

OBJETOS FUNCION (FUNCTORS)

- * MUY importantes en la STL
- * Los functors son objetos de una clase que tienen sobre cargado el operador parentesis

```
class Sumador {
    int valor;
public:
    Sumador (int s = 0) : valor(s) {}
    void operator() (int & x) const
        { x += valor; }
};
```

Uso:

```
Sumador s1(5), s2(10);
int var = 1;
s1(var); // suma 5 a var
s2(var); // suma 10 a var
```

La clase sumador es un tipo y por tanto puede ser un parametro de una funcion plantilla

```
template <class T>
class Mayor {
public:
    bool operator() (const T& x, const T& y)
        { return x > y }
};
```

Mezclar (Vd.begin(), Vd.end(), Mayor());

se le pasa a la función mezclar una función que implementa un mecanismo de comparación de valores

```
template <class T> // template <typename T>
inline const T& max (const T& a, const T& b)
{ return a < b ? b : a; }
```

```
#include <iostream>
#include <algorithm>
using namespace std;

int main()
{
    cout << max(4, 4.5) << endl; // MAL
}

int main()
{
    int a; double b;
    ...
    cout << max(static_cast<double>(a), b)
        << endl; // BIEN
}

int main()
{
    cout << max<double>(4, 4.5) << endl;
} // BIEN
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <...>

using namespace std;

int main() {
    vector<int> vec;
    vec.push_back(2); vec.push_back(5);
    vec.push_back(1); vec.push_back(4); vec.push_back(3);
    cout << "Desordenado ";
    for (int i = 0; i < vec.size(); ++i) {
        cout << vec[i] << ' ';
    }
    cout << endl;
    // Ordena el vector en orden ascendente usando
    // el functor less
    sort(vec.begin(), vec.end(), less<int>());
}
```

```
cout << "Ordenado ";
for (int i = 0; i < vec.size(); ++i) {
    cout << vec[i] << ' ';
}
cout << endl;
}
```

TDA Conjunto (set)

Un conjunto es un contenedor de valores únicos, llamados claves o miembros del conjunto, que se almacenan de manera ordenada. El TDA Conjunto está basado en el concepto matemático de conjunto.

Ejemplos:

- Un corrector ortográfico de un procesador de textos puede utilizar un conjunto para almacenar todas las palabras que considera correctas ortográficamente hablando.

En la STL, el TDA Conjunto recibe el nombre de *set*.

Especificación del TDA Conjunto (set) - I

```
/**  
set<T,Comp>
```

```
TDA set:: set, empty, clear, size,  
count, find, lower_bound,  
upper_bound, insert, erase,  
begin, end, swap, ~set
```

Cada objeto del TDA Conjunto, modela un conjunto de elementos de la clase T.

Un conjunto es un contenedor que almacena elementos de manera ordenada que no están repetidos.

Son objetos mutables.
Residen en memoria dinámica.

```
*/
```

Especificación del TDA Conjunto (set) - II

```
/**  
 * @brief Constructor primitivo.  
 * @doc Crea un conjunto vacío.  
 */  
set();  
  
/**  
 * @brief Constructor de copia.  
 * @param c: conjunto que se copia.  
 * @doc Crea un nuevo conjunto que es copia de c.  
 */  
set(const set<T> & c);  
  
/**  
 * @brief Constructor de copia de un rango.  
 * @param inicio: apunta al elemento inicial a copiar.  
 * @param final: apunta al elemento final a copiar.  
 * @doc Crea un nuevo conjunto que es copia del rango  
 * [inicio,final].  
 */  
set(iterator inicio, iterator final);
```

Especificación del TDA Conjunto (set) - III

```
/**  
 * @brief Informa si el conjunto está vacío.  
 * @return true, si el conjunto está vacío.  
 *          false, en otro caso.
```

```
*/
```

```
bool empty() const;
```

```
/**
```

```
 * @brief Borra todos los elementos del contenedor.  
 * @doc Deja el contenedor completamente vacío.  
 * Eficiencia: lineal.
```

```
*/
```

```
void clear();
```

```
/**
```

```
 * @brief Obtiene el número de elementos.  
 * @return Número de elementos del conjunto.
```

```
*/
```

```
size_type size() const;
```

Especificación del TDA Conjunto (set) - IV

```
/**  
 * @brief Busca un elemento en el conjunto.  
 * @param clave: elemento a buscar.  
 * @return 1 si el elemento está en el conjunto y  
 *         0 en caso contrario.  
 * @doc Determina si un elemento está o no en el  
 *      conjunto receptor. Eficiencia: Lineal.  
 */  
int count(const T & clave) const;
```

```
/**  
 * @brief Busca un elemento en el conjunto.  
 * @param clave: elemento a buscar.  
 * @return Un iterador que apunta al elemento dentro  
 *         del conjunto en caso de que se encuentre  
 *         en él, o end() en caso contrario.  
 *         Eficiencia: Logarítmica.  
 */  
iterator find(const T & clave);
```

Especificación del TDA Conjunto (set) - VI

```
/**
```

@brief Busca un elemento en el conjunto.

