map<clave, datos, compara>

Descripción

Un diccionario (map) es un contenedor que permite asociar objetos del tipo clave con los objetos de tipo datos, almacenándolos de forma ordenada. Por tanto, el tipo de información que manipula son pair<clave,datos>. Al igual que set, también un contendor asociativo único, esto es, no hay dos elementos en el map que tengan la misma clave

En un map al insertar un nuevo elemento no invalida los iteradores que apuntan a los objetos existentes. De igual forma, el borrar un elemento del par tampoco invalida ningún iterator, excepto, por supuesto, para los iterators que señalan realmente al elemento que se está borrando.

Ejemplo

```
int main()

{

  map<string, int> meses;


  meses["january"] = 31;

  meses["february"] = 28;

  meses["march"] = 31;

  meses["april"] = 30;

  meses["may"] = 31;

  meses["june"] = 30;

  meses["july"] = 31;

  meses["august"] = 31;

  meses["september"] = 30;

  meses["october"] = 31;

  meses["november"] = 30;

  meses["december"] = 31;


  cout << "junio -> " << meses["june"] << endl;

  map<string,int>::iterator cur  = meses.find("june");
```

```
  map<string,int>::iterator prev = cur;

  map<string,int>::iterator next = cur;

  ++next;

  --prev;

  cout << "Anterior  (en orden alfabetico) es " << (*prev).first << endl;

  cout << "Siguiente (en orden alfabetico) is " << (*next).first << endl;


  for (cur = meses.begin(); cur!= meses.end(); ++cur)

      cout << Meses “<< cur->first << “Dias “<< cur->dias << endl;

}
```

Definición

Definido en el fichero cabecera <map> y el fichero <map.h> por compatibilidad con versiones anteriores

Parámetros de la plantilla

Parámetro	Descripción
`Clave`	El tipo dominante del map.
`Datos`	El tipo de datos del map.
`Comparar`	La función de comparación, que impone un orden parcial estricto sobre los elementos de la clave

Requerimientos del tipo

- `Los datos tienen el operador de asignación`
- `Comparar` permite definir un orden parcial estricto sobre los objetos de tipo clave

Miembros y tipos

Tipos		Descripción
`size_type`		Un entero sin signo
`iterator`		Iterator que itera a través de un `map`. Si it es de tipo iterador, (*it) es del tipo pair<clave,datos> Además en el map se cumple que (*it).first no es mutable, mientras que (*it).second si lo es.
`const_iterator`		Un iterador constante sobre map. Si it es de tipo iterador, (*it) es del tipo pair<clave,datos>
`reverse_iterator`		Iterator que itera en orden inverso a través de un `map`. Si it es de tipo iterador, (*it) es del tipo pair<clave,datos> Si it es de tipo iterador, (*it) es del tipo pair<clave,datos>. Además en el map se cumple que (*it).first no es mutable, mientras que (*it).second si lo

		es.
const_reverse_iterator		Un iterador constante en orden inverso sobre map. Si it es de tipo iterador, (*it) es del tipo pair<clave,datos>
Miembros	**Eficiencia**	**Descripción**
iterator begin()	O(1)	Devuelve un iterator que señala al principio del map.
iterator end()	O(1)	Devuelve un iterator que señala al final del map. (siguiente al ultimo elemento)
const_iterator begin() const	O(1)	Devuelve un const_iterator que señala al principio del map.
const_iterator end() const	O(1)	Devuelve un const_iterator que señala al extremo del map, (siguiente al ultimo elemento).
reverse_iterator rbegin()	O(1)	Devuelve un reverse_iterator que señala al principio del map, en orden inverso.
reverse_iterator rend()	O(1)	Devuelve un reverse_iterator que señala al

