# set<clave, compara>

## Descripción

**set** es un contenedor que almacena objetos del tipo clave. Un **set** es un contenedor que permite ordenar los elementos en orden creciente según el valor de la clave. Importante, no está permitido que dentro del contenedor existan dos elementos iguales.

El **set** tiene la importante característica de que al insertar un nuevo elemento no se invalidan los iteradores (iterator) que señalan a los elementos existentes. Igualmente al borrar un elemento del **set** tampoco se invalidan los iterators, excepto aquellos iteradores que señalan realmente al elemento que se ha borrado

## Ejemplo 1

```
#include<set>
int main(){
  set<int> cjto;
  for (int i = 10; i>0; i++)
       cjto.insert(i);
  set<int>::iterator it;
  for (it = cjto.begin(); it!=cjto.end();++it)
         coout << *it;
}
```

## Ejemplo 2

```
#include <set>
#include <string>
#include <iostream>

using namespace std;

class ltstr
{ public:
  bool operator()(const string & s1, const string & s2) const;

};

bool ltstr::operator()(const string & s1, const string & s2) const
  {
    return (s1>s2);
  }


int main() {
    set<int> x;
    set<string,greater<string> > W;
    set<string,ltstr> S2;

    set<int,greater<int> > y;

    int a;
```

```
    for (int i = 0;i<10;i++)
        { x.insert(i);
        }


   for (int i = 0;i<10;i++)
        { y.insert(i);
        }

   set<int>::iterator it;
   set<int,greater<int> >::iterator it_aux;

    for (it = x.begin(); it!=x.end(); it++)
        { cout << *it << " "; }
    cout << endl;


    for (it_aux = y.begin(); it_aux!=y.end(); it_aux++)
        { cout << *it_aux << " "; }
    cout << endl;



    cout << y.max_size()<< endl;
    cout << (y.begin() == y.end()) << endl;
    cout << (x.begin() == x.end()) << endl;

    set<int>::iterator it1,it2;
    it1 = x.find(2);
    it2 = x.find(5);

    x.erase(it1,it2);

    cout << "tras borrar [2,5) " <<endl;
    for (it = x.begin(); it!=x.end(); it++)
        { cout << *it<< " "; }
    cout << endl;

    it1 = x.upper_bound(6);
    cout << *x.lower_bound(6)<<endl;
    cout << *it1 <<endl;

    it2 = x.insert(it1,-3);
    it2 = x.insert(it1,-3);
    cout << "inserto -3 " <<*it2 << endl;
     for (it = x.begin(); it!=x.end(); it++)
        { cout << *it<< " "; }
    cout << endl;

    cin >> a;
}
```

## Definición

Definido en el fichero cabecera <**set**> y <**set**.h> para compatibilidad con versiones anteriores.

## Parámetros de la plantilla

| Parámetro | Descripción |
|---|---|
| `clave` | El tipo sobre el que se construye el set. |
| `Comparar` | La función de comparación. Debe generar un orden parcial estricto sobre los elementos de la clav |

## Requerimientos

- `La clave tiene definido el operador de asignación`
- `Comparar define un orden parcial estricto sobre la clave.`

## Tipos y Miembros

| Tipos | | Descripción |
|---|---|---|
| `size_type` | | Entero sin signo |
| `iterator` | | Tipo utilizado para iterar sobre el set |
| `const_iterator` | | Iterador constante para iterar sobre el set. (`Iterator` y el `const_iterator` son el mismo tipo.) |
| `reverse_iterator` | | Tipo utilizado para iterar en orden inverso sobre el set |
| `const_reverse_iterator` | | Iterador constante para iterar en orden inverso sobre el set. (`Reverse_iterator` y el `const_reverse_iterator` son el mismo tipo.) |
| **Miembros** | **Eficiencia** | **Descripción** |
| `iterator begin() const` | O(1) | Devuelve un `iterator` que señala al primer elemento del set |

| | | |
|---|---|---|
| `iterator end() const` | O(1) | Devuelve un `iterator` que señala al final del set. (posición siguiente al último elemento). |
| `reverse_iterator rbegin() const` | O(1) | Devuelve un `reverse_iterator` que señala al principio del set en orden inverso. |
| `reverse_iterator rend() const` | O(1) | Devuelve un `reverse_iterator` que señala al final del set en orden inverso (posición siguiente al último en orden inverso). |
| `size_type size() const` | O(1) | Devuelve el tamaño del `set` |
| `size_type max_size() const` | O(1) | Devuelve el tamaño posible más grande del `set`. |
| `bool empty() const` | O(1) | `verdad` si el tamaño del `set` es 0.<br><br>si S.empty() entonces<br>( S.begin()==S.end() ) |
| `set()` | O(1) | Crea un `set` vacío. |
| `set(const set&)` | O(n) | El constructor de copia. |
| `set& operator=(const set&)` | O(n) | El operador de asignación |
| `void swap(set&)` | O(1) | Intercambia el contenido de dos sets.<br><br>El intercambio de dos sets, tambien implica el intercambio de los iteradores que les apuntan. |
| `pair<iterator, bool>`<br>`insert(const clave& x)` | O(log n) | Inserta el elemento `x` en el `set`.<br><br>Si el elemento no se encuentra en el set, 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.<br><br>En cualquier paso, devuelve en el primer campo del par la posicion del elemento dentro del set.<br><br>(S.find(x)!=S.end()) S.size() se incrementa en 1 |
| `iterator insert(iterator pos,const clave& x)` | O(log n) | Inserta `x` en el `set`, usando la `posición` pos como indicativo donde pudiera ser insertada.<br><br>Devuelve la posicion donde se encuentra el elemento tras la inserción. |
| `void erase(iterator pos)` | O(log n) | Borra el elemento que se encuentra en la posición pos. 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 set es 0 o 1) |
| `void erase(iterator first, iterator last)` | O(log n) | Borra todos los elementos del set dentro de un rango definido por [first,last) |
| `void clear()` | O(1) | Borra todos los elementos. |
| `iterator find(const clave& k) const` | O(log n) | Encuentra un elemento cuya clave es `K.`<br><br>Si el elemento no se encuentra en el set devuelve end(), i.e. S.find(k)==S.end() |
| `size_type count(const clave& k) const` | O(log n) | Cuenta el número de los elementos que clave es `K.(En el set` |

| | | |
|---|---|---|
| | | debe ser 0 o 1) |
| 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) const | O(log n) | Encuentra el primer elemento cuya clave es mayor que K. |
| pair<iterator, iterator> equal_range(const clave& k) const | O(log n) | Encuentra todos los elementos cuya clave sea K. |
| bool operator==(const set&, const set&) | O(n) | Chequea dos sets para la igualdad. Es una función global, no una función miembro. |
| bool operator<(const set&, const set&) | O(n) | Comparación lexicográfica. Es una función global, no una función miembro. |