00246                     if(letra == (*it).first[0])
00247                         tlfs.insert(to_string(contador++) + ".",(*it).second);
00248
00249                 return tlfs;
00250             }
```

Here is the call graph for this function:



4.3.3 Friends And Related Function Documentation

4.3.3.1 operator<<

```
ostream & operator<< (
            ostream & os,
            Guia_Tlf & g )  [friend]
```

Escritura de la guia de telefonos.

Parameters

os	flujo de salida. Es MODIFICADO
g	guia de telefonos que se escribe

Returns

el flujo de salida

Definition at line 282 of file guiatlf.h.

```
00282                                                           {
00283                   map<string,string>::iterator it;
00284                   for (it=g.datos.begin(); it!=g.datos.end();++it){
00285                     os<<it->first<<"\t"<<it->second<<endl;
00286                   }
00287                   return os;
00288             }
```

4.3.3.2 operator>>

```
istream & operator>> (
            istream & is,
            Guia_Tlf & g )  [friend]
```

Lectura de la guia de telefonos.

Parameters

is	flujo de entrada. ES MODIFICADO
g	guia de telefonos. ES MODIFICADO

Returns

el flujo de entrada

Definition at line 297 of file guiatlf.h.

```
00297                                                                    {
00298                   pair<string,string> p;
00299                   Guia_Tlf aux;
00300
00301                   while (is»p){
00302                   aux.insert(p);
00303                   }
00304                   g=aux;
00305                   return is;
00306           }
```

The documentation for this class was generated from the following file:

- guiatlfs/include/guiatlf.h

4.4 Diccionario< T, U >::iterator Class Reference

clase para iterar sobre la guia

```
#include <diccionario.h>
```

Public Member Functions

- iterator & operator++ ()
- iterator & operator-- ()
- list< const T, list< U > > & operator* ()
- bool operator== (const iterator &i)
- bool operator!= (const iterator &i)

Friends

- class Diccionario

4.4.1 Detailed Description

template<class T, class U>
class Diccionario< T, U >::iterator

clase para iterar sobre la guia

Definition at line 310 of file diccionario.h.

4.4.2 Member Function Documentation

4.4.2.1 operator"!=()

```
template<class T , class U >
bool Diccionario< T, U >::iterator::operator!= (
            const iterator & i )  [inline]
```

Definition at line 329 of file diccionario.h.

```
00329                                                {
00330                return i.it!=it;
00331                }
```

4.4.2.2 operator*()

```
template<class T , class U >
list< const T, list< U > > & Diccionario< T, U >::iterator::operator* ( )  [inline]
```

Definition at line 322 of file diccionario.h.

```
00322                                                {
00323                return *it;
00324                }
```

4.4.2.3 operator++()

```
template<class T , class U >
iterator & Diccionario< T, U >::iterator::operator++ ( )  [inline]
```

Definition at line 314 of file diccionario.h.

```
00314                                        {
00315                 ++it;
00316                 return *this;
00317                }
```

4.4.2.4 operator--()

```
template<class T , class U >
iterator & Diccionario< T, U >::iterator::operator-- ( )  [inline]
```

Definition at line 318 of file diccionario.h.

```
00318                                   {
00319                 --it;
00320                 return *this;
00321                 }
```

4.4.2.5 operator==()

```
template<class T , class U >
bool Diccionario< T, U >::iterator::operator== (
             const iterator & i )  [inline]
```

Definition at line 325 of file diccionario.h.

```
00325                                                {
00326                 return i.it==it;
00327                 }
```

4.4.3 Friends And Related Function Documentation

4.4.3.1 Diccionario

```
template<class T , class U >
friend class Diccionario  [friend]
```

Definition at line 332 of file diccionario.h.

The documentation for this class was generated from the following file:

- diccionario/include/diccionario.h

4.5 Guia_Tlf::iterator Class Reference

clase para iterar sobre la guia

`#include <guiatlf.h>`

Public Member Functions

- iterator & operator++ ()
- iterator & operator-- ()
- pair< const string, string > & operator* ()
- bool operator== (const iterator &i)
- bool operator!= (const iterator &i)

Friends

- class Guia_Tlf

4.5.1 Detailed Description

clase para iterar sobre la guia

Definition at line 311 of file guiatlf.h.

4.5.2 Member Function Documentation

4.5.2.1 operator"!=()

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

Definition at line 330 of file guiatlf.h.

```
00330                                                {
00331               return i.it!=it;
00332                }
```

4.5.2.2 operator*()

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

Definition at line 323 of file guiatlf.h.

```
00323                                                 {
00324               return *it;
00325                }
```

4.5.2.3 operator++()

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

Definition at line 315 of file guiatlf.h.

```
00315                                       {
00316                 ++it;
00317                 return *this;
00318                }
```

4.5.2.4 operator--()

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

Definition at line 319 of file guiatlf.h.

```
00319                                       {
00320                 --it;
00321                 return *this;
00322                 }
```

4.5.2.5 operator==()

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

Definition at line 326 of file guiatlf.h.