@param clave: elemento a buscar.

@return Un iterador constante que apunta al elemento dentro del conjunto en caso de que se encuentre en él, o end() en caso contrario.

Eficiencia: logarítmica.

```
*/
```

```
const_iterator find(const T & clave);
```

```
/**
```

@brief Inserta un elemento en el conjunto.

@param clave: elemento a insertar.

@return Un objeto de la clase pair instanciada con un iterador y un valor booleano. En caso de que el elemento ya existiera en el conjunto, el iterador señalará al elemento en el conjunto y false en el valor bool. Si el elemento no estaba en el conjunto el iterador igualmente señalará al elemento insertado y true en el valor bool.

@doc Inserta un elemento en el conjunto si no estaba ya incluído en él.

Eficiencia: logarítmica.

```
*/
```

```
pair<iterator,bool> insert(const T & clave);
```

Especificación del TDA Conjunto (set)- VII

```
/**  
 * @brief Borra un elemento del conjunto.  
 * @param clave: elemento a eliminar.  
 * @return 1 si el elemento estaba en el conjunto y  
 *         ha sido eliminado, 0 en caso contrario.  
 * @doc Al eliminarse el elemento, el tamaño se  
 *      decrementa en una unidad.  
 * Eficiencia: logarítmica.  
 */  
int erase(const T & clave);
```

```
/**  
 * @brief Borra un elemento del conjunto.  
 * @param pos: señala al elemento a borrar.  
 * @pre El conjunto no está vacío.  
 * @doc Elimina el elemento y el tamaño del  
 *      conjunto se decrementa en una unidad.  
 */  
void erase(iterator pos);
```

Especificación del TDA Conjunto (set)-VIII

```
/**  
 * @brief Borra elementos del rango [primero,ultimo].  
 * @param primero: primer elemento del rango.  
 *         ultimo: último elemento del rango.  
 * @pre El conjunto no está vacío.  
 * @doc Elimina los elementos incluidos en el rango  
 *      pasado como argumento. El tamaño del  
 *      conjunto se decrementa según los  
 *      elementos del rango.  
 */
```

```
void erase(iterator primero, iterator ultimo);
```

```
/**  
 * @brief Devuelve un iterador señalando al primer  
 *        elemento del conjunto.  
 * @return Un iterador apuntando al primer elemento.  
 */  
iterator begin();
```

```
/**  
 * @brief Devuelve un iterador constante señalando al  
 *        primer elemento del conjunto.  
 * @return Un iterador constante apuntando al primer  
 *        elemento.  
 */  
const_iterator begin() const;
```

Especificación del TDA Conjunto (set)- IX

```
/**  
 * @brief Devuelve un iterador señalando al último  
 *        elemento del conjunto.  
 * @return Un iterador apuntando al último elemento.  
 */  
iterator end();  
  
/**  
 * @brief Devuelve un iterador constante señalando al  
 *        último elemento del conjunto.  
 * @return Un iterador constante apuntando al último  
 *        elemento.  
 */  
const_iterator end() const;  
  
/**  
 * @brief Intercambia el contenido del receptor y  
 *        del argumento.  
 * @param s: pila a intercambiar con el receptor.  
 *          ES MODIFICADO.  
 * @doc Este método asigna el contenido del  
 *      receptor al del parámetro y el del  
 *      parámetro al del receptor.  
 */  
void swap (set<T> & s);
```

Especificación del TDA Conjunto (set) - X

```
/**  
 * @brief Destructor.  
 * @post El receptor es MODIFICADO.  
 * @doc El receptor es destruido liberando todos los  
 *      recursos que usaba.  
 */  
~set();
```

Otras operaciones: ==, !=, <, >, <=, >=, =.

