

Practica4 : Diccionario/Guia telefonos

Generated by Doxygen 1.9.2

1 Rep del TDA Guia_Tlf	1
1.1 Invariante de la representación	1
1.2 Función de abstracción	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 data< T, U > Struct Template Reference	7
4.1.1 Detailed Description	7
4.1.2 Member Data Documentation	7
4.1.2.1 clave	7
4.1.2.2 info_asoci	8
4.2 Diccionario< T, U > Class Template Reference	8
4.2.1 Detailed Description	9
4.2.2 Constructor & Destructor Documentation	9
4.2.2.1 Diccionario() [1/2]	9
4.2.2.2 Diccionario() [2/2]	9
4.2.2.3 ~Diccionario()	10
4.2.3 Member Function Documentation	10
4.2.3.1 AddSignificado_Palabra()	10
4.2.3.2 begin() [1/2]	11
4.2.3.3 begin() [2/2]	11
4.2.3.4 end() [1/2]	12
4.2.3.5 end() [2/2]	12
4.2.3.6 Esta_Clave()	12
4.2.3.7 getElementosEntreClaves()	13
4.2.3.8 getInfo_Asoc()	14
4.2.3.9 Insertar()	15
4.2.3.10 operator+()	16
4.2.3.11 operator=()	16
4.2.3.12 removePalabra()	17
4.2.3.13 removeSignificados_Palabra()	18
4.2.3.14 size()	18
4.3 Guia_Tlf Class Reference	19
4.3.1 Detailed Description	20
4.3.2 Member Function Documentation	20
4.3.2.1 begin()	20
4.3.2.2 borrar() [1/2]	20
4.3.2.3 borrar() [2/2]	21

4.3.2.4 clear()	22
4.3.2.5 contabiliza()	22
4.3.2.6 end()	22
4.3.2.7 gettelefono()	23
4.3.2.8 insert() [1/2]	23
4.3.2.9 insert() [2/2]	23
4.3.2.10 modificarTlf()	24
4.3.2.11 operator+()	24
4.3.2.12 operator-()	25
4.3.2.13 operator[]()	26
4.3.2.14 previos()	26
4.3.2.15 size()	27
4.3.2.16 TlfSEntreNombres()	27
4.3.2.17 TlfSPorLetra()	28
4.3.3 Friends And Related Function Documentation	29
4.3.3.1 operator<<	29
4.3.3.2 operator>>	30
4.4 Diccionario< T, U >::iterator Class Reference	30
4.4.1 Detailed Description	31
4.4.2 Member Function Documentation	31
4.4.2.1 operator"!=()	31
4.4.2.2 operator*()	31
4.4.2.3 operator++()	31
4.4.2.4 operator--()	32
4.4.2.5 operator==()	32
4.4.3 Friends And Related Function Documentation	32
4.4.3.1 Diccionario	32
4.5 Guia_Tlf::iterator Class Reference	32
4.5.1 Detailed Description	33
4.5.2 Member Function Documentation	33
4.5.2.1 operator"!=()	33
4.5.2.2 operator*()	33
4.5.2.3 operator++()	33
4.5.2.4 operator--()	34
4.5.2.5 operator==()	34
4.5.3 Friends And Related Function Documentation	34
4.5.3.1 Guia_Tlf	34
5 File Documentation	35
5.1 diccionario/include/diccionario.h File Reference	35
5.1.1 Detailed Description	36
5.1.2 Function Documentation	36

5.1.2.1 operator<()	36
5.2 diccionario.h	37
5.3 diccionario/src/usodiccionario.cpp File Reference	40
5.3.1 Detailed Description	41
5.3.2 Function Documentation	41
5.3.2.1 EscribeSigni()	41
5.3.2.2 main()	41
5.3.2.3 operator<<()	43
5.3.2.4 operator>>()	43
5.3.2.5 separador()	44
5.4 usodiccionario.cpp	45
5.5 guiatlfs/include/guiatlf.h File Reference	46
5.5.1 Detailed Description	47
5.5.2 Function Documentation	47
5.5.2.1 operator<<()	48
5.5.2.2 operator>>()	48
5.6 guiatlf.h	48
5.7 usoguia.cpp	51
Index	53

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:

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	7
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	8
Guia_Tlf	
T.D.A. Guia_Tlf	19
Diccionario< T, U >::iterator	
Clase para iterar sobre la guia	30
Guia_Tlf::iterator	
Clase para iterar sobre la guia	32

