STL:Guia Telefonica

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 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.

4.1.1 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 42 of file guiatlf.h.

4.1.2 Member Function Documentation

4.1.2.1 begin()

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

Inicializa un iterator al comienzo de la guia.

Definition at line 338 of file guiatlf.h.

```
00338                                {
00339             iterator i;
00340             i.it=datos.begin();
00341             return i;
00342             }
```

4.1.2.2 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 119 of file guiatlf.h.

```
00119                                              {
00120                map<string,string>::iterator itlow = datos.lower_bound(nombre);//el primero que tiene
       dicho nombre
00121                map<string,string>::iterator itupper = datos.upper_bound(nombre);//el primero que ya no
       tiene dicho nombre
00122                datos.erase(itlow,itupper);//borramos todos aquellos que tiene dicho nombre
00123                //OTRA ALTERNATIVA
00124                //pair<map<string,string>::iterator,map<string,string>::iterator>ret;
00125                //ret = datos.equal_range(nombre
00126                //datos.erase(ret.first,ret.second);
00127             }
```

4.1.2.3 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 135 of file guiatlf.h.

```
00135                                                                   {
00136                     map<string,string>::iterator itlow = datos.lower_bound(nombre);//el primero que
      tiene dicho nombre
00137                     map<string,string>::iterator itupper = datos.upper_bound(nombre);//el primero que ya no
      tiene dicho nombre
00138                     map<string,string>::iterator it;
00139                     bool salir =false;
00140                     for (it=itlow; it!=itupper && !salir;++it){
00141                           if (it->second==tlf){
00142                          datos.erase(it);
00143                          salir =true;
00144                           }
00145                     }
00146
00147             }
```

4.1.2.4 clear()

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

Limpia la guia.

Definition at line 169 of file guiatlf.h.

```
00169                           {
00170                     datos.clear();
00171             }
```

4.1.2.5 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 162 of file guiatlf.h.

```
00162                                                   {
00163                     return datos.count(nombre);
00164             }
```

4.1.2.6 end()

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

Inicializa un iterator al final de la guia.

Definition at line 346 of file guiatlf.h.

```
00346                           {
00347             iterator i;
00348             i.it=datos.end();
00349             return i;
00350             }
```

4.1.2.7 gettelefono()

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

Definition at line 75 of file guiatlf.h.

```
00075                                                   {
00076                   map<string,string>::iterator it=datos.find(nombre);
00077                   if (it==datos.end())
00078                     return string("");
00079                   else return it->second;
00080             }
```

4.1.2.8 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 105 of file guiatlf.h.

```
00105                                                                       {
00106
00107                   pair<map<string,string> ::iterator,bool> ret;
00108
00109                   ret=datos.insert(p); //datos.insert(datos.begin(),p); tambien funcionaría
00110                   return ret;
00111
00112             }
```

4.1.2.9 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 89 of file guiatlf.h.

```
00089                                                                                   {
00090                   pair<string,string> p (nombre,tlf);
00091                   pair< map<string,string> ::iterator,bool> ret;
00092
00093                   ret=datos.insert(p); //datos.insert(datos.begin(),p); tambien funcionaría
00094                   return ret;
00095
00096           }
```

4.1.2.10 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 226 of file guiatlf.h.

```
00226                                                   {
00227                   map<string,string>::iterator it=datos.find(nombre);
00228                   if(it != datos.end()){
00229                      pair<string,string> p(it->first,tlf);
00230                      datos.erase(it);
00231                      datos.insert(p);
00232                   }
00233           }
```

4.1.2.11 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 177 of file guiatlf.h.

```
00177                                                {
00178             Guia_Tlf aux(*this);
00179             map<string,string>::const_iterator it;
00180             for (it=g.datos.begin();it!=g.datos.end();++it){
00181                aux.insert(it->first,it->second);
00182             }
00183             return aux;
00184
00185             }
```

Here is the call graph for this function:

4.1.2.12 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 192 of file guiatlf.h.

```
00192                                                {
00193             Guia_Tlf aux(*this);
00194             map<string,string>::const_iterator it;
00195             for (it=g.datos.begin();it!=g.datos.end();++it){
00196                aux.borrar(it->first,it->second);
00197             }
00198             return aux;
00199
00200             }
```

