C++ Standard Template Library

C++ Standard Template Library
Objetivos

- Comprender los TDAs que nos proporciona la STL.
- Comprender los métodos que nos proporciona la STL.

C++ Standard Template Library
Comprender los TDAs que nos proporciona la STL.

La **STL** nos proporciona:

- **Pila** (stack).
- **Cola** (queue).
- **Cola con prioridad** (priority_queue).
- **Diccionario** (map).
- **Diccionario múltiple** (multimap).
- **Conjunto** (set).
- **Bolsa** (multiset).
- **Vector dinámico** (vector).
- **Lista** (list).
- **Cola con dos puntas** (deque).

C++ Standard Template Library
Comprender los métodos que nos proporciona la STL.

Para **todos** los TDAs contenedores (generalizo, nombre C):

C x;	Declara y crea un contenedor vacío x.
C()	Construye un contenedor vacío.
C(y)	Construye un contenedor como copia de y.
C x(y)	Declara y crea un contenedor x como copia de y.
C x = y	Declara un contenedor x y le asigna el valor de y.
~C()	Destruye un contenedor y sus miembros.

value_type	Tipo de dato que almacena la pila.

size_type size()	Da número de elementos del contenedor.
bool empty()	Da true si contenedor vacío, false si tiene algo.

operator==, operator!=, operator<, ...

C++ Standard Template Library
Comprender los métodos que nos proporciona la STL.

Para **todos** los TDAs contenedores que ofrecen **iteradores**:

begin()	Primer elemento.
end()	Índice tras el último elemento.
rbegin()	Último elemento (ojo, iterar con ite++, no con ite--).
rend()	Índice antes del primer elemento.

iterator	Clase iterador (del primero al último).
const_iterator	Clase iterador (del primero al último) constante.
reverse_iterator	Clase iterador (del último al primero).
const_reverse_iterator	Clase iterador (del último al primero) constante.

Si los TDAs ofrecen acceso aleatorio (**vector**, **deque**) o están fundamentados en un TDA que lo ofrece (**vector**, **deque**), todos los iteradores son de acceso aleatorio (ite+=5).

C++ Standard Template Library
Comprender los métodos que nos proporciona la STL.

Los contenedores que ofrecen iteradores van a ofrecer también métodos insert y erase que van a recibir como parámetros:

Un iterador (inserta o elimina el elemento apuntado).

Dos iteradores (inserta o elimina los elementos que hay desde el primer iterador hasta el segundo).

A continuación se ofrece un resumen de la especificación, destacando los elementos más significativos.

No obstante, se recomienda consultar las guías de referencia cuando falte algo o se tengan dudas:

http://www.cplusplus.com/reference/stl/

http://www.cppreference.com/wiki/stl/start

C++ Standard Template Library
Comprender los métodos que nos proporciona la STL.

Pila (stack):

#include <stack>

template < class T, class Container = deque<T> > class stack;

Ejemplo: stack<int>

void pop()	Elimina el elemento del tope de la pila.
T &top()	Devuelve el elemento del tope de la pila.
void push(const T&v)	Inserta el elemento en el tope de la pila.

C++ Standard Template Library
Comprender los métodos que nos proporciona la STL.

Cola (queue):

#include <queue>

template < class T, class Container = deque<T> > class queue;

Ejemplo: queue<int>

void pop()	Elimina el elemento del frente de la cola.
void push(const T&v)	Inserta el elemento en el final de la cola.
T &front()	Devuelve el elemento del frente de la cola.
T &back()	Devuelve el elemento del final de la cola.

C++ Standard Template Library
Comprender los métodos que nos proporciona la STL.

Cola con prioridad (priority_queue):

#include <queue>

template < class T, class Container = vector<T>,class Compare =
less<typename Container::value_type> > class priority_queue;

Ejemplo: priority_queue<int>

priority_queue(ite_inicio,ite_final,const &comp=..., const &cont=...)