```
00326                                                  {
00327                 return i.it==it;
00328                 }
```

4.5.3 Friends And Related Function Documentation

4.5.3.1 Guia_Tlf

```
friend class Guia_Tlf  [friend]
```

Definition at line 333 of file guiatlf.h.

The documentation for this class was generated from the following file:

- guiatlfs/include/guiatlf.h

Chapter 5

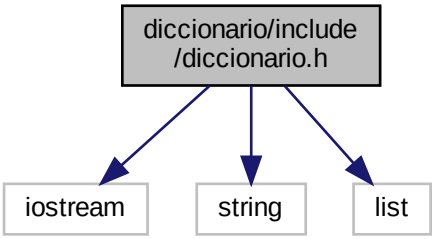
File Documentation

5.1 diccionario/include/diccionario.h File Reference

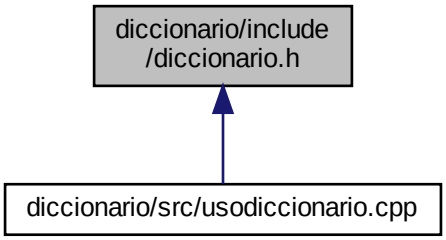
TDA diccionario. Es un .h pero funciona como .cpp.

```
#include <iostream>
#include <string>
#include <list>
```

Include dependency graph for diccionario.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct data< T, U >

  Tipo elemento que define el diccionario. T es el tipo de dato asociado a una clave que no se repite (DNI p.ej.) y list es una lista de datos (string p.ej) asociados a la clave de tipo T. El diccionario está ordenado de menor a mayor clave.
- class Diccionario< T, U >

  Un diccionario es una lista de datos de los definidos anteriormente. Cuidado porque se manejan listas de listas. Se añaden 2 funciones privadas que hacen más facil la implementación de algunos operadores o funciones de la parte pública. Copiar copia un diccioario en otro y borrar elimina todos los elementos de un diccionario. La implementación de copiar puede hacerse usando iteradores o directamente usando la función assign.
- class Diccionario< T, U >::iterator

  clase para iterar sobre la guia

Functions

- template<class T , class U >
  bool operator< (const data< T, U > &d1, const data< T, U > &d2)

  Comparador de datos. Ordena 2 registros de acuerdo a la clave de tipo T. Puede usarse como un funtor.

5.1.1 Detailed Description

TDA diccionario. Es un .h pero funciona como .cpp.

Author

Yeray Lopez Ramirez

Date

15 de diciembre de 2021

Definition in file diccionario.h.

5.1.2 Function Documentation

5.1.2.1 operator<()

```
template<class T , class U >
bool operator< (
            const data< T, U > & d1,
            const data< T, U > & d2 )
```

Comparador de datos. Ordena 2 registros de acuerdo a la clave de tipo T. Puede usarse como un funtor.

Definition at line 32 of file diccionario.h.

```
00032                                                   {
00033       if (d1.clave<d2.clave)
00034             return true;
00035       return false;
00036 }
```

5.2 diccionario.h

Go to the documentation of this file.

```
00001
00007 #ifndef _DICCIONARIO_H
00008 #define _DICCIONARIO_H
00009
00010 #include <iostream>
00011 #include <string>
00012 #include <list>
00013 using namespace std;
00014
00020 template <class T,class U>
00021 struct data{
00022     T clave;
00023     list<U> info_asoci;
00024  };
00025
00026
00031 template <class T, class U>
00032 bool operator< (const data<T,U> &d1,const data <T,U>&d2){
00033        if (d1.clave<d2.clave)
00034              return true;
00035        return false;
00036 }
00037
00045 template <class T,class U>
00046 class Diccionario{
00047       private:
00048
00049
00050             list<data<T,U> > datos;
00051
00052             void Copiar(const Diccionario<T,U>& D){
00053                /*typename list<data<T,U> >::const_iterator it_d;
00054                typename list<data<T,U> >::iterator it=this->datos.begin();*/
00055
00056                datos.assign(D.datos.begin(),D.datos.end());
00057                /*for (it_d=D.datos.begin(); it_d!=D.datos.end();++it_d,++it){
00058                      this->datos.insert(it,*it_d);
00059
00060                }*/
00061             }
00062
00063             void Borrar(){
00064
00065                 this->datos.erase(datos.begin(),datos.end());
00066             }
00067
00068
00069       public:
00073          Diccionario():datos(list<data<T,U> >()){}
00074
00079          Diccionario(const Diccionario &D){
00080                Copiar(D);
00081          }
00082
00086          ~Diccionario(){}
00087
00093          Diccionario<T,U> & operator=(const Diccionario<T,U> &D){
00094                if (this!=&D){
00095                      Borrar();
00096                      Copiar(D);
00097                }
00098                return *this;
00099          }
00100
00110         bool Esta_Clave(const T &p, typename  list<data<T,U> >::iterator &it_out){
00111
00112               if (datos.size()>0){
00113
00114                   typename list<data<T,U> >::iterator it;
00115
00116                   for (it=datos.begin(); it!=datos.end() ;++it){
00117                     if ((*it).clave==p) {
00118                           it_out=it;
00119                           return true;
00120                     }
00121                     else if ((*it).clave>p){
00122                           it_out=it;
00123                           return false;
00124                     }
00125
00126                   }
00127
```