Here is the call graph for this function:

4.1.2.13 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 70 of file guiatlf.h.

```
00070                                                {
00071                  return datos[nombre];
00072             }
```

4.1.2.14 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 207 of file guiatlf.h.

```
00207                                                                    {
00208             map<string,string>::value_compare vc=datos.value_comp();
00209             //map<string,string>::key_compare vc=datos.key_comp();
00210             Guia_Tlf aux;
00211             pair<string,string>p(nombre,tlf);
00212             map<string,string>::iterator it=datos.begin();
00213             while (vc(*it,p)){
00214                   aux.insert(*it++);
00215
00216             }
00217             return aux;
00218
00219              }
```

Here is the call graph for this function:

4.1.2.15 size()

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

Numero de telefonos.

Returns

el numero de telefonos asociados

Definition at line 152 of file guiatlf.h.

```
00152                          {
00153                   return datos.size();
00154             }
```

4.1.2.16 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 257 of file guiatlf.h.

```
00257                                                                                                {
00258                 Guia_Tlf tlfs;
00259                 map<string,string>::iterator itlow = datos.find(nombre_1);
00260                 map<string,string>::iterator itupper = datos.find(nombre_2);
00261
00262                 int contador = 1,
00263                     dis1 = distance(datos.begin(),itlow),
00264                     dis2 = distance(datos.begin(), itupper);
00265                 if(itlow != datos.end() && dis1 <= dis2){
00266                   tlfs.insert(to_string(contador++) + ".",itlow->second);
00267                   while(dis1 < dis2){
00268                      itlow++; dis1++;
00269                      tlfs.insert(to_string(contador++) + ".",itlow->second);
00270                   }
00271                 }
00272                 return tlfs;
00273             }
```

Here is the call graph for this function:

4.1.2.17 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

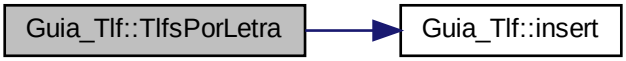
Returns

tlfs Una guia con los telefonos encontrados

Definition at line 240 of file guiatlf.h.

```
00240                                           {
00241                 Guia_Tlf tlfs;
00242                 int contador = 1; //Para enumerar los telefonos del map
00243                 map<string,string>::iterator it;
00244                 for(it = datos.begin(); it != datos.end(); ++it)
00245                     if(letra == (*it).first[0])
00246                         tlfs.insert(to_string(contador++) + ".",(*it).second);
00247
00248                 return tlfs;
00249             }
```

Here is the call graph for this function:



4.1.3 Friends And Related Function Documentation

4.1.3.1 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 281 of file guiatlf.h.

```
00281                                                           {
00282                   map<string,string>::iterator it;
00283                   for (it=g.datos.begin(); it!=g.datos.end();++it){
00284                     os<<it->first<<"\t"<<it->second<<endl;
00285                   }
00286                   return os;
00287             }
```

4.1.3.2 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 296 of file guiatlf.h.

```
00296                                                                    {
00297                   pair<string,string> p;
00298                   Guia_Tlf aux;
00299
00300                   while (is»p){
00301                   aux.insert(p);
00302                   }
00303                   g=aux;
00304                   return is;
00305           }
```

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

- include/guiatlf.h

4.2 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

4.2.1 Detailed Description

clase para iterar sobre la guia

Definition at line 310 of file guiatlf.h.

4.2.2 Member Function Documentation

4.2.2.1 operator"!=()

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

Definition at line 329 of file guiatlf.h.

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

4.2.2.2 operator*()

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

Definition at line 322 of file guiatlf.h.

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

4.2.2.3 operator++()

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

Definition at line 314 of file guiatlf.h.

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

4.2.2.4 operator--()

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

Definition at line 318 of file guiatlf.h.

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

4.2.2.5 operator==()

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

Definition at line 325 of file guiatlf.h.

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

4.2.3 Friends And Related Function Documentation

4.2.3.1 Guia_Tlf

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

Definition at line 332 of file guiatlf.h.

