La STL

La STL (Standard Template Library) de C++ es una de las características más potentes del lenguaje, ya que proporciona un conjunto de clases y funciones genéricas que permiten manejar colecciones de datos y realizar operaciones comunes de manera eficiente. Se basa en tres componentes principales:

1. **Contenedores** (Containers)
Los contenedores *son estructuras de datos genéricas que almacenan y organizan colecciones de objetos*. La STL ofrece varios tipos de contenedores, que se dividen en tres categorías principales:

1.1 Contenedores Secuenciales
Almacenan los elementos de forma secuencial, en el orden en que se añaden.

std::vector: Arreglo dinámico, puede cambiar su tamaño en tiempo de ejecución.

Ventaja: Acceso aleatorio rápido.
Desventaja: Inserción o eliminación de elementos en el medio puede ser lenta.

std::deque: Arreglo doblemente terminado que permite inserciones y eliminaciones en ambos extremos.

Similar a un vector, pero más eficiente cuando se hacen muchas inserciones/eliminaciones en los extremos.

std::list: Lista doblemente enlazada.

Ventaja: Inserción y eliminación eficiente en cualquier parte.
Desventaja: No tiene acceso aleatorio rápido (como vector o deque).

1.2 Contenedores Asociativos
Permiten almacenar datos de manera ordenada o no ordenada y proveen búsqueda rápida.

std::set: Conjunto ordenado de elementos únicos.

Utiliza un árbol binario de búsqueda balanceado (normalmente un Red-Black Tree).

std::map: Estructura de datos que almacena pares clave-valor, donde las claves son únicas.

Similar a set, pero cada clave se asocia a un valor.

std::multiset: Similar a set, pero permite elementos duplicados.

std::multimap: Similar a map, pero permite claves duplicadas.

std::unordered_set y std::unordered_map: Versiones sin orden de set y map que utilizan una tabla hash para lograr una búsqueda más rápida, aunque sin garantizar orden.

1.3 Contenedores Adaptadores
No son propiamente contenedores, sino adaptadores de contenedores.

std::stack: Implementa una pila (LIFO – Last In, First Out) utilizando otros contenedores como base (por defecto, deque).

std::queue: Implementa una cola (FIFO – First In, First Out) utilizando otros contenedores.

std::priority_queue: Cola con prioridad, donde los elementos de mayor prioridad se atienden primero.

2. Algoritmos (Algorithms)
La STL incluye una variedad de algoritmos que permiten realizar operaciones comunes sobre contenedores, como búsqueda, ordenación, modificación, etc. Todos estos algoritmos son funciones genéricas que pueden operar sobre diferentes tipos de contenedores, siempre que estos soporten el uso de iteradores.

Algunos algoritmos más comunes incluyen:

std::sort: Ordena los elementos de un contenedor.
std::find: Busca un elemento en un contenedor.
std::for_each: Aplica una función a cada elemento de un contenedor.
std::copy: Copia elementos de un contenedor a otro.
std::accumulate: Suma todos los elementos en un contenedor.
Cada algoritmo de la STL está diseñado para ser altamente eficiente y se puede personalizar mediante el uso de functors o funciones lambda.

3. Iteradores (Iterators)
Los iteradores proporcionan una forma de acceder a los elementos de los contenedores secuenciales y asociativos de manera genérica. Son un concepto fundamental en la STL, ya que permiten escribir código independiente del tipo de contenedor que se esté utilizando.

Tipos principales de iteradores:

Iteradores de Entrada: Se utilizan para leer datos de una secuencia.
Iteradores de Salida: Se utilizan para escribir datos en una secuencia.
Iteradores de Avance (Forward): Pueden moverse en una secuencia hacia adelante.
Iteradores Bidireccionales: Pueden moverse tanto hacia adelante como hacia atrás en una secuencia (ej: std::list).
Iteradores Aleatorios: Permiten acceder a cualquier elemento de una secuencia en tiempo constante (ej: std::vector).

Ejemplo básico de iteradores:

```
#include <iostream>
#include <vector>

int main() {
	std::vector<int> v = {1, 2, 3, 4, 5};

	// Usando iteradores para recorrer el vector
	for (std::vector<int>::iterator it = v.begin(); it != v.end(); ++it) {
		std::cout << *it << " ";
	}

	return 0;
}
```

En este ejemplo, utilizamos un iterador para recorrer los elementos del vector.

4. Funciones y Functores
Los algoritmos de la STL se pueden personalizar con functores, que son objetos de función, y lambdas para realizar operaciones específicas. Por ejemplo, si queremos ordenar un vector en orden descendente, podemos usar un functor o una lambda personalizada.

Ejemplo con lambda:

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

int main() {
	std::vector<int> v = {1, 5, 3, 2, 4};

	// Ordenar en orden descendente usando una lambda
	std::sort(v.begin(), v.end(), [](int a, int b) { return a > b; });

	for (int n : v) {
		std::cout << n << " ";
	}

	return 0;
}
```

5. Ventajas de la STL
Reutilización de código: Los contenedores y algoritmos de la STL son altamente reutilizables.
Optimización: Implementaciones eficientes de estructuras de datos y algoritmos.
Genéricos: Pueden trabajar con cualquier tipo de dato, gracias al uso de plantillas.
Modularidad: Se puede separar la lógica de datos de los algoritmos, permitiendo que los algoritmos sean reutilizables con diferentes tipos de contenedores.

6. Consideraciones de uso
La STL es parte del estándar de C++ y se recomienda usarla, en lugar de implementar estructuras de datos propias.
Es importante entender el costo de cada contenedor (por ejemplo, el acceso aleatorio en un vector es constante, mientras que en una list es lineal).