Chapter 3

File Index

3.1 File List

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

diccionario/include/ diccionario.h		
TDA diccionario. Es un .h pero funciona como .cpp	35
diccionario/src/ usodiccionario.cpp		
Archivo de ejemplo para probar el TDA diccionario. Se ejecuta ./usodiccionario < archivodeentrada o escribiendo los datos a mano(mas engorroso esta ultima)	40
guiatlfs/include/ guiatlfs.h		
TDA guia_tlf. Es un .h pero funciona como .cpp	46
guiatlfs/src/ usoguia.cpp	51

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 diccionario 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 información asociada 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 información 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 informacin (una lista) asociada a una clave p. Podran 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 tamao 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 ms facil la implementacin de algunos operadores o funciones de la parte pblica. Copiar copia un diccioario en otro y borrar elimina todos los elementos de un diccionario. La implementacin de copiar puede hacerse usando iteradores o directamente usando la funcin 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

<i>D</i>	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 asociada 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.

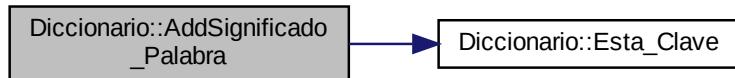
Parameters

<i>s</i>	La definicion a añadir
<i>clave</i>	La palabra a la que queremos añadir la definicion <i>s</i>

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 significado 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 >\nlist< 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\n00344           return datos.begin();\n00345       }
```

4.2.3.3 begin() [2/2]

```
template<class T , class U >\nlist< 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\n00359           return datos.begin();\n00360       }
```

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

<code>p</code>	La palabra a buscar
<code>it_out</code>	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

<i>clavea</i>	La clave inferior
<i>claveb</i>	La clave superior

Returns

D Un nuevo diccionario con las claves intermedias con sus definiciones

Definition at line 246 of file diccionario.h.

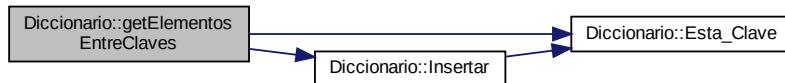
```

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     }
```

```

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

<i>p</i>	La palabra de la cual se busca su lista de definiciones
----------	---

Returns

La lista de definiciones de la palabra si existe, devuelve una vacía 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

<i>clave</i>	La palabra a insertar
<i>info</i>	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         datos.insert(it,p);
00153     }
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

<i>d</i>	Diccionario a sumar, añadiendo palabras y definiciones nuevas
----------	---

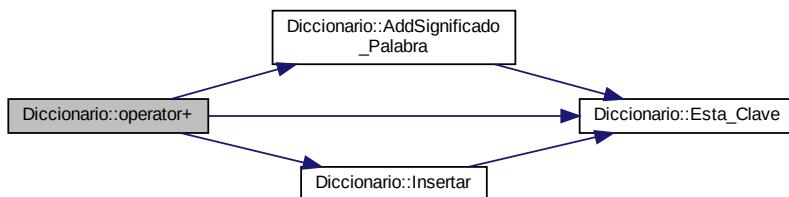
Returns

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
00223             palabra
00224                 typename list<U>::const_iterator itinfoaux;
00225                 bool existedef = false;
00226                 for (itinfoaux=(*it).info_asoci.begin();itinfoaux !=
00227 (*it).info_asoci.end();++itinfoaux){
00228                     existedef = false;
00229                     for(itinfoaux=(*it2).info_asoci.begin();itinfoaux != (*it2).info_asoci.end()
00230 && !existedef;++itinfoaux)
00231                         if((*itinfoaux) == (*itinfo))
00232                             existedef = true;
00233                         if(!existedef)
00234                             aux.AddSignificado_Palabra((*itinfo),(*it).clave);
00235                         else //Si no existe la palabra, la mete
00236                             aux.Insertar((*it).clave,(*it).info_asoci);
00237
00238     }
00239 }
```

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

<i>D</i>	Diccionario a ser asignado
----------	----------------------------

Returns

this el propio objeto

Definition at line 93 of file diccionario.h.

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

<i>clave</i>	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

<i>clave</i>	La palabra a la que borrar todas sus definiciones, sin borrar la palabra
--------------	--

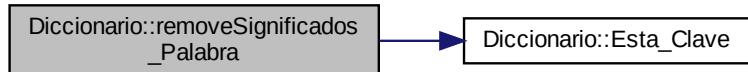
Returns

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     {
00304         return datos.size();
00305     }
```

