

STL Standard Template Library

La STL tiene tres pilares fundamentales que son: Contenedores(donde se guardan los datos), Iteradores(el puente para acceder a ellos) y Algoritmos(funciones para manipularlos)

CLASIFICACION DE LOS CONTENEDORES

En C++ los contenedores se dividen en tres categorías principales segun como organizan los datos en la memoria y como permiten acceder a ellos:

1. Contenedores Secuenciales

Mantienen el orden en el que insertas los elementos. La posición de cuando y donde los pusiste.

- ❖ `std::vector`: es un arreglo dinámico.
- ❖ `std::deque`: es una cola de doble final (Double-Ended Queue).
- ❖ `std::list`: es una lista doblemente enlazada.

2. Contenedores Asociativos

No mantienen un orden cronológico, sino que se basan en las llaves (keys). Los datos se ordenan automáticamente (usualmente mediante arboles binarios) para permitir búsquedas rápidas.

- ❖ `std::set` : es una colección de elementos únicos.
- ❖ `std::map` : parejas de clave-valor.

3. Adaptadores de Contenedores

No son contenedores “reales” por si mismos, sino que imitan la interfaz de un contenedor secuencial para que se comporte de una forma específica.

- ❖ `std::stack` : LIFO (Last In, First Out).
- ❖ `std::queue` : FIFO (First In, First Out).

1. COMPARATIVA DE CONTENEDORES SECUENCIALES

`std::vector` (Arreglo Dinamico)

Todo esta en un bloque contiguo. Si quieres el elemento `v[2]`, el procesador simplemente salta a la direccion de memoria exacta.

Memoria: [2 | 5 | 26 | 48]

^ ^ ^ ^

Índice: 0 1 2 3

- Comportamiento: Si insertas al inicio, tiene que “empujar” todos los demas hacia la derecha. Muy costoso ($O(n)$).

`std::deque` (Cola de Doble Final)

Usa un mapa de punteros a diferentes bloques .

Mapa Central: [Bloque A | Bloque B]

| |
v v

Bloque A: [2 | 5 | 26] Bloque B: [98 | 55 | 13]

- Comportamiento: Puede añadir bloques nuevos al inicio o al final sin mover los datos existentes. Por eso es rapido en ambos extremos ($O(1)$).

`std::list` (Lista Doblemente Enlazada)

No hay bloques. Cada elemento es un nodo que vive en cualquier parte de la memoria y apunta al siguiente y al anterior.

[nullptr | 2 | *] <-> [* | 5 | *] <-> [* | 26 | nullptr]

- Comportamiento: No puedes saltar a la posicion 10 directamente. Tienes que seguir los punteros uno por uno ($O(n)$). Pero insertar es solo cambiar un par de flechas.

2. COMPARATIVA DE CONTENEDORES ASOCIATIVOS

Aqui la memoria no es lineal, sino jerarquica.

`std::map` y `std::set`

Se organizan como un Arbol Binario. Cada nodo tiene un valor (o clave-valor) y dos “hijos”.

[Clave: 26] <-- Raíz

/ \

[Clave: 5] [Clave: 48]

- Comportamiento: Siempre estan ordenados. Par buscar el 48, el programa pregunta: “Es mayor que 26? Si, ve a la derecha”. Esto es muy eficiente para conjuntos grandes ($O(\log n)$).

3. ADAPTADORES (El concepto de “ENVOLTURA”)

`std::stack` y `std::queue`

No tienen una estructura de memoria propia. Por defecto, son como una “caja” que envuelve a un `std::deque`.

STACK:

[Entrada/Salida] <-- Solo el tope es visible

|

v

[Bloque de Deque] <-- Los datos están aquí abajo, protegidos.

- Comportamiento: El stack le dice al deque : “Solo dejame usar tus funciones `push_back` y `pop_back`”. Oculta todo lo demas (como los iteradores) para garantizar que nadie rompa la regla LIFO.

METODOS DE ITERACION Y BORRADO

Método	Propósito	¿Quién lo usa?
push_back()	Añade al final.	vector, deque, list
push_front()	Añade al inicio.	deque, list
pop_back()	Elimina el último.	vector, deque, list
pop_front()	Elimina el primero.	deque, list
insert()	Inserta en una posición específica.	vector, deque, list, map, set
erase()	Elimina un elemento específico.	Todos los contenedores

METODOS DE ACCESO Y BUSQUEDA

Método	Propósito	¿Quién lo usa?
at() o []	Acceso directo por índice.	vector, deque
front()	Mira el primer elemento.	vector, deque, list
back()	Mira el último elemento.	vector, deque, list
top()	Mira el elemento superior.	stack
find()	Busca una llave específica.	map, set

METODOS DE CAPACIDAD Y ESTADO

Método	Propósito	¿Quién lo usa?
size()	Cuántos elementos hay.	Todos

<code>empty()</code>	¿Está vacío? (devuelve true/false).	Todos
<code>clear()</code>	Borra todo el contenido.	Todos (excepto adaptadores)

CASOS ESPECIALES

`std::stack` y `std::queue` tienen adaptadores y metodos especificos para reforzar su logica

- `push()` Para meter datos equivale a `push_back`.
- `pop()` Para sacar datos (en stack el ultimo y en queue el primero)