Ejemplo de uso del TDA Conjunto (set) - I

```
#include <set>
using namespace std;

set<int> conjuntoEnteros; // Ordenación creciente

set<double, greater<double>> conjuntoReales;

set<long, less<long>> conjuntoMiTipo;

struct ltstr
{
    bool operator()(const char* s1,
                     const char* s2) const
    {
        return strcmp(s1, s2) < 0;
    }
};

set <char *, ltstr> conjuntoChar;
```

Ejemplo de uso del TDA Conjunto - II

```
#include <iostream>
#include <set>
using namespace std;

int main()
{
    typedef set<int,greater<int> > IntSet;
    IntSet coll1; // Conjunto vacío de enteros.

    // Inserción de elementos ‘‘aleatoriamente’’.
    coll1.insert(4);
    coll1.insert(3);
    coll1.insert(5);
    coll1.insert(1);
    coll1.insert(6);
    coll1.insert(2);
    coll1.insert(5);

    // Iteramos sobre todos los elementos.
    IntSet::iterator pos;
    for (pos = coll1.begin(); pos != coll1.end(); ++pos)
        cout << *pos << ' ';
    cout << endl;
```

```
// Insertamos el 4 otra vez y procesamos el valor
// que se devuelve.

pair<IntSet::iterator,bool> status = coll1.insert(4);
if (status.second) {
    cout << "4 insertado. "
        << distance(coll1.begin(),status.first) + 1
        << endl;
}
else {
    cout << "4 ya existe." << endl;
}

// Creación de otro conjunto y asignación de los
// elementos del primero de ellos.

set<int> coll2(coll1.begin(),coll1.end());

// Borrado de los elementos desde el principio hasta
// el tres.

coll2.erase (coll2.begin(), coll2.find(3));

// Eliminación del elemento 5.

int num;
num = coll2.erase (5);
cout << num << " elemento(s) eliminados." << endl;
}
```

Usando la clase set de la STL, construir una función para determinar si un conjunto tiene todos los elementos pares incluidos dentro de otro.

```
bool inclusionpares (const set<int> &S,  
const set<int> &Z)  
{  
    set<int>::iterator p;  
    for (p = S.begin(); p != S.end(); ++p)  
        if (*p % 2 == 0)  
            if (!Z.find(*p)) return false;  
        else ++p;  
    return true;  
}
```

Usando la clase set de la STL, construir una función que divida un conjunto de enteros c en dos subconjuntos par y cimpar que contienen respectivamente los elementos pares e impares de c y que devuelva true si el número de elementos de cpar es mayor que el de cimpar y falso en caso contrario.

bool masparqueimpares (vector<int> & c,
set<int> & cpar,
set<int> & cimpar)

```
    set<int> :: iterator p;  
    for (p = c.begin(); p != c.end(); ++p)  
        if (*p % 2 == 0)  
            cpar.insert (*p);  
        else  
            cimpar.insert (*p);  
    if (cpar.size() > cimpar.size())  
        return true;  
    else return false;  
}
```

// return ((cpar.size()) >(cimpar.size()))

Usando la clase set de la STL, construir una función para determinar si un conjunto tiene más de la mitad de sus elementos comunes con otro.

bool masdelamitadcomunes (const set<int>&(1),
const set<int>&(2))

{
 set<int>::const_iterator p;
 int n1, n2;
 n1 = 0; n2 = (d. size());
 for (p = (d. begin()); p != (d. end()); ++p)
 if ((2. count (*p))
 n1++;
 if (n1 > n2 / 2) return true;
 else return false;

5

Usando la clase set de la STL, diseñar una función que dados 2 conjuntos C_1 y C_2 determine el conjunto de los elementos de C_1 que no están en C_2 y todos los de C_2 que no están en C_1 .

```
set<int> no_comunes (set<int> C1,  
                      set<int> C2)
```

```
{  
    set<int>::iterator p, q;  
    set<int> solucion;  
    for (p = C1.begin(); p != C1.end(); ++p)  
        if (!C2.count(*p))  
            solucion.insert(*p);  
    for (q = C2.begin(); q != C2.end(); ++q)  
        if (!C1.count(*q))  
            solucion.insert(*q);  
    return solucion;
```

5

Usando la clase set de la STL, diseñar una función que determine la intersección de dos conjuntos C1 y C2.

Void comunas (set <int> c1, set <int> c2,
set <int> & resultado)

```
    set <int>:: iterator p, q;  
    for (p = c1.begin(); p != c1.end(); ++p)  
        if (c2.count (*p))  
            resultado.insert (*p);
```



TDA Bolsa (multiset)

Una bolsa es un caso especial de conjunto en el que se permite almacenar valores repetidos.

Ejemplos:

-  Aplicaciones en que interese controlar las repeticiones de los valores
- 

En la STL, el TDA Bolsa recibe el nombre de *multiset*.

Especificación del TDA Bolsa (multiset) - I

```
/**
```

```
multiset<T,Comp>
```

```
TDA multiset::multiset, empty, clear, size,  
count, find, lower_bound,  
upper_bound, find, equal_range,  
insert, erase, begin, end,  
swap, ~multiset
```

Cada objeto del TDA Bolsa, modela un conjunto de elementos de la clase T.

Una bolsa es un contenedor que almacena elementos de manera ordenada que pueden estar repetidos.