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 parametro.
- [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

<i>nombre</i>	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           {
00122             map<string, string>::iterator itlow = datos.lower_bound(nombre); //el primero que tiene
00123             dicho nombre
00124             map<string, string>::iterator itupper = datos.upper_bound(nombre); //el primero que ya no
00125             tiene dicho nombre
00126             datos.erase(itlow, itupper); //borramos todos aquellos que tiene dicho nombre
00127             //OTRA ALTERNATIVA
00128             //pair<map<string, string>::iterator, map<string, string>::iterator>ret;
00129             //ret = datos.equal_range(nombre
00130             //datos.erase(ret.first, ret.second);
00131           }
```

4.3.2.3 [borrar\(\)](#) [2/2]

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

Borrar un telefono.

Parameters

<i>nombre</i>	nombre que se quiere borrar y telefono asociado
---------------	---

Note

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

Definition at line 136 of file [guiatlf.h](#).

```
00136
00137           {
00138             map<string, string>::iterator itlow = datos.lower_bound(nombre); //el primero que
00139             tiene dicho nombre
00140             map<string, string>::iterator itupper = datos.upper_bound(nombre); //el primero que ya no
00141             tiene dicho nombre
00142             map<string, string>::iterator it;
00143             bool salir =false;
00144             for (it=itlow; it!=itupper && !salir; ++it){
00145               if (it->second==tlf){
00146                 datos.erase(it);
00147                 salir =true;
00148               }
00149             }
00150           }
```

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

<i>nombre</i>	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

<i>p</i>	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 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

<i>nombre</i>	nombre clave del nuevo telefono
<i>tlf</i>	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 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

<i>nombre</i>	El nombre de contacto cuyo telefono se quiere modificar
<i>tlf</i>	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

<i>g</i>	guia que se une
----------	-----------------

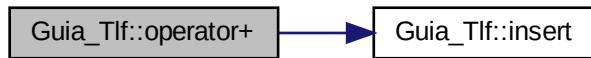
Returns

: 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

<i>g</i>	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 }
```

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

<i>nombre</i>	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

<i>nombre</i>	nombre delimitador
<i>tlf</i>	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     return aux;
00218
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     {
00155         return datos.size();
}
```

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

<i>nombre_1</i>	El nombre a partir empieza a listar
<i>nombre_2</i>	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++;
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

<i>letra</i>	La letra a partir del cual se buscan los telefonos
--------------	--

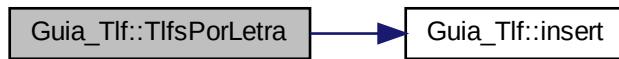
Returns

`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

<code>os</code>	flujo de salida. Es MODIFICADO
<code>g</code>	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

<i>is</i>	flujo de entrada. ES MODIFICADO
<i>g</i>	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             return i.it!=it;
00331
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

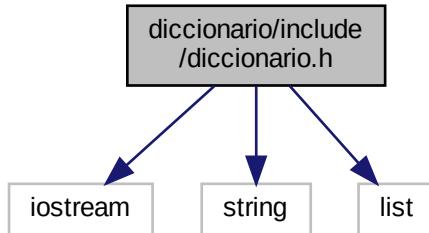
File Documentation

5.1 diccionario/include/diccionario.h File Reference

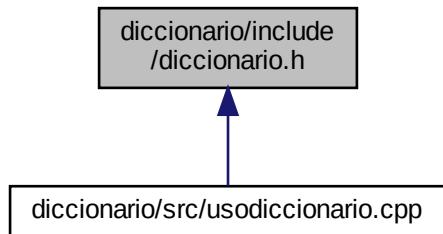
TDA diccionario. Es un .h pero funciona como .cpp.

```
#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 diccionario 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 functor.

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 functor.