```
00128                   it_out=it;
00129                   return false;
00130               }
00131               else {
00132                     it_out=datos.end();
00133                     return false;
00134               }
00135          }
00136
00144          void Insertar(const T& clave,const list<U> &info){
00145
00146              typename list<data<T,U> >::iterator it;
00147
00148              if (!Esta_Clave(clave,it)){
00149                    data<T,U> p;
00150                    p.clave = clave;
00151                    p.info_asoci=info;
00152
00153                    datos.insert(it,p);
00154
00155              }
00156
00157          }
00158
00166          void AddSignificado_Palabra(const U & s ,const T &clave){
00167                typename list<data<T,U> >::iterator it;
00168
00169                if (!Esta_Clave(clave,it)){
00170                     data<T,U> p;
00171                     p.clave = clave;
00172                     datos.insert(it,p);
00173                }
00174
00175                 //Insertamos el siginificado al final
00176                    (*it).info_asoci.insert((*it).info_asoci.end(),s);
00177          }
00178
00185         bool removeSignificados_Palabra(const T & clave){
00186             bool borrados = false;
00187             typename list<data<T,U> >::iterator it;
00188             cout « "Va a borrar las definiciones de " « clave « ":" « endl;
00189                 if (Esta_Clave(clave,it)){
00190                     (*it).info_asoci.clear();
00191                     borrados = true;
00192                 }
00193
00194             return borrados;
00195         }
00196
00202         bool removePalabra(const T & clave){
00203             bool borrada = false;
00204             typename list<data<T,U> >::iterator it;
00205             if(Esta_Clave(clave,it)){
00206                 datos.erase(it);
00207                 borrada = true;
00208             }
00209             return borrada;
00210         }
00211
00217         Diccionario operator+(const Diccionario & d){
00218             Diccionario aux(*this);
00219             typename list<data<T,U»::iterator it2;
00220             typename list<data<T,U»::const_iterator it;;
00221             for (it=d.datos.begin();it!=d.datos.end();++it){
00222                 if(aux.Esta_Clave((*it).clave,it2)){ //mete las definiciones que no existan de la
      palabra
00223                     typename list<U>::const_iterator itinfod, itinfoaux;
00224                     bool existedef = false;
00225                     for (itinfod=(*it).info_asoci.begin();itinfod !=
      (*it).info_asoci.end();++itinfod){
00226                         existedef = false;
00227                         for(itinfoaux=(*it2).info_asoci.begin();itinfoaux != (*it2).info_asoci.end()
      && !existedef;++itinfoaux)
00228                             if((*itinfoaux) == (*itinfod))
00229                                 existedef = true;
00230                         if(!existedef)
00231                             aux.AddSignificado_Palabra((*itinfod),(*it).clave);
00232                     }
00233                 }
00234                 else //Si no existe la palabra, la mete
00235                     aux.Insertar((*it).clave,(*it).info_asoci);
00236             }
00237             return aux;
00238         }
00239
00246         Diccionario getElementosEntreClaves(const T & clavea, const T & claveb){
00247             Diccionario<T,U> D;
```

```
00248             typename list<data<T,U»::iterator ita, itb;
00249             bool terminar = false;
00250             int disa,disb;
00251             if (Esta_Clave(clavea, ita) && Esta_Clave(claveb, itb)){
00252                 disa = distance(datos.begin(), ita);
00253                 disb = distance(datos.begin(), itb);
00254                 if(disa <= disb){ //Si clave_a esta antes o es igual que clave_b
00255                     while (ita != itb){
00256                         D.Insertar((*ita).clave, (*ita).info_asoci);
00257                         ita++;
00258                     }
00259                     D.Insertar((*ita).clave, (*ita).info_asoci); //Añadimos la clave b
00260                 }
00261                 else{ //Si clave_b esta antes que clave_a
00262                     while (itb != ita){
00263                         D.Insertar((*itb).clave, (*itb).info_asoci);
00264                         itb++;
00265                         //cout « (*itb).clave « " y " « (*itb).clave « endl;
00266                     }
00267                     D.Insertar((*itb).clave, (*itb).info_asoci); //Añadimos la clave b
00268                 }
00269
00270             }
00271             else
00272                 cerr « "Una de las claves introducidas no existe" « endl;
00273
00274             return D;
00275         }
00276
00285           list<U> getInfo_Asoc(const T & p) {
00286                typename list<data<T,U> >::iterator it;
00287
00288                if (!Esta_Clave(p,it)){
00289                 return list<U>();
00290                }
00291                else{
00292                 return (*it).info_asoci;
00293                }
00294          }
00295
00296
00297
00302          int size()const{
00303                return datos.size();
00304          }
00305
00306
00310             class iterator{
00311              private:
00312               typename list<T,list<U»::iterator it;
00313              public:
00314                iterator & operator++(){
00315                 ++it;
00316                 return *this;
00317                }
00318                iterator & operator--(){
00319                --it;
00320                return *this;
00321                }
00322                list<const T,list<U» &operator *(){
00323                return *it;
00324                }
00325                bool operator ==(const iterator &i){
00326                return i.it==it;
00327                }
00328
00329                bool operator !=(const iterator &i){
00330                return i.it!=it;
00331                }
00332                friend class Diccionario;
00333             };
00334
00343          typename list<data<T,U> >::iterator  begin(){
00344                return datos.begin();
00345          }
00350          typename list<data<T,U> >::iterator end(){
00351                return datos.end();
00352          }
00353
00358          typename list<data<T,U> >::const_iterator  begin()const{
00359                return datos.begin();
00360          }
00361
00366          typename list<data<T,U> >::const_iterator end()const {
00367                return datos.end();
00368          }
00369
```

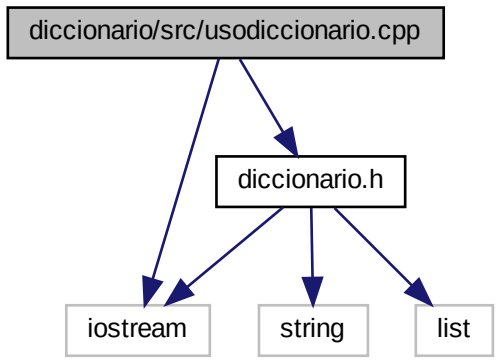
```
00370 };
00371
00372 #endif
00373
00374
00375
00376
```

5.3 diccionario/src/usodiccionario.cpp File Reference

Archivo de ejemplo para probar el TDA diccionario. Se ejecuta ./usodiccionario < archivodeentrada o escribiendo los datos a mano(mas engorroso esta ultima)

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

Include dependency graph for usodiccionario.cpp:



Functions

- ostream & operator<< (ostream &os, const Diccionario< string, string > &D)

  Operator<<. Obsérvese el uso de 2 tipos diferentes de iteradores. Uno sobre listas de listas y otro sobre listas.
- istream & operator>> (istream &is, Diccionario< string, string > &D)

  Operator >>. El formato de la entrada es: numero de claves en la primera linea clave-iésima retorno de carro numero de informaciones asociadas en la siguiente linea y en cada linea obviamente la informacion asociada.
- void EscribeSigni (const list< string > &l)

  Recorre la lista de información asociada a una clave y la imprime.
- void separador ()

  separa las salidas de texto para una mejor lectura
- int main ()

  Lee un diccioario e imprime datos asociados a una clave. Hay un fichero ejemplo de prueba: data.txt.Para lanzar el programa con ese fichero se escribe ./usodiccionario < data.txt.

5.3.1 Detailed Description

Archivo de ejemplo para probar el TDA diccionario. Se ejecuta ./usodiccionario < archivodeentrada o escribiendo los datos a mano(mas engorroso esta ultima)

Author

Yeray Lopez Ramirez

Date

Diciembre de 2021

Definition in file usodiccionario.cpp.

5.3.2 Function Documentation

5.3.2.1 EscribeSigni()