T &top()	Devuelve el elemento al frente de la cola.
void push(const T&v)	Inserta el elemento en la cola (ordenado).
void pop()	Elimina el elemento del frente de la cola.

C++ Standard Template Library
Comprender los métodos que nos proporciona la STL.

Diccionario (map):

#include <map>
template < class Key, class T, class Compare = less<Key>,
class Allocator = allocator<pair<const Key,T> > > class map;
Ejemplo: map<int,string> **//INTERNAMENTE pair (ojo, *ite -> pair)**

map(ite_inicio,ite_final,const &comp=..., const &allocator=...)

insert(pair<key,T>/it)	Inserta elemento, clave->objeto/iterador(es).
find(key)	Devuelve iterador al objeto.
erase(key/ite)	Elimina elem. identificado por clave/iterador(es).
clear()	Vacia el contenedor.
swap(map1,map2)	Intercambia el contenido de dos maps.
count(key)	Devuelve núm elementos con una clave dada.
lower_bound(key)	Devuelve iterador a cota inferior de clave.
upper_bound(key)	Devuelve iterador a cota superior de clave.
equal_range(k1,k2)	Devuelve iteradores a rango de elem. entre claves.
operator[](clave)	Devuelve/crea ref. a elemento con esa clave.
max_size()	Devuelve tamaño máximo posible.

C++ Standard Template Library
Comprender los métodos que nos proporciona la STL.

Diccionario múltiple (multimap):

```
#include <map>
template < class Key, class T, class Compare = less<Key>,
        class Allocator = allocator<pair<const Key,T> > > class multimap;
```

Ejemplo: multimap<int,string> **//INTERNAMENTE pair (*ite -> pair)**

multimap(ite_inicio,ite_final,const &comp=..., const &allocator=...)

insert(pair<key,T>/it)	Inserta elemento, clave->objeto/iterador(es).
find(key)	Devuelve iterador al primer objeto con esa clave.
erase(key/ite)	Elimina todos elem. identificado por clave/it(es).
clear()	Vacia el contenedor.
swap(mm1,mm2)	Intercambia el contenido de dos multimaps.
count(key)	Devuelve núm elementos con una clave dada.
lower_bound(key)	Devuelve iterador a cota inferior de clave.
upper_bound(key)	Devuelve iterador a cota superior de clave.
equal_range(k1,k2)	Devuelve iteradores a rango de elem. entre claves.
operator[](clave)	Devuelve/crea ref. a elemento con esa clave.
max_size()	Devuelve tamaño máximo posible.

C++ Standard Template Library
Comprender los métodos que nos proporciona la STL.

Conjunto (set):

```
#include <set>
template < class Key, class Compare = less<Key>,
        class Allocator = allocator<Key> > class set;
Ejemplo: set<int>

set(ite_inicio,ite_final,const &comp=..., const &allocator=...)
```

insert(obj/ite)	Inserta objeto (objeto, iterador(es)).
find(obj)	Devuelve iterador a ese objeto.
erase(obj)	Elimina objeto.
clear()	Vacia el contenedor.
swap(set1,set2)	Intercambia el contenido de dos sets.
count(obj)	Devuelve núm de copias de ese objeto.
lower_bound(obj)	Devuelve iterador a cota inferior de objeto.
upper_bound(obj)	Devuelve iterador a cota superior de objeto.
equal_range(o1,o2)	Devuelve iteradores a rango de elem. entre objs.
max_size()	Devuelve tamaño máximo posible.

C++ Standard Template Library
Comprender los métodos que nos proporciona la STL.

Bolsa (multiset):

```
#include <set>
template < class Key, class Compare = less<Key>,
        class Allocator = allocator<Key> > class multiset;
Ejemplo: multiset<int>

multiset(ite_inicio,ite_final,const &comp=..., const &allocator=...)
```

insert(obj/ite)	Inserta objeto (objeto, iterador(es)).
find(obj)	Devuelve iterador a la primera copia de ese objeto.
erase(obj)	Elimina todas las instancias de ese objeto.
clear()	Vacia el contenedor.
swap(ms1,ms2)	Intercambia el contenido de dos multisets.
count(obj)	Devuelve núm de copias de ese objeto.
lower_bound(obj)	Devuelve iterador a cota inferior de objeto.
upper_bound(obj)	Devuelve iterador a cota superior de objeto.
equal_range(o1,o2)	Devuelve iteradores a rango de elem. entre objs.
max_size()	Devuelve tamaño máximo posible.