Son objetos mutables.

Residen en memoria dinámica.

```
*/
```

Esp. del TDA Bolsa (multiset) - II

```
/**  
 * @brief Constructor primitivo.  
 * @doc Crea una bolsa vacía.  
 */  
 multiset();  
  
/**  
 * @brief Constructor de copia.  
 * @param b: bolsa que se copia.  
 * @doc Crea una nueva bolsa que es copia de b.  
 */  
 multiset(const multiset<T> & b);  
  
/**  
 * @brief Constructor de copia de un rango.  
 * @param inicio: apunta al elemento inicial a copiar.  
 * @param final: apunta al elemento final a copiar.  
 * @doc Crea un nuevo conjunto que es copia del rango  
 * [inicio,final].  
 */  
 multiset(iterator inicio, iterator final);
```

Esp. del TDA Bolsa (multiset) -III

```
/**
```

```
    @brief Informa si la bolsa está vacía.
```

```
    @return true, si la bolsa está vacía.
```

```
                false, en otro caso.
```

```
*/
```

```
bool empty() const;
```

```
/**
```

```
    @brief Borra todos los elementos del contenedor.
```

```
    @doc Deja el contenedor completamente vacío.
```

```
    Eficiencia: lineal.
```

```
*/
```

```
void clear();
```

```
/**
```

```
    @brief Obtiene el número de elementos.
```

```
    @return Número de elementos de la bolsa.
```

```
*/
```

```
size_type size() const;
```

Esp. del TDA Bolsa (multiset) -IV

```
/**
```

```
 @brief Busca un elemento en la bolsa.
```

```
 @param clave: elemento a buscar.
```

```
 @return número de ocurrencias del elemento.
```

```
 @doc Determina si un elemento está o no en la  
 bolsa receptora y devuelve cuántas veces.
```

```
Eficiencia: lineal.
```

```
*/
```

```
int count(const T & clave) const;
```

```
/**
```

```
 @brief Busca un elemento en la bolsa.
```

```
 @param clave: elemento a buscar.
```

```
 @return Un iterador que apunta al primer elemento  
 dentro de la bolsa que coincide con  
 la clave o end() en caso de  
 que no exista.
```

```
Eficiencia: Logarítmica.
```

```
*/
```

```
iterator find(const T & clave);
```

Esp. del TDA Bolsa (multiset) -IV

/**

@brief Determina la primera posición donde se insertaría un elemento.

@para clave: clave cuya ubicación se desea encontrar
@return Un iterador que señala a la primera posición donde una copia del argumento se insertaría de acuerdo con el criterio de ordenación.

@doc Según el criterio de inserción que se use, determina la primera posición donde se insertaría una clave. Eficiencia: logarítmica.

*/

iterator lower_bound (const T & clave);

/**

@brief Determina la última posición donde se insertaría un elemento.

@para clave: clave cuya ubicación se desea encontrar
@return Un iterador que señala a la ~~última~~ posición donde una copia del argumento se insertaría de acuerdo con el criterio de ordenación.

@doc Según el criterio de inserción que se use, determina la última posición donde se insertaría una clave.

Eficiencia: logarítmica.

*/

iterator upper_bound (const T & clave);

Esp. del TDA Bolsa (multiset) -V

```
/**  
 * @brief Busca un elemento en la bolsa.  
 * @param clave: elemento a buscar.  
 * @return Un iterador constante que apunta al primer  
 *         elemento dentro de la bolsa que  
 *         coincida con la clave o end() en caso de  
 *         que no exista.  
 * Eficiencia: logarítmica.  
 */  
const_iterator find(const T & clave);  
  
/**  
 * @brief Busca un elemento en la bolsa.  
 * @param clave: elemento a buscar.  
 * @return Un objeto de la clase pair, conteniendo un  
 *         iterador al primer elemento que coincide  
 *         con la clave y otro al último. En caso de  
 *         que no exista los dos serán end().  
 * Eficiencia: logarítmica.  
 */  
pair<iterator,iterator> equal_range(const T & clave);
```

Esp. del TDA Bolsa (multiset) -V

```
/**  
 * @brief Inserta un elemento en la bolsa.  
 * @param clave: elemento a insertar.  
 * @return iterador dentro del multiset apuntando al  
 *         elemento recién insertado.  
 * @doc Inserta un elemento en la bolsa.  
 *      Eficiencia: logarítmica.  
 */  
iterator insert(const T & clave);
```

```
/**  
 * @brief Borra todas las ocurrencias del argumento  
 *        en la bolsa.  
 * @param clave: elemento a eliminar.  
 * @return Número de elementos eliminados.  
 * @doc Al eliminarse el elemento, el tamaño se  
 *      decrementa en tantos elementos como se  
 *      hayan eliminado.  
 *      Eficiencia: logarítmica.  
 */  
int erase(const T & clave);
```