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 _DICCCIONARIO_H
00008 #define _DICCCIONARIO_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     list<data<T,U> > datos;
00050
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 significado 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
00223             palabra
00224                 typename list<U>::const_iterator itinfod, itinfoaux;
00225                 bool existedef = false;
00226                 for (itinfod=(*it).info_asoci.begin();itinfod != (*it).info_asoci.end();++itinfod)
00227                     existedef = false;
00228                     for(itinfoaux=(*it2).info_asoci.begin();itinfoaux != (*it2).info_asoci.end()
00229                         && !existedef;++itinfoaux)
00230                         if((*itinfoaux) == (*itinfod))
00231                             existedef = true;
00232                         if(!existedef)
00233                             aux.AddSignificado_Palabra((*itinfod),(*it).clave);
00234                         }
00235                         else //Si no existe la palabra, la mete
00236                             aux.Insertar((*it).clave,(*it).info_asoci);
00237                         }
00238             return aux;
00239     }
00246 Diccionario getElementosEntreClaves(const T & clavea, const T & claveb){
00247     Diccionario<T,U> D;

```

```

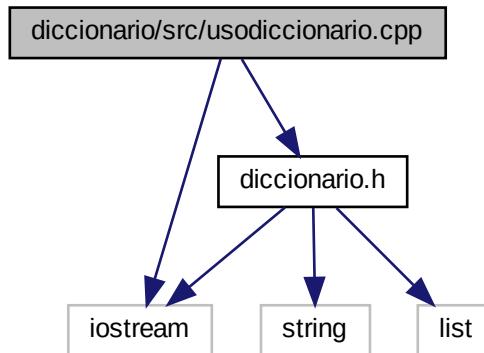
00248     typename list<data<T,U>::iterator ita, itb;
00249     bool terminar = false;
00250     int disa,disb;
00251     if (Esta_Clave(clavea, ita) && Esta_Clave(claveb, itb)){
00252         disa = distance(datos.begin(), ita);
00253         disb = distance(datos.begin(), itb);
00254         if(disa <= 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     else
00271         cerr << "Una de las claves introducidas no existe" << endl;
00272
00273     return D;
00274 }
00275
00276
00277     list<U> getInfo_Asoc(const T & p) {
00278     typename list<data<T,U> ::iterator it;
00279
00280     if (!Esta_Clave(p,it)){
00281         return list<U>();
00282     }
00283     else{
00284         return (*it).info_asoci;
00285     }
00286 }
00287
00288
00289     int size() const{
00290         return datos.size();
00291     }
00292
00293
00294
00295
00296
00297
00298     class iterator{
00299     private:
00300         typename list<T,list<U>::iterator> it;
00301     public:
00302         iterator & operator++(){
00303             ++it;
00304             return *this;
00305         }
00306         iterator & operator--(){
00307             --it;
00308             return *this;
00309         }
00310         list<const T,list<U> &operator *(){
00311             return *it;
00312         }
00313         bool operator ==(const iterator &i){
00314             return i.it==it;
00315         }
00316         bool operator !=(const iterator &i){
00317             return i.it!=it;
00318         }
00319         friend class Diccionario;
00320     };
00321
00322
00323     typename list<data<T,U> ::iterator begin(){
00324         return datos.begin();
00325     }
00326     typename list<data<T,U> ::iterator end(){
00327         return datos.end();
00328     }
00329
00330
00331
00332
00333
00334
00335     typename list<data<T,U> ::const_iterator begin() const{
00336         return datos.begin();
00337     }
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000
02001
02002
02003
02004
02005
```

```
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 informacion 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

/	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 diccionario 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.

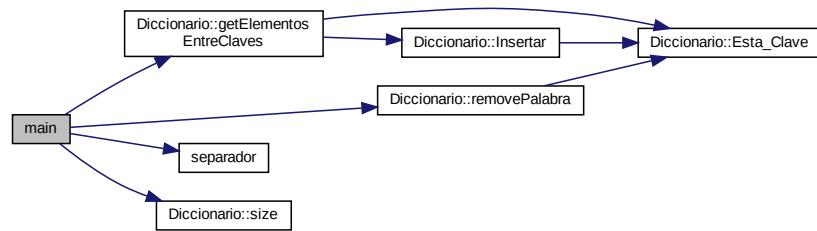
Definition at line 101 of file [usodiccionario.cpp](#).

```
00101
00102     {
00103     //Diccionario 1
00104     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

<i>os</i>	El operador de salida
<i>D</i>	El diccionario a escribir

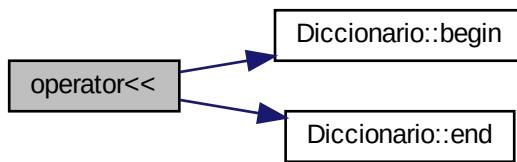
Returns

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

<i>is</i>	El operador de entrada
<i>D</i>	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     D=Daux;
00072     return is;
00073 }
```