		final del map, en orden inverso..
`const_reverse_iterator rbegin() const`	O(1)	Devuelve un `const_reverse_iterator` que señala al principio del map, en orden inverso.
`const_reverse_iterator rend() const`	O(1)	Devuelve un `const_reverse_iterator` que señala al extremo del map, en orden inverso.
`size_type size() const`	O(1)	Devuelve el tamaño del `map`.
`size_type max_size() const`	O(1)	Devuelve el tamaño más grande posible del `map`.
`bool empty() const`	O(1)	`verdad` si el tamaño del `map` es `0`.
`map()`	O(1)	Crea un `map` vacío.
`map(const map&)`	O(n)	El constructor de copia.
`map& operator=(const map&)`	O(n)	El operador de asignación
`void swap(map&)`	O(1)	Intercambiar el contenido de dos maps.
`pair<iterator, bool> insert(const pair<clave,datos>& x)`	O(log n)	Inserta el elemento `x` en el map. Si el elemento no se encuentra en el map,

		inserta el elemento. El segundo campo del par que devuelve toma el valor true si se ha podido realizar la inserción con éxito. En caso contrario, el segundo campo del par toma el valor falso. En cualquier caso, devuelve en el primer campo del par la posicion del elemento dentro del map. (S.find(x)!=S.end()) S.size() se incrementa en 1
`iterator insert(iterator pos, const pair<clave,datos>& x)`	O(log n)	Inserta `x` en el `set`, usando la `posición` pos como indicativo donde pudiera ser insertada. Devuelve la posicion donde se encuentra el elemento tras la inserción.
`void erase(iterator pos)`	O(log n)	Borra el elemento señalado por a la `posición.` Pos debe apuntar a una posición válida dentro del set, pos !=S.end()
`size_type erase(const clave& k)`	O(log n)	Borra el elemento cuya clave es `K.` Devuelve el numero de elementos borrados

		(en el caso del map es 0 o 1)
`void erase(iterator first, iterator last)`	O(log n)	Borra todos los elementos del map dentro de un rango definido por [first,last)
`void clear()`	O(1)	Borra todos los elementos.
`iterator find(const clave& k)`	O(log n)	Encuentra un elemento cuya clave es `K`. Devuelve un iterador que apunta a la posición donde se encuentra el elemente Si el elemento no se encuentra en el set devuelve end(), i.e. S.find(k)==S.end()
`const_iterator find(const clave& k) const`	O(log n)	Encuentra un elemento cuya clave es `K`. Devuelve un iterador constante que apunta a la posición donde se encuentra el elemente Si el elemento no se encuentra en el set devuelve end(), i.e. S.find(k)==S.end()
`size_type count(const clave& k)`	O(log n)	Cuenta el número de elementos cuya llave es `K`.
`iterator lower_bound(const clave& k)`	O(log n)	Encuentra el primer elemento cuya clave sea mayor o igual

		que K.
const_iterator lower_bound(const clave& k) const	O(log n)	Encuentra el primer elemento cuya clave sea mayor o igual que K.
iterator upper_bound(const clave& k)	O(log n)	Encuentra el primer elemento cuya clave es mayor que K.
const_iterator upper_bound(const clave& k) const	O(log n)	Encuentra el primer elemento cuya clave es mayor que K.
pair<iterator, iterator> equal_range(const clave& k)	O(log n)	Encuentra todos los elementos cuya clave sea K.
pair<const_iterator, const_iterator> equal_range(const clave& k) const	O(log n)	Encuentra todos los elementos cuya clave sea K.
datos& operator[](const clave& k) [3]	O(log n)	Devuelve una referencia al objeto que se asocia a una clave particular, k. Si el map no contiene tal objeto, el operador [] inserta el objeto por defecto del tipo datos.. m [k] es equivalente a (*((m.insert (pair<clave,datos>(k, datos()))) .first)).second
bool operator==(const map&, const map&)	O(n)	Chequea dos map para la igualdad. Es una función global,

		no una función miembro.
`bool operator<(const map&, const map&)`	O(n)	Comparación lexicográfica. Es una función global, no una función miembro.