Esp. del TDA Bolsa (multiset) -VI

```
/**  
 * @brief Borra elementos del rango [primero,ultimo].  
 * @param primero: primer elemento del rango.  
 *         ultimo: último elemento del rango.  
 * @pre El bolsa no está vacío.  
 * @doc Elimina los elementos incluidos en la rango  
 *      pasado como argumento. El tamaño de la  
 *      bolsa se decrementa según los elementos  
 *      del rango.  
 */  
void erase(iterator primero, iterator ultimo);  
  
/**  
 * @brief Devuelve un iterador señalando al primer  
 *        elemento de la bolsa.  
 * @return Un iterador apuntando al primer elemento.  
 */  
iterator begin();  
  
/**  
 * @brief Devuelve un iterador constante señalando al  
 *        primer elemento del bolsa.  
 * @return Un iterador constante apuntando al primer  
 *        elemento.  
 */  
const_iterator begin() const;
```

Esp. del TDA Bolsa (multiset)-VII

```
/**  
 * @brief Devuelve un iterador señalando al último  
 * elemento del bolsa.  
 * @return Un iterador apuntando al último elemento.  
 */  
iterator end();  
  
/**  
 * @brief Devuelve un iterador constante señalando al  
 * último elemento de la bolsa.  
 * @return Un iterador constante apuntando al último  
 * elemento.  
 */  
const_iterator end() const;  
  
/**  
 * @brief Intercambia el contenido del receptor y  
 * del argumento.  
 * @param c: bolsa a intercambiar con el receptor.  
 * ES MODIFICADO.  
 * @doc Este método asigna el contenido del  
 * receptor al del parámetro y el del  
 * parámetro al del receptor.  
 */  
void swap (multiset<T> & c);
```

Esp. del TDA Bolsa (multiset)-VIII

```
/**  
 * @brief Destructor.  
 * @post El receptor es MODIFICADO.
```

El receptor es destruido liberando todos los recursos que usaba. Eficiencia: lineal.

```
*/  
~multiset();
```

Otras operaciones: ==, !=, <, >, <=, >=, =.

Ejemplo de uso del TDA Bolsa (multiset)-I

```
#include <multiset>
```

```
using namespace std;
```

```
multiset<int> bolsaEnteros;
```

```
multiset<char, greater<char>> bolsaCaracteres;
```

```
multiset<long, less<long>> bolsaLong;
```

```
struct ltstr
{
    bool operator()(const char* s1,
                     const char* s2) const
    {
        return strcmp(s1, s2) < 0;
    }
};
```

```
multiset <char *, ltstr> bolsaChar;
```

Ejemplo de uso del TDA Bolsa - II

```
int main()
{
    typedef multiset<int,greater<int> > IntSet;
    IntSet coll1;
    coll1.insert(4);
    coll1.insert(3);
    coll1.insert(5);
    coll1.insert(1);
    coll1.insert(6);
    coll1.insert(2);
    coll1.insert(5);

    IntSet::iterator pos;
    for (pos = coll1.begin(); pos != coll1.end(); ++pos)
        cout << *pos << ' ';
    cout << endl;

    // Se inserta 4 de nuevo, procesando el valor devuelto
    IntSet::iterator ipos = coll1.insert(4);
    cout << "4 se inserta como elemento "
        << distance(coll1.begin(),ipos) + 1
        << endl;
```

TDA Diccionario (map)

Un diccionario es un contenedor que almacena elementos (valores) identificados mediante una clave. No existen ~~pares~~ (clave,valor) ~~repetidos con la clave repetida~~

Análogamente, al multiset para el set, existe el multimap para el map, permitiendo elementos con ~~claves-repetidas mas de un valor asociado a una clave~~

Ejemplos:

- El diccionario de la RAE contiene parejas (palabra,definición).
- El stock de un almacén: para cada artículo, el número de unidades que existe.
- Un listín telefónico: para cada abonado, su número de teléfono.

En la STL, el TDA Diccionario recibe el nombre de *map*.

Especificación del TDA Diccionario (map)-I

```
/**
```

```
map<K, T, Comp>
```

```
TDA map::map, empty, clear, size, count,  
find, equal_range, erase, insert,  
lower_bound, upper_bound, [],  
begin, end, swap, ~map
```

Cada objeto del TDA Bolsa, modela un conjunto de elementos de la clase T.

Un diccionario es un contenedor que almacena elementos de manera ordenada, según sus correspondientes claves.

Cada par (clave, valor) es único.

Son objetos mutables.