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

- include/guiatlf.h

Chapter 5

File Documentation

5.1 guiatlf.h

```
00001
00006 #ifndef _GUIA_TLF_H
00007 #define _GUIA_TLF_H
00008 #include <map>
00009 #include <iostream>
00010 #include <string>
00011 using namespace std;
00012 istream & operator»(istream &is,pair<string,string> &d){
00013
00014       getline(is,d.first,'\t');
00015       getline(is,d.second);
00016       return is;
00017 }
00018
00019 ostream & operator«(ostream &os,const pair<const string,string> &d){
00020
00021       os«d.first«'\t'«d.second«endl;
00022       return os;
00023 }
00024
00025
00042 class Guia_Tlf{
00057       private:
00058             map<string,string> datos; //si admites que haya nombres repetidos tendrías que usar un
00059                         //multimap
00060
00061       public:
00062
00068             //si fuese un multimap no podriamos usar []. Ademas que deberiamos devolver p.e un vector
       con todos
00069             // los telefonos asociados a dicho nombre
00070             string & operator[](const string &nombre) {
00071                   return datos[nombre];
00072             }
00073
00074
00075             string  gettelefono(const string & nombre){
00076                   map<string,string>::iterator it=datos.find(nombre);
00077                   if (it==datos.end())
00078                     return string("");
00079                   else return it->second;
00080             }
00081
00089             pair<map<string,string>::iterator,bool>  insert(string nombre, string tlf){
00090                   pair<string,string> p (nombre,tlf);
00091                   pair< map<string,string> ::iterator,bool> ret;
00092
00093                   ret=datos.insert(p); //datos.insert(datos.begin(),p); tambien funcionaría
00094                   return ret;
00095
00096             }
00097
00105             pair<map<string,string>::iterator,bool>  insert(pair<string,string> p){
00106
00107                   pair<map<string,string> ::iterator,bool> ret;
00108
00109                   ret=datos.insert(p); //datos.insert(datos.begin(),p); tambien funcionaría
00110                   return ret;
```

```
00111
00112            }
00113
00119            void borrar(const string &nombre){
00120               map<string,string>::iterator itlow = datos.lower_bound(nombre);//el primero que tiene
     dicho nombre
00121               map<string,string>::iterator itupper = datos.upper_bound(nombre);//el primero que ya no
     tiene dicho nombre
00122               datos.erase(itlow,itupper);//borramos todos aquellos que tiene dicho nombre
00123               //OTRA ALTERNATIVA
00124               //pair<map<string,string>::iterator,map<string,string>::iterator>ret;
00125               //ret = datos.equal_range(nombre
00126               //datos.erase(ret.first,ret.second);
00127            }
00128
00134            //con map siempre hay uno con multimap puede existir mas de uno
00135            void borrar(const string &nombre,const string &tlf){
00136                  map<string,string>::iterator itlow = datos.lower_bound(nombre);//el primero que
     tiene dicho nombre
00137               map<string,string>::iterator itupper = datos.upper_bound(nombre);//el primero que ya no
     tiene dicho nombre
00138               map<string,string>::iterator it;
00139               bool salir =false;
00140               for (it=itlow; it!=itupper && !salir;++it){
00141                     if (it->second==tlf){
00142                    datos.erase(it);
00143                    salir =true;
00144                     }
00145               }
00146
00147            }
00152            int size()const{
00153                  return datos.size();
00154            }
00161            //al ser un map debe de ser 0 o 1. Si fuese un multimap podríamos tener mas de uno
00162            unsigned int contabiliza(const string &nombre){
00163                  return datos.count(nombre);
00164            }
00165
00169            void clear(){
00170                  datos.clear();
00171            }
00177            Guia_Tlf operator+(const Guia_Tlf & g){
00178            Guia_Tlf aux(*this);
00179            map<string,string>::const_iterator it;
00180            for (it=g.datos.begin();it!=g.datos.end();++it){
00181               aux.insert(it->first,it->second);
00182            }
