

# Programación Orientada a Objetos

## Grado en Ingeniería Informática

### Biblioteca estándar de plantillas STL II

Departamento de Ingeniería Informática  
Universidad de Cádiz



ver 0.1

# Indice

- 1 Algoritmos
- 2 Contenedores asociativos
- 3 Bibliografía



# Sección 1 | Algoritmos



# Algoritmos

## Descripción

- En STL existen una gran cantidad de algoritmos que puede ser utilizados con los contenedores e iteradores.
- Existen algoritmos de ordenamiento, búsqueda, mezcla, matemáticos, etc...
- Estos algoritmos no son otra que cosa que funciones **template** que operan sobre los contenedores a través de los iteradores.



# Algoritmos

## Estructura general

La estructura general que poseen los algoritmos es la siguiente

```
template <class T>
T Max( vector <T> & v )
{
    // crear una variable para devolver el maximo
    T maximo = v[0];
    // buscar el maximo
    for( unsigned i=1; i<v.size(); ++i )
        if( maximo < v[i] )
            maximo = v[i];
    return maximo;
}
```

- Solución: Encontrar el valor máximo de un vector
- Problema: Solo funciona con **vectores**
- Opción: Buscar en parte del contenedor



# Algoritmos

## Ejemplo algo mejor

La estructura general que poseen los algoritmos es la siguiente

```

#include <iostream>
#include <vector>
using namespace std;
// crear un algoritmo generico para encontrar el maximo
template <class Iter>
Iter Max( Iter inicio, Iter fin )
{
    Iter maximo = inicio; // crear una variable para devolver el maximo
    Iter aux = inicio;
    while( aux != fin ) // buscar el maximo
    {
        if( (*maximo) < (*aux) )
            maximo = aux;
        ++aux;
    }
    return maximo;
}

int main()
{
    vector <int> datos;
    for( unsigned i=0; i<10; ++i ) // llenar el vector con enteros al azar
        datos.push_back( rand() % 10 );
    for( unsigned i=0; i<datos.size(); ++i ) // mostrar por pantalla
        cout << datos[i] << " ";
    cout << "\nMaximo: " << *( Max( datos.begin(), datos.end() ) ); // utilizar el algoritmo para encontrar el maximo
    return 0;
}

```



# Algoritmos

## Observaciones del ejemplo

### Mejoras obtenidas en este ejemplo

- El algoritmo funciona con cualquier tipo de contenedor que ofrezca iteradores
- Se puede cambiar el rango de búsqueda con cambiar las posiciones de los iteradores inicio - fin
- El iterador devuelto encapsula la información y la posición, teniendo por lo tanto como resultado el mayor elemento y su posición en la secuencia.

### En resumen:

- Los algoritmos están separado de los contenedores, lo que facilita **la escritura de algoritmos genéricos** que se apliquen a diversos contenedores.
- Por lo tanto es sencillo **añadir nuevos algoritmos** sin modificar los contenedores STL

# Algoritmos

## Ejemplo 02

```
#include <iostream>
#include <list>
#include <iterator>
#include <algorithm>
using namespace std;
int main()
{
    // generar una lista de valores al azar utilizando "generate"
    list<int> valores(10);
    generate( valores.begin(), valores.end(), rand );

    // mostrar los elementos por pantalla utilizando "copy"
    ostream_iterator<int> salida( cout, " " );
    copy( valores.begin(), valores.end(), salida );

    // buscar la primer aparicion del numero 5 utilizando "find"
    list<int>::iterator p;
    p = find( valores.begin(), valores.end(), 5 );

    // analizar si el iterador quedo en el rango de busqueda
    if( p != valores.end() )
        cout << "nEl valor " << *p << " esta en la lista\n";
    else
        cout << "nNo se encontro el valor buscado\n";
    return 0;
}
```



UCA

Universidad



# Algoritmos

## Explicación: Ejemplo 02

- **generate** Lo hemos utilizado para rellenar una lista de elementos al azar
- **copy** Para copiar los datos de salida a pantalla
- **find** Buscamos un número con esta función, si no nos devuelve *end()* es que se encontró el elemento dentro del rango.