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     list<data<string,string> ::const_iterator it;
00023
00024     for (it=D.begin(); it!=D.end(); ++it){
00025         list<string>::const_iterator it_s;
00027
00028         os<<(*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
00074     return is;
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)

```

```

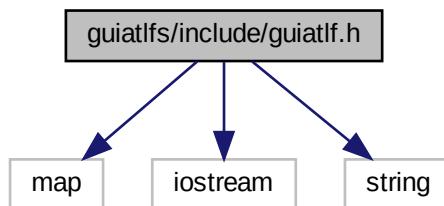
00118      cout << "El diccionario 2 esta vacio" << endl;
00119  //removePalabra: elimina una palabra del diccionario
00120  separador();
00121  cout << "Palabra a eliminar del Diccionario 1:" << endl;
00122  string a;
00123  cin>>a;
00124  D1.removePalabra(a);
00125  cout << "Diccionario despues de 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

```

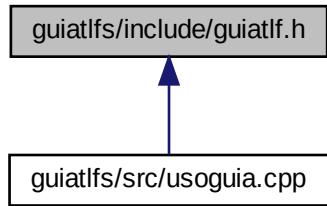
5.5 guiatlfs/include/guiatlf.h File Reference

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 [guiatl.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
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
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
00018 }
00019
00020 ostream & operator<<(ostream &os,const pair<const string,string> &d){
00021
00022     os<<d.first<<'\'t'<<d.second<<endl;
00023
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
00070         //con todos
00071         // los telefonos asociados a dicho nombre
00072         string & operator[](const string &nombre) {
00073             return datos[nombre];
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     }
00114
00120     void borrar(const string &nombre){
00121         map<string,string>::iterator itlow = datos.lower_bound(nombre);//el primero que tiene
00122         map<string,string>::iterator itupper = datos.upper_bound(nombre);//el primero que ya no
00123         tiene dicho nombre
00124         datos.erase(itlow,itupper);//borramos todos aquellos que tiene dicho nombre
00125         //OTRA ALTERNATIVA
00126         //pair<map<string,string>::iterator,map<string,string>::iterator>ret;
00127         //ret = datos.equal_range(nombre
00128         //datos.erase(ret.first,ret.second);
00129     }
00135
00136     //con map siempre hay uno con multimap puede existir mas de uno
00137     void borrar(const string &nombre,const string &tlf){
00138         map<string,string>::iterator itlow = datos.lower_bound(nombre);//el primero que ya no
00139         tiene dicho nombre
00140         map<string,string>::iterator itupper = datos.upper_bound(nombre);//el primero que ya no
00141         tiene dicho nombre
00142         map<string,string>::iterator it;
00143         bool salir =false;
00144         for (it=itlow; it!=itupper && !salir;++it){
00145             if (it->second==tlf){
00146                 datos.erase(it);
00147                 salir =true;
00148             }
00149         }
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         }
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         }
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     return aux;
00218 }
00219 }
00220 }
00221 }
00222 void modificarTlf(string nombre, string tlf){
00223     map<string,string>::iterator it=datos.find(nombre);
00224     if(it != datos.end()){
00225         pair<string,string> p(it->first,tlf);
00226         datos.erase(it);
00227         datos.insert(p);
00228     }
00229 }
00230
00231 Guia_Tlf TlfsPorLetra(char letra){
00232     Guia_Tlf tlfs;
00233     int contador = 1; //Para enumerar los telefonos del map
00234     map<string,string>::iterator it;
00235     for(it = datos.begin(); it != datos.end(); ++it)
00236         if(letra == (*it).first[0])
00237             tlfs.insert(to_string(contador++) + ".", (*it).second);
00238
00239     return tlfs;
00240 }
00241
00242 Guia_Tlf TlfsEntreNombres(const string &nombre_1, const string &nombre_2){
00243     Guia_Tlf tlfs;
00244     map<string,string>::iterator itlow = datos.find(nombre_1);
00245     map<string,string>::iterator itupper = datos.find(nombre_2);
00246
00247     int contador = 1,
00248         dis1 = distance(datos.begin(),itlow),
00249         dis2 = distance(datos.begin(), itupper);
00250     if(itlow != datos.end() && dis1 <= dis2){
00251         tlfs.insert(to_string(contador++) + ".",itlow->second);
00252         while(dis1 < dis2){