Residen en memoria dinámica.

```
*/
```

Esp. del TDA Diccionario (map) - II

```
/**  
 * @brief Constructor primitivo.  
 * @doc Crea un diccionario vacío.  
 */  
map();  
  
/**  
 * @brief Constructor de copia.  
 * @param d: Diccionario que se copia.  
 * @doc Crea un nuevo diccionario que es copia de d.  
 */  
map(const map<key_type, T> & b);  
  
/**  
 * @brief Constructor de copia de un rango.  
 * @param inicio: apunta al elemento inicial a copiar.  
 * @param final: apunta al elemento final a copiar.  
 * @doc Crea un nuevo conjunto que es copia del rango  
 * [inicio,final].  
 */  
map(iterator inicio, iterator final);
```

Esp. del TDA Diccionario (map) - III

```
/**  
 * @brief Informa si el diccionario está vacío.  
 * @return true, si el diccionario está vacío;  
 *         false, en otro caso.  
 */  
bool empty() const;  
  
/**  
 * @brief Borra todos los elementos del contenedor.  
 * @doc Deja el contenedor completamente vacío.  
 * Eficiencia: lineal.  
 */  
void clear();  
  
/**  
 * @brief Devuelve el número de elementos del  
 * diccionario.  
 * @return Número de elementos (pares clave-valor) del  
 * diccionario.  
 */  
size_type size() const;
```

Esp. del TDA Diccionario (map) - IV

```
/**  
 * @brief Busca la clave en el diccionario.  
 * @param clave: identificador a buscar.  
 * @return Número de elementos con esa clave.  
 * @doc Determina si hay algún elemento en el  
 * diccionario con esa clave y devuelve  
 * el número de ellos.  
 * Eficiencia: lineal.  
 */  
int count(const key_type & clave) const;  
  
/**  
 * @brief Busca elementos con una clave dada.  
 * @param clave: identificador de los elementos.  
 * @return Un iterador al primer elemento dentro  
 * del diccionario que coincida con la  
 * clave o end() en caso contrario.  
 * Eficiencia: logarítmica.  
 */  
iterator find(const key_type & clave);
```

Esp. del TDA Diccionario (map) - V

```
/**
```

@brief Busca elementos con una clave concreta.

@param clave: identificador de los elementos.

@return Un iterador constante que apunta al primer elemento del diccionario cuya clave coincida con el argumento o end() en caso de que no exista.

Eficiencia: logarítmica.

```
*/
```

```
const_iterator find(const key_type & clave);
```

```
/**
```

@brief Busca elementos en el diccionario.

@param clave: identificador de los elemento a buscar

@return Un objeto de la clase pair, conteniendo un iterador al primer elemento que coincide con la clave y otro al último. En caso de que no exista los dos serán end().

Eficiencia: logarítmica.

```
*/
```

```
pair<iterator,iterator> equal_range  
(const key_range & clave);
```

Esp. del TDA Diccionario (map) - VI

```
/**
```

```
 @brief Borra el elemento del diccionario cuya  
 clave coincida con la del argumento argumento  
 @param clave: identificador del elemento a eliminar.  
 @return 1 si el elemento estaba en el conjunto y  
 ha sido eliminado, 0 en caso contrario.  
 @doc Al eliminarse el elemento, el tamaño se  
 decrementa en una unidad.  
 Eficiencia: logarítmica.
```

```
 */
```

```
 int erase(const key_type & clave);
```

```
/**
```

```
 @brief Borra un elemento del diccionario.  
 @param pos: señala al elemento a borrar.  
 @pre El diccionario no está vacío.  
 @doc Elimina el elemento y el tamaño del  
 diccionario se decrementa en una unidad.
```

```
 */
```

```
 void erase(iterator pos);
```

Esp. del TDA Diccionario (map) - VII

```
/**
```

@brief Borra elementos del rango [primero,ultimo].

@param primero: primer elemento del rango.

ultimo: último elemento del rango.

@pre El diccionario no está vacío.

@doc Elimina los elementos incluidos en el rango pasado como argumento. El tamaño del diccionario se decremente según los elementos del rango.

```
*/
```

```
void erase(iterator primero, iterator último);
```

```
/**
```

@brief Inserta un elemento con su clave en el diccionario si no existe.

@param par: Objeto de la clase pair conteniendo la clave y el elemento a insertar.