```
void EscribeSigni (
            const list< string > & l )
```

Recorre la lista de información asociada a una clave y la imprime.

Parameters

l	La lista de definiciones

Definition at line 80 of file usodiccionario.cpp.

```
00080                                       {
00081       list<string>::const_iterator it_s;
00082       for (it_s=l.begin();it_s!=l.end();++it_s){
00083             cout<<*it_s<<endl;
00084       }
00085 }
```

5.3.2.2 main()

```
int main ( )
```

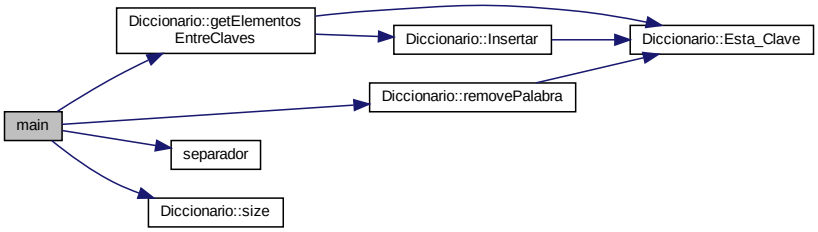
Lee un diccioario e imprime datos asociados a una clave. Hay un fichero ejemplo de prueba: data.txt.Para lanzar el programa con ese fichero se escribe ./usodiccionario < data.txt.

Definition at line 101 of file usodiccionario.cpp.

```
00101           {
00102     //Diccionario 1
00103     Diccionario<string,string> D1;
00104     cin>>D1;
```

```
00105     cout « "Diccionario 1:" « endl;
00106     cout«D1;
00107
00108     if(D1.size() == 0)
00109         cout « "El diccionario 1 esta vacio" « endl;
00110     //Diccionario 2
00111     separador();
00112     Diccionario<string,string> D2;
00113     cin»D2;
00114     cout « "Diccionario 2:" « endl;
00115     cout«D2;
00116
00117     if(D2.size() == 0)
00118         cout « "El diccionario 2 esta vacio" « endl;
00119     //removePalabra: elimina una palabra del diccionario
00120     separador();
00121     cout « "Palabra a eliminar del Diccionario 1:" « endl;
00122     string a;
00123     cin»a;
00124     D1.removePalabra(a);
00125     cout « "Diccionario despues de borrar " « a « endl;
00126     cout « D1;
00127
00128     //operator+: une dos diccionarios
00129     separador();
00130     cout « "La union de los dos diccionarios es " « endl;
00131     Diccionario<string,string> D3 = D1+D2;
00132     cout « D3;
00133
00134     //getDiccionarioEntreClaves: Devuelve elementos entre dos claves
00135     separador();
00136     cout « "Escribes las 2 claves entre las que buscar" « endl;
00137     string clave1, clave2;
00138     cin » clave1 » clave2;
00139     cout « "Los elementos entre las claves '" « clave1 « "' y '" « clave2 «
00140         "' en el diccionario fusionado es:" « endl;
00141     Diccionario<string,string> DEntreClaves = D3.getElementosEntreClaves(clave1,clave2);
00142     cout « DEntreClaves « endl;
00143
00144     /* //Para borrar definiciones. Funcion extra "accidental"
00145     string b;
00146
00147     cout«"Introduce una palabra"«endl;
00148     cin»b;
00149
00150     //Borra las definiciones de una palabra
00151     //D1.removeSignificados_Palabra(b);
00152
00153     /*
00154     list<string>l=D1.getInfo_Asoc(b);
00155
00156     //Imprime las definiciones de una palabra
00157     cout « "Las definiciones de " « b « " son:" « endl;
00158     if (l.size()>0)
00159         EscribeSigni(l);
00160     */
00161 }
```

Here is the call graph for this function:



5.3.2.3 operator<<()

```
ostream & operator<< (
            ostream & os,
            const Diccionario< string, string > & D )
```

Operator<<. Obsérvese el uso de 2 tipos diferentes de iteradores. Uno sobre listas de listas y otro sobre listas.

Parameters

os	El operador de salida
D	El diccionario a escribir

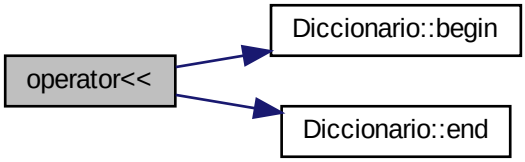
Returns

os La salida estandar

Definition at line 20 of file usodiccionario.cpp.