# Algoritmos

## 01: No modificadores

### Non-modifying sequence operations:

<b>all_of</b> <small>C++11</small>	Test condition on all elements in range (function template )
<b>any_of</b> <small>C++11</small>	Test if any element in range fulfills condition (function template )
<b>none_of</b> <small>C++11</small>	Test if no elements fulfill condition (function template )
<b>for_each</b>	Apply function to range (function template )
<b>find</b>	Find value in range (function template )
<b>find_if</b>	Find element in range (function template )
<b>find_if_not</b> <small>C++11</small>	Find element in range (negative condition) (function template )
<b>find_end</b>	Find last subsequence in range (function template )
<b>find_first_of</b>	Find element from set in range (function template )
<b>adjacent_find</b>	Find equal adjacent elements in range (function template )
<b>count</b>	Count appearances of value in range (function template )
<b>count_if</b>	Return number of elements in range satisfying condition (function template )
<b>mismatch</b>	Return first position where two ranges differ (function template )
<b>equal</b>	Test whether the elements in two ranges are equal (function template )
<b>is_permutation</b> <small>C++11</small>	Test whether range is permutation of another (function template )
<b>search</b>	Search range for subsequence (function template )
<b>search_n</b>	Search range for elements (function template )


# Algoritmos

## 02: Modificadores 1 de 2

<b>copy</b>	Copy range of elements (function template )
<b>copy_n</b> <small>C++11</small>	Copy elements (function template )
<b>copy_if</b> <small>C++11</small>	Copy certain elements of range (function template )
<b>copy_backward</b>	Copy range of elements backward (function template )
<b>move</b> <small>C++11</small>	Move range of elements (function template )
<b>move_backward</b> <small>C++11</small>	Move range of elements backward (function template )
<b>swap</b>	Exchange values of two objects (function template )
<b>swap_ranges</b>	Exchange values of two ranges (function template )
<b>iter_swap</b>	Exchange values of objects pointed by two iterators (function template )
<b>transform</b>	Transform range (function template )
<b>replace</b>	Replace value in range (function template )
<b>replace_if</b>	Replace values in range (function template )
<b>replace_copy</b>	Copy range replacing value (function template )
<b>replace_copy_if</b>	Copy range replacing value (function template )
<b>fill</b>	Fill range with value (function template )
<b>fill_n</b>	Fill sequence with value (function template )

# Algoritmos

## 02: Modificadores 2 de 2

<b>generate</b>	Generate values for range with function (function template )
<b>generate_n</b>	Generate values for sequence with function (function template )
<b>remove</b>	Remove value from range (function template )
<b>remove_if</b>	Remove elements from range (function template )
<b>remove_copy</b>	Copy range removing value (function template )
<b>remove_copy_if</b>	Copy range removing values (function template )
<b>unique</b>	Remove consecutive duplicates in range (function template )
<b>unique_copy</b>	Copy range removing duplicates (function template )
<b>reverse</b>	Reverse range (function template )
<b>reverse_copy</b>	Copy range reversed (function template )
<b>rotate</b>	Rotate left the elements in range (function template )
<b>rotate_copy</b>	Copy range rotated left (function template )
<b>random_shuffle</b>	Randomly rearrange elements in range (function template )
<b>shuffle</b> 	Randomly rearrange elements in range using generator (function template )

# Algoritmos

## 03: Particiones

<code>is_partitioned</code> <small>C++11</small>	Test whether range is partitioned (function template )
<code>partition</code>	Partition range in two (function template )
<code>stable_partition</code>	Partition range in two - stable ordering (function template )
<code>partition_copy</code> <small>C++11</small>	Partition range into two (function template )
<code>partition_point</code> <small>C++11</small>	Get partition point (function template )

# Algoritmos

## 04: Ordenación

<b>sort</b>	Sort elements in range (function template )
<b>stable_sort</b>	Sort elements preserving order of equivalents (function template )
<b>partial_sort</b>	Partially sort elements in range (function template )
<b>partial_sort_copy</b>	Copy and partially sort range (function template )
<b>is_sorted</b> <small>C++11</small>	Check whether range is sorted (function template )
<b>is_sorted_until</b> <small>C++11</small>	Find first unsorted element in range (function template )
<b>nth_element</b>	Sort element in range (function template )

# Algoritmos

## 05: Búsqueda binaria

<code>lower_bound</code>	Return iterator to lower bound (function template )
<code>upper_bound</code>	Return iterator to upper bound (function template )
<code>equal_range</code>	Get subrange of equal elements (function template )
<code>binary_search</code>	Test if value exists in sorted sequence (function template )

# Algoritmos

## 06: Combinación

<b>merge</b>	Merge sorted ranges (function template )
<b>inplace_merge</b>	Merge consecutive sorted ranges (function template )
<b>includes</b>	Test whether sorted range includes another sorted range (function template )
<b>set_union</b>	Union of two sorted ranges (function template )
<b>set_intersection</b>	Intersection of two sorted ranges (function template )
<b>set_difference</b>	Difference of two sorted ranges (function template )
<b>set_symmetric_difference</b>	Symmetric difference of two sorted ranges (function template )



# Algoritmos

## 07: Montículos

<code>push_heap</code>	Push element into heap range (function template )
<code>pop_heap</code>	Pop element from heap range (function template )
<code>make_heap</code>	Make heap from range (function template )
<code>sort_heap</code>	Sort elements of heap (function template )
<code>is_heap</code> <small>C++11</small>	Test if range is heap (function template )
<code>is_heap_until</code> <small>C++11</small>	Find first element not in heap order (function template )