@return Un objeto de la clase pair instanciada con un iterador y un valor booleano. Si la clave existe el iterador señala al elemento con dicha clave y false en el valor bool. Si no existe, el iterador señalará al elemento recién insertado, y true en la parte bool.
Eficiencia: logarítmica.

```
*/
```

```
pair<iterator, bool> insert(const value_type& Par)
```

Esp. del TDA Diccionario (map) - IX

```
/**  
 * @brief Inserta un elemento con clave k al diccionario en caso de que no exista. Si existe, sustituye el valor existente por el nuevo.  
 * @param k: Clave del elemento.  
 * @return El valor contenido en el diccionario cuya clave es la pasada como argumento.  
 * Eficiencia: logarítmica.  
 */
```

```
T& operator[](const key_type& k)
```

```
/**  
 * @brief Devuelve un iterador señalando al primer elemento del diccionario.  
 * @return Un iterador apuntando al primer elemento.  
 */  
iterator begin();
```

```
/**  
 * @brief Devuelve un iterador constante señalando al primer elemento del diccionario.  
 * @return Un iterador constante apuntando al primer elemento.  
 */  
const_iterator begin() const;
```

Esp. del TDA Diccionario (map) - X

```
/**  
 * @brief Devuelve un iterador señalando al último  
 * elemento del diccionario.  
 * @return Un iterador apuntando al último elemento.  
 */  
iterator end();  
  
/**  
 * @brief Devuelve un iterador constante señalando al  
 * último elemento del diccionario.  
 * @return Un iterador constante apuntando al último  
 * elemento.  
 */  
const_iterator end() const;  
  
/**  
 * @brief Intercambia el contenido del receptor y  
 * del argumento.  
 * @param m: diccionario a intercambiar con el receptor.  
 * ES MODIFICADO.  
 * @doc Este método asigna el contenido del  
 * receptor al del parámetro y el del  
 * parámetro al del receptor.  
 */  
void swap (map<K,T> & m);
```

Esp. del TDA Diccionario (map) - XI

```
/**  
 * @brief Destructor.  
 * @post El receptor es MODIFICADO.
```

El receptor es destruido liberando todos los recursos que usaba.

Eficiencia: lineal.

```
*/  
~map();
```

Otras operaciones: ==, !=, <, >, <=, >=, =.

Ej. de uso del TDA Diccionario (map) - I

```
#include <map>
using namespace std;
```

```
map<String,double> diccStringReal;
```

```
map<int,String> diccEntCad;
```

```
map<int,less<double>> diccEntReal;
```

```
struct ltstr
{
    bool operator()(const char* s1,
                     const char* s2) const
    {
        return strcmp(s1, s2) < 0;
    }
};
```

```
map<char *, int, ltstr> diccCharEnt;
```

Ej. del uso del TDA Diccionario (map) - II

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main()
{
    typedef map<string,float> StringFloatMap;

    StringFloatMap acciones;

    // Inserción de elementos.
    acciones["BASF"] = 369.50;
    acciones["VW"] = 413.50;
    acciones["Daimler"] = 819.00;
    acciones["BMW"] = 834.00;
    acciones["Siemens"] = 842.20;

    // Impresión
    StringFloatMap::iterator pos;
    for (pos = acciones.begin(); pos != acciones.end();
         ++pos)
        cout << "acción: " << pos->first << "\t"
            << "precio: " << pos->second << endl;

    cout << endl;
```

```
// Se doblan los precios de la acción
for (pos = acciones.begin(); pos != acciones.end();
     ++pos)
    pos->second *= 2;
}
```

construir una clase guia_de_teléfonos que de
soporte al manejo de información del tipo:

Susana Amiba 958665544

Antonio Santillan 653453111

Carlos Pineda 606112233

Carolina Fuentes 958334455

Fernando Ruiz 654234567

: -- = : (sin nombres repetidos)

Con funciones para:

- Devolver un tipo asociado a un nombre
- Insertar un nuevo elemento en la guia
- borrar un elemento de la guia
- Imprimir la guia

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

```
class Alia_Tlf {
```

```
private:
```

```
map<string, string> datos;
```

```
public:
```

```
string & operator[](const string & nombre)
{
    return datos[nombre];
}
```

```
pair<map<string, string>::iterator, bool>
insert(pair<string, string> p)
```

```
{
    pair<map<string, string>::iterator, bool> ret;
    ret = datos.insert(p);
    return ret;
}
```

```
void borrow (const string & number)
```

```
}
```

```
map<string, string>::iterator it_low =
```

```
dados.lower_bound(number),
```

```
map<string, string>::iterator it_upper =
```

```
dados.upper_bound(number),
```

```
dados.erase(it_low, it_upper);
```

```
}
```

```
friend ostream & operator << (ostream & os,
```

```
Guia_Tip & g)
```

```
{
```

```
map<string, string>::iterator it;
```

```
for (it = g.dados.begin();
```

```
it != g.dados.end(); ++it) {
```

```
os << it->first << "\t" <<
```

```
it->second << endl;
```

```
}
```

```
return os;
```

```
}
```

class iterator {

private:

map<string, string>::iterator it;

public:

iterator& operator++() {
 ++it;
}

iterator& operator--() {
 --it;
}

pair<const string, string>& operator*() {
 return *it;
}

bool operator != (const iterator& i) {
 return i.it != it;

bool operator == (const iterator& i) {
 return i.it == it;

friend class Guia_Tlf;

};