00253             itlow++; dis1++;
00254             tlfs.insert(to_string(contador++) + ".",itlow->second);
00255         }
00256     }
00257     return tlfs;
00258 }
00259
00260 friend ostream & operator<<(ostream & os, Guia_Tlf & g){
00261     map<string,string>::iterator it;
00262     for (it=g.datos.begin(); it!=g.datos.end();++it){
00263         os<<it->first<<"\t"<<it->second<<endl;
00264     }
00265     return os;
00266 }
00267
00268 friend istream & operator>>(istream & is, Guia_Tlf & g){
00269     pair<string,string> p;
00270     Guia_Tlf aux;
00271
00272     while (is>>p){
00273         aux.insert(p);
00274     }
00275     g=aux;
00276     return is;
00277 }
00278
00279 class iterator{
00280 private:
00281     map<string,string>::iterator it;
00282 public:
00283     iterator & operator++(){
00284         ++it;
00285         return *this;
00286     }
00287     iterator & operator--(){
00288         --it;
00289         return *this;
00290     }
00291     pair<const string,string> &operator *(){
00292         return *it;
00293     }
00294     bool operator ==(const iterator &i){
00295         return i.it==it;
00296     }
00297
00298     bool operator !=(const iterator &i){
00299         return i.it!=it;
00300     }
00301 }
```

```

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 quieras obtener el telefono[Escribe 'quit' o 'q' para no
00025     buscar]"<<endl;
00026     string n;
00027     bool seguirleyendo = true;
00028
00029     getline(cin,n);
00030     if(n == "quit" || n == "q")
00031         seguirleyendo = false;
00032     //Busca nombre en la guia para obtener su tlf
00033     while (seguirleyendo){
00034         cout<<"Buscando "=>>n"...."=>>endl;
00035         string tlf = g.gettelefono(n);
00036         if (tlf=="")
00037             cout<<"No existe ese nombre en la guia"=>>endl;
00038         else
00039             cout<<"El telefono es "<<tlf<<endl;
00040
00041         cout<<"[Escribe 'quit' o 'q' para salir del bucle] Dime un nombre sobre el que quieras obtener el
00042         telefono"=>>endl;
00043         getline(cin,n);
00044         if(n == "quit" || n == "q")
00045             seguirleyendo = false;
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 tlf0 = g.gettelefono(n);
00059         if (tlf0=="")
00060             cout<<"No existe ese nombre en la guia"=>>endl;
00061         else{
00062             g.modificarTlf(n, tlf);
00063             cout<<"El telefono original era "<<tlf0" 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
00066      modificarlo:<endl;
00067      getline(cin,n);
00068      if(n == "quit" || n == "q")
00069          seguirleyendo = false;
00070      else
00071          getline(cin,tlf);
00072      }
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
00077      letra:[Escribe 'q' para no buscar]" << endl;
00078      cin.clear();
00079      cin >> letra;
00080      if(letra == 'q')
00081          seguirleyendo = false;
00082      while (seguirleyendo){
00083          cout<<"Buscando tlfs cuyos nombres asociados empiezan por '" << letra << "' ...."<<endl;
00084          entreLetras = g.TlfsPorLetra(letra);
00085          if (entreLetras.size()==0)
00086              cout<<"No existe numeros cuyos nombres empiecen por '" << letra << "'<< endl;
00087          else
00088              cout << entreLetras << endl;
00089          cout<<"[Escribe 'q' para salir]Dime una letra para listar numeros cuyos nombres empiecen por esa
00090          letra:<endl;
00091          cin >> letra;
00092          if(letra == 'q')
00093              seguirleyendo = false;
00094      }
00095      //Saca telefonos entre dos nombres
00096      string nombre1,nombre2;
00097      cout << "Dime entre que nombres quieres sacar la guia: " << endl;
00098      cin.ignore(); //Ignoramos el \n del cin anterior
00099      getline(cin,nombre1);
00100      getline(cin,nombre2);
00101      Guia_Tlf entreNombres = g.TlfsEntreNombres(nombre1,nombre2);
00102      cout << "Los telefonos entre " << nombre1 << " y " << nombre2 << " es: " << endl << entreNombres << endl;
00103      //Lista la guia final
00104      cout<<"Listando la guia resultante con iteradores:"<<endl;
00105      Guia_Tlf::iterator it;
00106      for (it=g.begin(); it!=g.end(); ++it)
00107          cout<<*it<<endl;
00108 }
```