# Algoritmos

## 08: Mínimo y máximo

<b>min</b>	Return the smallest (function template )
<b>max</b>	Return the largest (function template )
<b>minmax</b> <small>C++11</small>	Return smallest and largest elements (function template )
<b>min_element</b>	Return smallest element in range (function template )
<b>max_element</b>	Return largest element in range (function template )
<b>minmax_element</b> <small>C++11</small>	Return smallest and largest elements in range (function template )

# Algoritmos

09: Otros

<b>lexicographical_compare</b>	Lexicographical less-than comparison (function template )
<b>next_permutation</b>	Transform range to next permutation (function template )
<b>prev_permutation</b>	Transform range to previous permutation (function template )



# Algoritmos

## Ejemplo 03: algoritmo matemáticos

```
#include <iostream>
#include <fstream>
#include <vector>
#include <iterator>
#include <algorithm>
#include <numeric> // para accumulate
using namespace std;

template <class T> // para contar la cantidad de elementos positivos
bool EsPositivo( T val )
{
    return val >= 0;
}

int main()
{
    // cargar una sennal de archivo
    vector<float> senial;
    ifstream archi("ecg.txt");
    copy( istream_iterator<float>(archi), istream_iterator<float>(),
        back_inserter( senial ) );
    archi.close();
    // calcular los valores maximo y minimo
    cout << "Maximo: " << *max_element( senial.begin(), senial.end() ) << endl;
    cout << "Minimo: " << *min_element( senial.begin(), senial.end() ) << endl;
    // calcular el valor medio
    cout << "Valor medio: " << accumulate ( senial.begin(), senial.end(), 0.0 )/senial.size() << endl;
    // contar la cantidad de elementos positivos
    cout << count_if ( senial.begin(), senial.end(), EsPositivo<float> ) << endl;
    return 0;
}
```



# Algoritmos

## Explicación: Ejemplo 03

- **max\_element y min\_element** Buscan el máximo y mínimo de una secuencia y retornan un iterador a el.
- **accumulate** Suma los elementos que está en el rango más el tercer parámetro, que es el valor inicial del sumatorio.
- **size** Obtenemos el número de elementos de la serie
- **count\_if** Cuenta los números que cumplen una condición.
- **back\_inserter** Actúa como un iterador que agrega un elemento al final de la estructura. Su ventaja es que el contenedor no tiene que tener un tamaño inicial conocido y así poder un número indeterminado de elementos del archivo.

# Algoritmos

## Ejemplo 04: algoritmo de transformación

```
#include <iostream>
#include <vector>
#include <iterator>
#include <algorithm>
using namespace std;
void Mayusculas( char & c ) // funcion que pasa una letra a mayusculas
{
    if ( 'a' <= c ) && ( c <= 'z' )
        c = c - ( 'a' - 'A' );
}
char Vocales( char c ) // funcion unaria para identificar vocales
{
    return
        ( c=='A' || c=='E' || c=='I' || c=='O' || c=='U' )? c : '-';
}
int main()
{
    vector<char> v; // generar un vector con letras en minusculas
    for( int i=0; i<10; ++i )
        v.push_back( 'a' + i );
    random_shuffle( v.begin(), v.end() ); // desordenarlas
    copy( v.begin(), v.end(), ostream_iterator<char>(cout) );
    cout << endl;
    for_each( v.begin(), v.end(), Mayusculas ); // aplicar a cada elemento la funcion Mayusculas
    copy( v.begin(), v.end(), ostream_iterator<char>(cout) );
    cout << endl;
    // buscar las vocales
    transform( v.begin(), v.end(), ostream_iterator<char>(cout), Vocales );
    return 0;
}
```



## Sección 2 | Contenedores asociativos



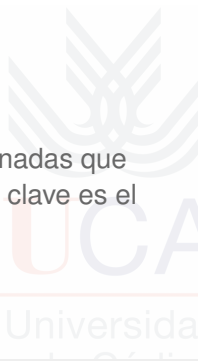
# Contenedores Asociativos

## Introducción

Hasta ahora hemos visto los contenedores lineales, algoritmos e iteradores. Continuamos ahora con los contenedores **asociativos**

- **map**
- **multimap**
- **set**
- **multiset**

Los cuales tienen la característica de almacenar claves ordenadas que están asociadas a un valor, y en el caso de **set** y **multiset** la clave es el valor en sí mismo

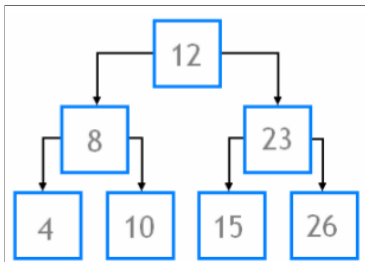




# Contenedores Asociativos

## Estructura

Su estructura en memoria no es secuencial sino que se implementan como **árboles binarios de búsqueda balanceados**.