```
iterator begin() {  
    iterator i;  
    i.it = datos.begin();  
    return i;  
}
```

```
iterator end() {  
    iterator i;  
    i.it = datos.end();  
    return i;  
}
```

Se quiere construir el tipo de dato vectorDisperso de string, que se caracteriza porque la mayoría de los elementos toman el mismo valor (que llamaremos valor por defecto). Para representar dicho valor es más eficiente almacenar solo los elementos distintos por lo que se propone la siguiente representación

```
class VectorDisperso {
```

```
private:
```

```
map<int, string> M; //Map que almacena los  
string v_def; //valor por defecto
```

```
};
```

donde, para un vectorDisperso v , el elemento de la posición i -ésima del vector, $v[i]$ sería:

$$v[i] = \begin{cases} M[i] & \text{si } M.\text{find}(i) \neq M.\text{end}() \\ v_def & \text{en otro caso} \end{cases}$$

Implementar un método que cambie el valor por defecto del vector (teniendo en cuenta los elementos del vector disperso cuyo valor anterior fuese nr)

void VectorDisperso::cambiar_valor_defecto (const string & nr)

```
void vectorDisperso::cambiar_valor_defecto (const
{
    map<int, string>::iterator it;
    for (it = M.begin(); it != M.end(); )
    {
        if (M[it] == nv) // (it).second == nv
            M.erase(it);
        else it++;
    }
    v_def = nv;
}
```

Para gestionar un documento, se usa un TDA Documento. Este TDA tiene en su representación una tabla Hash en la que cada palabra del documento tiene asociada una lista ordenada con las posiciones en las que aparece la palabra en el mismo.

Implementar una función

`int Documento::min_distancia (string pal1,
string pal2)`

que devuelva la distancia mínima en la que aparecen las palabras `pal1` y `pal2` en el documento. Para la representación de la tabla Hash se usa hashing abierto

Usamos el TDS map

class Documento {

std::map<string, list<int>> tabla_hash;

} ! ↑
unordered_map //Tabla hash STL

int Documento::min_distanza (string p1, string p2)

{

list<int> l1 = tabla_hash[p1];

list<int> l2 = tabla_hash[p2];

int minima = numeric_limits<int>::max();

for (list<int>::iterator it1 = l1.begin();
it1 != l1.end(); ++it1)

for (list<int>::iterator it2 = l2.begin();

it2 != l2.end(); ++it2)

{ int d = abs(*it1 - *it2);

if (d < minima) minima = d;

}

return minima;

}

Vectores de vectores: matrices

vector <vector<T>>

una matriz es un vector de "filas" donde cada "fila" es un vector de elementos. La diferencia de los arrays bidimensionales de C++ no existe obligación de que todas las filas tengan el mismo número de elementos: cada vector fila es independiente de los demás y la única restricción es que todos contengan elementos del mismo tipo

Crear una matriz de n filas por m columnas no es más que crear un vector de n filas, donde cada uno de los elementos de ese vector es una fila de m columnas

Usar vector <vector<T>> es algo menos eficiente que usar arrays al guardar las filas por separado y tener que realizar varios accesos de memoria para acceder a una posición (i,j).

A cambio las matrices son más versátiles que los arrays.

Vectores de vectores: matrices

typedef vector<int> Fila;

typedef vector<Fila> Matriz;

int main()

Matriz M(5); //Matriz de 5 filas vacias

for (int i = 0; i < M.size(); ++i)

M[i].resize(3); //Matriz 5x3

Matriz M2(5, Fila(3)) //otra forma de tener
//una matriz 5x3

for (int i = 0; i < M.size(); ++i)

for (int j = 0; j < M[i].size(); ++j)

M[i][j] = M2[i][j] = i + j;

//se rellenan las matrices

}

typedef vector<int> File;

typedef vector<File> Matrix;

Matrix products (const Matrix & ma,
const Matrix & mb)

{

int nfa = ma.size(),
nca = ma[0].size(),
ncb = mb[0].size();

Matrix m (nfa, File(ncb));

for (int i = 0; i < nfa; ++i)

 for (int j = 0; j < ncb; ++j)

 for (int k = 0; k < nca; ++k)

 m[i][j] += ma[i][k] * mb[k][j];

return m;

}

//ma y mb rectangulares y el número
// de columnas de ma es igual al número
// de filas de mb