```
00020                                                                   {
00021
00022       list<data<string,string> >::const_iterator it;
00023
00024       for (it=D.begin(); it!=D.end(); ++it){
00025
00026             list<string>::const_iterator it_s;
00027
00028             os«endl«(*it).clave«endl«" informacion asociada:"«endl;
00029             for (it_s=(*it).info_asoci.begin();it_s!=(*it).info_asoci.end();++it_s){
00030                   os«(*it_s)«endl;
00031             }
00032             os«"************************************"«endl;
00033       }
00034
00035       return os;
00036 }
```

Here is the call graph for this function:



5.3.2.4 operator>>()

```
istream & operator>> (
            istream & is,
            Diccionario< string, string > & D )
```

Operator >>. El formato de la entrada es: numero de claves en la primera linea clave-iésima retorno de carro numero de informaciones asociadas en la siguiente linea y en cada linea obviamente la informacion asociada.

Parameters

is	El operador de entrada
D	El diccionario a leer

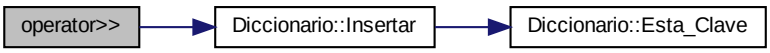
Returns

is La entrada estandar

Definition at line 48 of file usodiccionario.cpp.

```
00048                                                                    {
00049       int np;
00050       is»np;
00051       is.ignore();//quitamos \n
00052       Diccionario<string,string> Daux;
00053       for (int i=0;i<np; i++){
00054             string clave;
00055
00056             getline(is,clave);
00057
00058             int ns;
00059             is»ns;
00060             is.ignore();//quitamos \n
00061             list<string>laux;
00062             for (int j=0;j<ns; j++){
00063                   string s;
00064                   getline(is,s);
00065
00066                   // cout«"Significado leido "«s«endl;
00067                   laux.insert(laux.end(),s);
00068             }
00069             Daux.Insertar(clave,laux);
00070
00071       }
00072       D=Daux;
00073       return is;
00074 }
```

Here is the call graph for this function:



5.3.2.5 separador()

```
void separador ( )
```

separa las salidas de texto para una mejor lectura

Definition at line 90 of file usodiccionario.cpp.

```
00090                {
00091     for(int i = 0; i < 100; i++)
00092         cout « "=";
00093     cout « endl;
00094 }
```

5.4 usodiccionario.cpp

Go to the documentation of this file.