00183            return aux;
00184
00185            }
00186
00192            Guia_Tlf operator-(const Guia_Tlf & g){
00193            Guia_Tlf aux(*this);
00194            map<string,string>::const_iterator it;
00195            for (it=g.datos.begin();it!=g.datos.end();++it){
00196               aux.borrar(it->first,it->second);
00197            }
00198            return aux;
00199
00200            }
00207            Guia_Tlf previos(const string &nombre,const string &tlf){
00208            map<string,string>::value_compare vc=datos.value_comp();
00209            //map<string,string>::key_compare vc=datos.key_comp();
00210            Guia_Tlf aux;
00211            pair<string,string>p(nombre,tlf);
00212            map<string,string>::iterator it=datos.begin();
00213            while (vc(*it,p)){
00214                  aux.insert(*it++);
00215
00216            }
00217            return aux;
00218
00219             }
00220
00226            void modificarTlf(string nombre, string tlf){
00227                  map<string,string>::iterator it=datos.find(nombre);
00228                  if(it != datos.end()){
00229                     pair<string,string> p(it->first,tlf);
00230                     datos.erase(it);
00231                     datos.insert(p);
00232                  }
00233            }
00234
00240            Guia_Tlf TlfsPorLetra(char letra){
00241                Guia_Tlf tlfs;
00242                int contador = 1; //Para enumerar los telefonos del map
```

```
00243                 map<string,string>::iterator it;
00244                 for(it = datos.begin(); it != datos.end(); ++it)
00245                     if(letra == (*it).first[0])
00246                         tlfs.insert(to_string(contador++) + ".",(*it).second);
00247 
00248                 return tlfs;
00249             }
00250 
00257             Guia_Tlf TelfsEntreNombres(const string &nombre_1, const string &nombre_2){
00258                 Guia_Tlf tlfs;
00259                 map<string,string>::iterator itlow = datos.find(nombre_1);
00260                 map<string,string>::iterator itupper = datos.find(nombre_2);
00261 
00262                 int contador = 1,
00263                     dis1 = distance(datos.begin(),itlow),
00264                     dis2 = distance(datos.begin(), itupper);
00265                 if(itlow != datos.end() && dis1 <= dis2){
00266                   tlfs.insert(to_string(contador++) + ".",itlow->second);
00267                   while(dis1 < dis2){
00268                      itlow++; dis1++;
00269                      tlfs.insert(to_string(contador++) + ".",itlow->second);
00270                   }
00271                 }
00272                 return tlfs;
00273             }
00274 
00281             friend ostream & operator<<(ostream & os, Guia_Tlf & g){
00282                   map<string,string>::iterator it;
00283                   for (it=g.datos.begin(); it!=g.datos.end();++it){
00284                     os<<it->first<<"\t"<<it->second<<endl;
00285                   }
00286                   return os;
00287             }
00288 
00296             friend istream & operator>>(istream & is, Guia_Tlf & g){
00297                   pair<string,string> p;
00298                   Guia_Tlf aux;
00299 
00300                   while (is>>p){
00301                   aux.insert(p);
00302                   }
00303                   g=aux;
00304                   return is;
00305             }
00306 
00310             class iterator{
00311              private:
00312               map<string,string>::iterator it;
00313              public:
00314                iterator & operator++(){
00315                 ++it;
00316                 return *this;
00317                }
00318                iterator & operator--(){
00319                --it;
00320                return *this;
00321                }
00322                pair<const string,string> &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 Guia_Tlf;
00333             };
00334 
00338             iterator  begin(){
00339             iterator i;
00340             i.it=datos.begin();
00341             return i;
00342             }
00346             iterator  end(){
00347             iterator i;
00348             i.it=datos.end();
00349             return i;
00350             }
00351 
00352 
00353 
00354 
00355 };
00356 #endif
00357 
```