C++ Standard Template Library
Comprender los métodos que nos proporciona la STL.

Vector dinámico (vector):

#include <vector>
template < class T, class Allocator = allocator<T> > class vector;
Ejemplo: vector<int>

vector(ite_inicio,ite_final,const &comp=..., const &allocator=...)

insert(ite,(n),ele/ites)	Inserta (n) elementos/ite_ini-ite_fin en posición.
erase(ite)	Elimina elemento en iterador o entre iteradores.
clear()	Vacia el contenedor.
swap(vec1,vec2)	Intercambia el contenido de dos listas.
operator[](pos)	Devuelve REFERENCIA a elemento en esa pos.
max_size()	Devuelve tamaño máximo posible.
resize(tama,ele)	Rellena con ele/recorta el vector hasta tama.
capacity()	Devuelve tamaño reservado.
reserve(tama)	Solicita un cambio en tamaño reservado.
push_back()/pop_back(ele)	Inserta/extrae elemento al final.
front()/back()	Devuelve elemento del principio/al final.

C++ Standard Template Library
Comprender los métodos que nos proporciona la STL.

Lista (list):

#include <list>
template < class T, class Allocator = allocator<T> > class vector;
Ejemplo: list<int>

list(ite_inicio,ite_final, const &allocator=...)

insert(ite,(n),ele/ites)	Inserta (n) elementos/ite_ini-ite_fin en posición.
erase(ite)	Elimina elemento en iterador o entre iteradores.
remove(elem)	Elimina elementos con valor elem.
clear()	Vacia el contenedor.
swap(list1,list2)	Intercambia el contenido de dos listas.
push_back()/pop_back(ele)	Inserta/extrae elemento al final.
push_front()/pop_front(ele)	Inserta/extrae elemento al frente.
front()/back()	Devuelve elemento del principio/al final.
unique()	Eliminar valores duplicados.
sort() / reverse()	Ordena elementos / Invierte orden elementos.
merge(list2,(comp))	Une listas ordenadas.
splice(pos,list,ite(s))	Mueve elementos de una lista a otra.

C++ Standard Template Library
Comprender los métodos que nos proporciona la STL.

Cola con dos puntas (deque):

#include <deque>
template < class T, class Allocator = allocator<T> > class deque;
Ejemplo: deque<int>

deque(ite_inicio,ite_final,const &comp=..., const &allocator=...)

insert(ite,(n),ele/ites)	Inserta (n) elementos/ite_ini-ite_fin en posición.
erase(ite)	Elimina elemento en iterador o entre iteradores.
clear()	Vacia el contenedor.
swap(deq1,deq)	Intercambia el contenido de dos deques.
operator[](pos)	Devuelve REFERENCIA a elemento en esa pos.
max_size()	Devuelve tamaño máximo posible.
resize(tama,ele)	Rellena con ele/recorta el vector hasta tama.
push_back()/pop_back(ele)	Inserta/extrae elemento al final.
push_front()/pop_front(ele)	Inserta/extrae elemento al frente.
front()/back()	Devuelve elemento del principio/al final.

C++ Standard Template Library
Comprender los métodos que nos proporciona la STL.

¿Cómo aprendemos los nombres y los usos de todos estos métodos?

Sólo hay una manera:

Programando Y **ejecutando**.

El compilador nos indicará con sus errores qué estamos haciendo mal.

C++ Standard Template Library
¿Qué hemos aprendido?

- **Los TDAs que nos proporciona la STL.**
- **Los métodos que nos proporciona la STL.**