```
00001
00008 #include <iostream>
00009 #include "diccionario.h"
00010
00011 //COMANDO EJECUCION: ./usodiccionario < ../datos/data.txt
00012
00020 ostream & operator«(ostream & os, const Diccionario<string,string> & D){
00021
00022       list<data<string,string> >::const_iterator it;
00023
00024       for (it=D.begin(); it!=D.end(); ++it){
00025
00026             list<string>::const_iterator it_s;
00027
00028             os«endl«(*it).clave«endl«" informacion asociada:"«endl;
00029             for (it_s=(*it).info_asoci.begin();it_s!=(*it).info_asoci.end();++it_s){
00030                   os«(*it_s)«endl;
00031             }
00032             os«"************************************"«endl;
00033       }
00034
00035       return os;
00036 }
00037
00048 istream & operator »(istream & is,Diccionario<string,string> &D){
00049       int np;
00050       is»np;
00051       is.ignore();//quitamos \n
00052       Diccionario<string,string> Daux;
00053       for (int i=0;i<np; i++){
00054             string clave;
00055
00056             getline(is,clave);
00057
00058             int ns;
00059             is»ns;
00060             is.ignore();//quitamos \n
00061             list<string>laux;
00062             for (int j=0;j<ns; j++){
00063                   string s;
00064                   getline(is,s);
00065
00066                   // cout«"Significado leido "«s«endl;
00067                   laux.insert(laux.end(),s);
00068             }
00069             Daux.Insertar(clave,laux);
00070
00071       }
00072       D=Daux;
00073       return is;
00074 }
00075
00080 void EscribeSigni(const list<string>&l){
00081       list<string>::const_iterator it_s;
00082       for (it_s=l.begin();it_s!=l.end();++it_s){
00083             cout«*it_s«endl;
00084       }
00085 }
00086
00090 void separador(){
00091     for(int i = 0; i < 100; i++)
00092         cout « "=";
00093     cout « endl;
00094 }
00095
00101 int main(){
00102     //Diccionario 1
00103     Diccionario<string,string> D1;
00104     cin»D1;
00105     cout « "Diccionario 1:" « endl;
00106     cout«D1;
00107
00108     if(D1.size() == 0)
00109         cout « "El diccionario 1 esta vacio" « endl;
00110     //Diccionario 2
00111     separador();
00112     Diccionario<string,string> D2;
00113     cin»D2;
00114     cout « "Diccionario 2:" « endl;
00115     cout«D2;
00116
00117     if(D2.size() == 0)
```

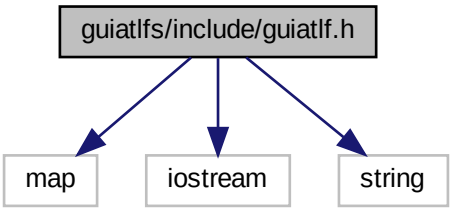
```
00118         cout « "El diccionario 2 esta vacio" « endl;
00119     //removePalabra: elimina una palabra del diccionario
00120     separador();
00121     cout « "Palabra a eliminar del Diccionario 1:" « endl;
00122     string a;
00123     cin»a;
00124     D1.removePalabra(a);
00125     cout « "Diccionario despues de borrar " « a « endl;
00126     cout « D1;
00127
00128     //operator+: une dos diccionarios
00129     separador();
00130     cout « "La union de los dos diccionarios es " « endl;
00131     Diccionario<string,string> D3 = D1+D2;
00132     cout « D3;
00133
00134     //getDiccionarioEntreClaves: Devuelve elementos entre dos claves
00135     separador();
00136     cout « "Escribes las 2 claves entre las que buscar" « endl;
00137     string clave1, clave2;
00138     cin » clave1 » clave2;
00139     cout « "Los elementos entre las claves '" « clave1 « "' y '" « clave2 «
00140         "' en el diccionario fusionado es:" « endl;
00141     Diccionario<string,string> DEntreClaves = D3.getElementosEntreClaves(clave1,clave2);
00142     cout « DEntreClaves « endl;
00143
00144     /* //Para borrar definiciones. Funcion extra "accidental"
00145     string b;
00146
00147     cout«"Introduce una palabra"«endl;
00148     cin»b;
00149
00150     //Borra las definiciones de una palabra
00151     //D1.removeSignificados_Palabra(b);
00152
00153     /*
00154     list<string>l=D1.getInfo_Asoc(b);
00155
00156     //Imprime las definiciones de una palabra
00157     cout « "Las definiciones de " « b « " son:" « endl;
00158     if (l.size()>0)
00159         EscribeSigni(l);
00160     */
00161 }
00162
```

5.5 guiatlfs/include/guiatlf.h File Reference

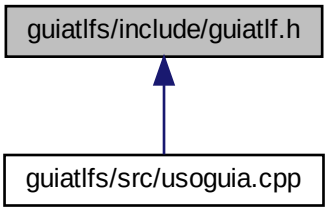
TDA guia_tlf. Es un .h pero funciona como .cpp.

```
#include <map>
#include <iostream>
#include <string>
```

Include dependency graph for guiatlf.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Guia_Tlf

  T.D.A. Guia_Tlf.
- class Guia_Tlf::iterator

  clase para iterar sobre la guia

Functions

- istream & operator>> (istream &is, pair< string, string > &d)
- ostream & operator<< (ostream &os, const pair< const string, string > &d)

5.5.1 Detailed Description

TDA guia_tlf. Es un .h pero funciona como .cpp.

Author

Yeray Lopez Ramirez

Date

15 de diciembre de 2021

Definition in file guiatlf.h.

5.5.2 Function Documentation

5.5.2.1 operator<<()

```
ostream & operator<< (
            ostream & os,
            const pair< const string, string > & d )
```

Definition at line 20 of file guiatlf.h.

```
00020                                                                   {
00021
00022       os«d.first«'\t'«d.second«endl;
00023       return os;
00024 }
```

5.5.2.2 operator>>()

```
istream & operator>> (
            istream & is,
            pair< string, string > & d )
```

Definition at line 13 of file guiatlf.h.

```
00013                                                   {
00014
00015       getline(is,d.first,'\t');
00016       getline(is,d.second);
00017       return is;
00018 }
```

5.6 guiatlf.h

Go to the documentation of this file.

```
00001
00007 #ifndef _GUIA_TLF_H
00008 #define _GUIA_TLF_H
00009 #include <map>
00010 #include <iostream>
00011 #include <string>
00012 using namespace std;
00013 istream & operator»(istream &is,pair<string,string> &d){
00014
00015       getline(is,d.first,'\t');
00016       getline(is,d.second);
00017       return is;
00018 }
00019
00020 ostream & operator«(ostream &os,const pair<const string,string> &d){
00021
00022       os«d.first«'\t'«d.second«endl;
00023       return os;
00024 }
00025
00026
00043 class Guia_Tlf{
00058       private:
00059             map<string,string> datos; //si admites que haya nombres repetidos tendrías que usar un
00060                         //multimap
00061
00062       public:
00063
00069             //si fuese un multimap no podriamos usar []. Ademas que deberiamos devolver p.e un vector
       con todos
00070             // los telefonos asociados a dicho nombre
00071             string & operator[](const string &nombre) {
00072                   return datos[nombre];
00073             }
00074
00075
00076             string  gettelefono(const string & nombre){
00077                   map<string,string>::iterator it=datos.find(nombre);
```