Index

~Diccionario
 Diccionario< T, U >, 10

AddSignificado_Palabra
 Diccionario< T, U >, 10

begin
 Diccionario< T, U >, 11
 Guia_Tlf, 20

borrar
 Guia_Tlf, 20, 21

clave
 data< T, U >, 7

clear
 Guia_Tlf, 21

contabiliza
 Guia_Tlf, 22

data< T, U >, 7
 clave, 7
 info_asoci, 7

Diccionario
 Diccionario< T, U >, 9
 Diccionario< T, U >::iterator, 32

Diccionario< T, U >, 8
 ~Diccionario, 10
 AddSignificado_Palabra, 10
 begin, 11
 Diccionario, 9
 end, 11, 12
 Esta_Clave, 12
 getElementosEntreClaves, 13
 getInfo_Asoc, 14
 Insertar, 15
 operator+, 15
 operator=, 16
 removePalabra, 17
 removeSignificados_Palabra, 17
 size, 18

Diccionario< T, U >::iterator, 30
 Diccionario, 32
 operator!=, 31
 operator*, 31
 operator++, 31
 operator--, 31
 operator==, 32

diccionario.h
 operator<, 36

diccionario/include/diccionario.h, 35, 37

diccionario/src/usodiccionario.cpp, 40, 45

end
 Diccionario< T, U >, 11, 12
 Guia_Tlf, 22

EscribeSigni
 usodiccionario.cpp, 41

Esta_Clave
 Diccionario< T, U >, 12

getElementosEntreClaves
 Diccionario< T, U >, 13

getInfo_Asoc
 Diccionario< T, U >, 14

gettelefono
 Guia_Tlf, 22

Guia_Tlf, 19
 begin, 20
 borrar, 20, 21
 clear, 21
 contabiliza, 22
 end, 22
 gettelefono, 22
 Guia_Tlf::iterator, 34
 insert, 23
 modificarTlf, 24
 operator<<, 29
 operator>>, 29
 operator+, 24
 operator-, 25
 operator[], 26
 previos, 26
 size, 27
 TlfFsEntreNombres, 27
 TlfFsPorLetra, 28

Guia_Tlf::iterator, 32
 Guia_Tlf, 34
 operator!=, 33
 operator*, 33
 operator++, 33
 operator--, 33
 operator==, 34

guiatlf.h
 operator<<, 47
 operator>>, 48

guiatlf/include/guiatlf.h, 46, 48

guiatlf/src/usoguia.cpp, 51

info_asoci
 data< T, U >, 7

insert
 Guia_Tlf, 23
Insertar
 Diccionario< T, U >, 15

main
 usodiccionario.cpp, 41
modificarTlf
 Guia_Tlf, 24

operator!=
 Diccionario< T, U >::iterator, 31
 Guia_Tlf::iterator, 33
operator<
 diccionario.h, 36
operator<<
 Guia_Tlf, 29
 guiatlf.h, 47
 usodiccionario.cpp, 42
operator>>
 Guia_Tlf, 29
 guiatlf.h, 48
 usodiccionario.cpp, 43
operator*
 Diccionario< T, U >::iterator, 31
 Guia_Tlf::iterator, 33
operator+
 Diccionario< T, U >, 15
 Guia_Tlf, 24
operator++
 Diccionario< T, U >::iterator, 31
 Guia_Tlf::iterator, 33
operator-
 Guia_Tlf, 25
operator--
 Diccionario< T, U >::iterator, 31
 Guia_Tlf::iterator, 33
operator=
 Diccionario< T, U >, 16
operator==
 Diccionario< T, U >::iterator, 32
 Guia_Tlf::iterator, 34
operator[]
 Guia_Tlf, 26

previos
 Guia_Tlf, 26

removePalabra
 Diccionario< T, U >, 17
removeSignificados_Palabra
 Diccionario< T, U >, 17

separador
 usodiccionario.cpp, 44
size
 Diccionario< T, U >, 18
 Guia_Tlf, 27

TelfsEntreNombres