La operación de comparación para el ordenamiento de las claves se establece en la instanciación, y por defecto se realiza con el operador **menor** (<) para un ordenamiento ascendente. Se debe tener en cuenta que los tipos de datos utilizados como clave deben “soportar” (deben tener sobrecargado) el operador que se utilice para la comparación, en caso de ser clases creadas por el programador.

# Contenedores Asociativos

## Set y Multiset

Estos contenedores se utilizan para manipular conjuntos de elementos, especialmente en operaciones de búsqueda. Tanto en **set** como en **multiset**, la clave de ordenamiento es el valor que se quiere contener. La diferencia entre estos contenedores es que **set no permite la existencia de claves repetidas**, mientras que **multiset sí**.



# Contenedores Asociativos

## Set

- Contenedor asociativo que almacena **elementos únicos** (las claves)
- La implementación más común usa árboles binarios de búsqueda auto-equilibrados y por construcción ordena los datos.
- Diseñados para acceder a los elementos por medio de su clave.



# Contenedores Asociativos

## Ejemplo: Set

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

```
using namespace std;
```

```
int main()
{
    set<string> palabras;
    string palabra;
    while (cin >> palabra)
        palabras.insert(palabra);
    copy(palabras.begin(), palabras.end(), ostream_iterator <string>(cout, "\n"));
}
```



UCA

Universida

# Contenedores Asociativos

## MultiSet

- Tiene la propiedad de poder almacenar varios elementos con la **misma clave**

- Implementación

```
template < class key, class Compare = less<key>, class  
Allocator=allocator<key> » class multiset;
```



# Contenedores Asociativos

## Ejemplo: Set

```
#include <iostream>
#include <algorithm>
#include <set>
#include <iterator>
```

```
using namespace std;
```

```
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;
}
```



UCA

Universida

# Contenedores Asociativos

## Ejemplo: Set y MultiSet

```
#include <iostream>
#include <fstream>
#include <iterator>
#include <set>
using namespace std; // para set y multiset
int main()
{
    set<int> valores;
    ifstream archi("datos.txt");
    istream_iterator<int> i(archi);
    istream_iterator<int> f;
    while( i != f) valores.insert( *i++ );
    archi.close();
    copy( valores.begin(), valores.end(), // mostrar los elementos por pantalla
    ostream_iterator<int>( cout, " " ) );
    set<int>::iterator p = valores.find( 0 ); // buscar un elemento en la estructura
    if( p != valores.end() ) cout << "nEl valor 0 esta en el archivo\n";
    else cout << "nEl valor 0 no esta en el archivo\n";
    multiset<int> multivalores; // comparar con multiset
    archi.open("datos.txt");
    istream_iterator<int> j(archi);
    while( j != f) multivalores.insert( *j++ );
    archi.close();
    copy( multivalores.begin(), multivalores.end(), ostream_iterator<int>( cout, " " ) );
    cout << "nEl valor 0 esta " << multivalores.count( 0 ) << " veces en el archivo\n";
    multivalores.erase( 2 ); // eliminar un elemento
    copy( multivalores.begin(), multivalores.end(),
    ostream_iterator<int>( cout, " " ) );
    return 0;
}
```



# Contenedores Asociativos

## Map

- Contenedores formados de la combinación **clave y valor**
- La clave se usa para identificar de manera única al elemento mientras que el valor mapeado está asociado a la llave.
- Un ejemplo típico de un map es la guía telefónica donde el nombre es la llave y el número telefónico es el valor mapeado.
- Internamente los elementos del mapa están ordenados de menor a mayor
- Implementación  
`template < class key, class T, class Compare = less<key>, class Allocator=allocator<pair<const key, T>» class map;`



# Contenedores Asociativos

## Ejemplo: map

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

using namespace std;

int main()
{
    map<string, long> directory;
    directory["a"] = 1234678;
    directory["b"] = 9876543;
    directory["c"] = 3459876;

    string name = "a";

    if (directory.find(name) != directory.end())
        cout << "The phone number for" << name << " is " << directory[name] << "\n";
    else
        cout << "Sorry, no listing for " << name << "\n";
    return 0;
}
```



UCA

Universida

# Contenedores Asociativos

## multimap

- Igual que map pero soporta claves duplicadas
- Un multimap no puede pues admitir indexado por claves como un map. Para acceder a los distintos valores con la misma clave se emplean las operaciones **equal\_range()**, **lower\_bound()**, **upper\_bound()**



## Sección 3 | Bibliografía



# Bibliografía

## Introducción

- Fundamentos de C++
- Biblioteca de Plantillas Estándar de C++ (STL) - David E. Medina R.
- <http://www.cplusplus.com/reference/stl/>