```
00078                   if (it==datos.end())
00079                     return string("");
00080                   else return it->second;
00081             }
00082
00090             pair<map<string,string>::iterator,bool>  insert(string nombre, string tlf){
00091                   pair<string,string> p (nombre,tlf);
00092                   pair< map<string,string> ::iterator,bool> ret;
00093
00094                   ret=datos.insert(p); //datos.insert(datos.begin(),p); tambien funcionaría
00095                   return ret;
00096
00097             }
00098
00106             pair<map<string,string>::iterator,bool>  insert(pair<string,string> p){
00107
00108                   pair<map<string,string> ::iterator,bool> ret;
00109
00110                   ret=datos.insert(p); //datos.insert(datos.begin(),p); tambien funcionaría
00111                   return ret;
00112
00113             }
00114
00120             void borrar(const string &nombre){
00121                map<string,string>::iterator itlow = datos.lower_bound(nombre);//el primero que tiene
    dicho nombre
00122                map<string,string>::iterator itupper = datos.upper_bound(nombre);//el primero que ya no
    tiene dicho nombre
00123                datos.erase(itlow,itupper);//borramos todos aquellos que tiene dicho nombre
00124                //OTRA ALTERNATIVA
00125                //pair<map<string,string>::iterator,map<string,string>::iterator>ret;
00126                //ret = datos.equal_range(nombre
00127                //datos.erase(ret.first,ret.second);
00128             }
00129
00135             //con map siempre hay uno con multimap puede existir mas de uno
00136             void borrar(const string &nombre,const string &tlf){
00137                    map<string,string>::iterator itlow = datos.lower_bound(nombre);//el primero que
    tiene dicho nombre
00138                map<string,string>::iterator itupper = datos.upper_bound(nombre);//el primero que ya no
    tiene dicho nombre
00139                map<string,string>::iterator it;
00140                bool salir =false;
00141                for (it=itlow; it!=itupper && !salir;++it){
00142                     if (it->second==tlf){
00143                     datos.erase(it);
00144                     salir =true;
00145                      }
00146                }
00147
00148             }
00153             int size()const{
00154                   return datos.size();
00155             }
00162             //al ser un map debe de ser 0 o 1. Si fuese un multimap podríamos tener mas de uno
00163             unsigned int contabiliza(const string &nombre){
00164                   return datos.count(nombre);
00165             }
00166
00170             void clear(){
00171                   datos.clear();
00172             }
00178             Guia_Tlf operator+(const Guia_Tlf & g){
00179             Guia_Tlf aux(*this);
00180             map<string,string>::const_iterator it;
00181             for (it=g.datos.begin();it!=g.datos.end();++it){
00182                aux.insert(it->first,it->second);
00183             }
00184             return aux;
00185
00186             }
00187
00193             Guia_Tlf operator-(const Guia_Tlf & g){
00194             Guia_Tlf aux(*this);
00195             map<string,string>::const_iterator it;
00196             for (it=g.datos.begin();it!=g.datos.end();++it){
00197                aux.borrar(it->first,it->second);
00198             }
00199             return aux;
00200
00201             }
00208             Guia_Tlf previos(const string &nombre,const string &tlf){
00209             map<string,string>::value_compare vc=datos.value_comp();
00210             //map<string,string>::key_compare vc=datos.key_comp();
00211             Guia_Tlf aux;
00212             pair<string,string>p(nombre,tlf);
00213             map<string,string>::iterator it=datos.begin();
```

```
00214             while (vc(*it,p)){
00215                   aux.insert(*it++);
00216
00217             }
00218             return aux;
00219
00220              }
00221
00227             void modificarTlf(string nombre, string tlf){
00228                   map<string,string>::iterator it=datos.find(nombre);
00229                   if(it != datos.end()){
00230                      pair<string,string> p(it->first,tlf);
00231                      datos.erase(it);
00232                      datos.insert(p);
00233                   }
00234             }
00235
00241             Guia_Tlf TlfsPorLetra(char letra){
00242                 Guia_Tlf tlfs;
00243                 int contador = 1; //Para enumerar los telefonos del map
00244                 map<string,string>::iterator it;
00245                 for(it = datos.begin(); it != datos.end(); ++it)
00246                     if(letra == (*it).first[0])
00247                         tlfs.insert(to_string(contador++) + ".",(*it).second);
00248
00249                 return tlfs;
00250             }
00251
00258             Guia_Tlf TelfsEntreNombres(const string &nombre_1, const string &nombre_2){
00259                 Guia_Tlf tlfs;
00260                 map<string,string>::iterator itlow = datos.find(nombre_1);
00261                 map<string,string>::iterator itupper = datos.find(nombre_2);
00262
00263                 int contador = 1,
00264                     dis1 = distance(datos.begin(),itlow),
00265                     dis2 = distance(datos.begin(), itupper);
00266                 if(itlow != datos.end() && dis1 <= dis2){
00267                   tlfs.insert(to_string(contador++) + ".",itlow->second);
00268                   while(dis1 < dis2){
00269                      itlow++; dis1++;
00270                      tlfs.insert(to_string(contador++) + ".",itlow->second);
00271                   }
00272                 }
00273                 return tlfs;
00274             }
00275
00282             friend ostream & operator<<(ostream & os, Guia_Tlf & g){
00283                   map<string,string>::iterator it;
00284                   for (it=g.datos.begin(); it!=g.datos.end();++it){
00285                     os<<it->first<<"\t"<<it->second<<endl;
00286                   }
00287                   return os;
00288             }
00289
00297             friend istream & operator>>(istream & is, Guia_Tlf & g){
00298                   pair<string,string> p;
00299                   Guia_Tlf aux;
00300
00301                   while (is>>p){
00302                   aux.insert(p);
00303                   }
00304                   g=aux;
00305                   return is;
00306             }
00307
00311             class iterator{
00312              private:
00313               map<string,string>::iterator it;
00314              public:
00315                iterator & operator++(){
00316                 ++it;
00317                 return *this;
00318                }
00319                iterator & operator--(){
00320                --it;
00321                return *this;
00322                }
00323                pair<const string,string> &operator *(){
00324               return *it;
00325                }
00326                bool operator ==(const iterator &i){
00327               return i.it==it;
00328                }
00329
00330                bool operator !=(const iterator &i){
00331               return i.it!=it;
00332                }
```