5.2 usoguia.cpp

```
00001 #include "guiatlf.h"
00002 #include <fstream>
00003 int main(int argc , char * argv[]){
00004    if (argc!=2){
00005       cout«"Dime el nombre del fichero con la guia"«endl;
00006    return 0;
00007    }
00008    ifstream f(argv[1]);
00009    if (!f){
00010       cout«"No puedo abrir el fichero "«argv[1]«endl;
00011       return 0;
00012    }
00013    Guia_Tlf g;
00014 
00015    f»g;
00016    cout«"La guia insertada: " « endl « g«endl;
00017    cin.clear();
00018    cout«"Dime un nombre sobre el que quieres obtener el telefono[Escribe 'quit' o 'q' para no
       buscar]"«endl;
00019    string n;
00020    bool seguirleyendo = true;
00021 
00022    getline(cin,n);
00023    if(n == "quit" || n == "q")
00024       seguirleyendo = false;
00025    //Busca nombre en la guia para obtener su tlf
00026    while (seguirleyendo){
00027       cout«"Buscando "«n«"...."«endl;
00028       string tlf = g.gettelefono(n);
00029       if (tlf=="")
00030          cout«"No existe ese nombre en la guia"«endl;
00031       else
00032          cout«"El telefono es "«tlf«endl;
00033 
00034       cout«"[Escribe 'quit' o 'q' para salir del bucle] Dime un nombre sobre el que quieres obtener el
       telefono"«endl;
00035       getline(cin,n);
00036       if(n == "quit" || n == "q")
00037          seguirleyendo = false;
00038    }
00039    seguirleyendo = true;
00040 
00041    //Modifica el tlf de un contacto
00042    string tlf;
00043    cout«"Dime un nombre y su telefono para modificarlo[Escribe 'quit' o 'q' para no modificar]:"«endl;
00044    getline(cin,n);
00045    if(n == "quit" || n == "q")
00046       seguirleyendo = false;
00047    else
00048       getline(cin,tlf);
00049 
00050    while (seguirleyendo){
00051       cout«"Modificando tlf de "«n«"...."«endl;
00052       string tlfO = g.gettelefono(n);
00053       if (tlfO=="")
00054          cout«"No existe ese nombre en la guia"«endl;
00055       else{
00056          g.modificarTlf(n, tlf);
00057          cout«"El telefono original era "«tlfO« " y ahora es " « g.gettelefono(n) « endl;
00058       }
00059       cout«"[Escribe 'quit' o 'q' para dejar de modificar]Dime un nombre y su telefono para
       modificarlo:"«endl;
00060       getline(cin,n);
00061       if(n == "quit" || n == "q")
00062          seguirleyendo = false;
00063       else
00064          getline(cin,tlf);
00065    }
00066    seguirleyendo = true;
00067    //Saca telefonos de nombres que empiezen por una letra
00068    Guia_Tlf entreLetras;
00069    char letra;
00070    cout « "Introduce la letra por la que buscar telefonos cuyos nombres empiecen por esa
       letra:[Escribe 'q' para no buscar]" « endl;
00071    cin.clear();
00072    cin » letra;
00073    if(letra == 'q')
00074       seguirleyendo = false;
00075    while (seguirleyendo){
00076       cout«"Buscando tlfs cuyos nombres asociados empiezan por '" « letra «"' ...."«endl;
00077       entreLetras = g.TlfsPorLetra(letra);
00078       if (entreLetras.size()==0)
00079          cout«"No existe numeros cuyos nombres empiecen por '" « letra « "'" « endl;
00080       else
00081             cout « entreLetras « endl;
```

```
00082       cout«"[Escribe 'q' para salir]Dime una letra para listar numeros cuyos nombres empiecen por esa
      letra:"«endl;
00083       cin » letra;
00084       if(letra == 'q')
00085          seguirleyendo = false;
00086    }
00087    //Saca telefonos entre dos nombres
00088    string nombre1,nombre2;
00089    cout « "Dime entre que nombres quieres quieres sacar la guia: " « endl;
00090    cin.ignore(); //Ignoramos el \n del cin anterior
00091    getline(cin,nombre1);
00092    getline(cin,nombre2);
00093    Guia_Tlf entreNombres = g.TelfsEntreNombres(nombre1,nombre2);
00094    cout « "Los telefonos entre " « nombre1 « " y " « nombre2 « " es: " « endl « entreNombres « endl;
00095
00096    //Lista la guia final
00097    cout«"Listando la guia resultante con iteradores:"«endl;
00098    Guia_Tlf::iterator it;
00099    for (it=g.begin(); it!=g.end(); ++it)
00100       cout«*it«endl;
00101
00102 }
```

Index