```
00333               friend class Guia_Tlf;
00334            };
00335
00339            iterator  begin(){
00340            iterator i;
00341            i.it=datos.begin();
00342            return i;
00343            }
00347            iterator  end(){
00348            iterator i;
00349            i.it=datos.end();
00350            return i;
00351            }
00352
00353
00354
00355
00356 };
00357 #endif
00358
```

5.7 usoguia.cpp

```
00001
00007 #include "guiatlf.h"
00008 #include <fstream>
00009 int main(int argc , char * argv[]){
00010    if (argc!=2){
00011       cout«"Dime el nombre del fichero con la guia"«endl;
00012    return 0;
00013    }
00014    ifstream f(argv[1]);
00015    if (!f){
00016       cout«"No puedo abrir el fichero "«argv[1]«endl;
00017       return 0;
00018    }
00019    Guia_Tlf g;
00020
00021    f»g;
00022    cout«"La guia insertada: " « endl « g«endl;
00023    cin.clear();
00024    cout«"Dime un nombre sobre el que quieres obtener el telefono[Escribe 'quit' o 'q' para no
       buscar]"«endl;
00025    string n;
00026    bool seguirleyendo = true;
00027
00028    getline(cin,n);
00029    if(n == "quit" || n == "q")
00030       seguirleyendo = false;
00031    //Busca nombre en la guia para obtener su tlf
00032    while (seguirleyendo){
00033       cout«"Buscando "«n«"...."«endl;
00034       string tlf = g.gettelefono(n);
00035       if (tlf=="")
00036          cout«"No existe ese nombre en la guia"«endl;
00037       else
00038          cout«"El telefono es "«tlf«endl;
00039
00040       cout«"[Escribe 'quit' o 'q' para salir del bucle] Dime un nombre sobre el que quieres obtener el
       telefono"«endl;
00041       getline(cin,n);
00042       if(n == "quit" || n == "q")
00043          seguirleyendo = false;
00044    }
00045    seguirleyendo = true;
00046
00047    //Modifica el tlf de un contacto
00048    string tlf;
00049    cout«"Dime un nombre y su telefono para modificarlo[Escribe 'quit' o 'q' para no modificar]:"«endl;
00050    getline(cin,n);
00051    if(n == "quit" || n == "q")
00052       seguirleyendo = false;
00053    else
00054       getline(cin,tlf);
00055
00056    while (seguirleyendo){
00057       cout«"Modificando tlf de "«n«"...."«endl;
00058       string tlfO = g.gettelefono(n);
00059       if (tlfO=="")
00060          cout«"No existe ese nombre en la guia"«endl;
00061       else{
00062          g.modificarTlf(n, tlf);
00063          cout«"El telefono original era "«tlfO« " y ahora es " « g.gettelefono(n) « endl;
```

```
00064       }
00065       cout«"[Escribe 'quit' o 'q' para dejar de modificar]Dime un nombre y su telefono para
      modificarlo:"«endl;
00066       getline(cin,n);
00067       if(n == "quit" || n == "q")
00068          seguirleyendo = false;
00069       else
00070          getline(cin,tlf);
00071    }
00072    seguirleyendo = true;
00073    //Saca telefonos de nombres que empiezen por una letra
00074    Guia_Tlf entreLetras;
00075    char letra;
00076    cout « "Introduce la letra por la que buscar telefonos cuyos nombres empiecen por esa
      letra:[Escribe 'q' para no buscar]" « endl;
00077    cin.clear();
00078    cin » letra;
00079    if(letra == 'q')
00080       seguirleyendo = false;
00081    while (seguirleyendo){
00082       cout«"Buscando tlfs cuyos nombres asociados empiezan por '" « letra «"' ...."«endl;
00083       entreLetras = g.TlfsPorLetra(letra);
00084       if (entreLetras.size()==0)
00085          cout«"No existe numeros cuyos nombres empiecen por '" « letra « "'" « endl;
00086       else
00087             cout « entreLetras « endl;
00088       cout«"[Escribe 'q' para salir]Dime una letra para listar numeros cuyos nombres empiecen por esa
      letra:"«endl;
00089       cin » letra;
00090       if(letra == 'q')
00091          seguirleyendo = false;
00092    }
00093    //Saca telefonos entre dos nombres
00094    string nombre1,nombre2;
00095    cout « "Dime entre que nombres quieres quieres sacar la guia: " « endl;
00096    cin.ignore(); //Ignoramos el \n del cin anterior
00097    getline(cin,nombre1);
00098    getline(cin,nombre2);
00099    Guia_Tlf entreNombres = g.TelfsEntreNombres(nombre1,nombre2);
00100    cout « "Los telefonos entre " « nombre1 « " y " « nombre2 « " es: " « endl « entreNombres « endl;
00101
00102    //Lista la guia final
00103    cout«"Listando la guia resultante con iteradores:"«endl;
00104    Guia_Tlf::iterator it;
00105    for (it=g.begin(); it!=g.end(); ++it)
00106       cout«*it«endl;
00107
00108 }
```

Index