Ayudantía 1

STL de C++ y Tipos abstractos de datos

Ayudante: Matías Francia

Email: matias.francia@usm.cl

Biblioteca Estándar del Lenguaje C++

Qué es una biblioteca estándar?

Es una librería que contiene herramientas, clases y funciones predefinidas que conforman una parte del lenguaje de programación.

Hacen que sea más rápido y eficiente programar.

Particularmente en C++, las operaciones de la STL están implementadas de forma muy eficiente.

Navegar en la STL

No es complejo, pero hay algunos detalles que considerar:

1. Entender de lo macro a lo micro.
2. Fijarse en las versiones
3. Ver la complejidad temporal de las operaciones (ej. erase, push_back, etc.)

Contenedores

Es una clase o estructura de datos que permite almacenar de forma eficiente y ordenada conjuntos de objetos.

Maneja el espacio en memoria para sus elementos y provee métodos para acceder a ellos, directamente o a través de ***iteradores*** (objetos de referencia similares a los punteros).

Replican estructuras comunes como los arrays dinámicos (vector), queues (queue), stacks (stack), entre otros.

Contenedores

Contenedores Secuenciales: almacenan elementos en un orden lineal, lo que permite acceder a ellos en un orden específico (array, vector, deque, forward list, list).

Contenedores Adaptadores: no son contenedores en sí mismos, sino interfaces que adaptan otros contenedores para proporcionar una funcionalidad específica, por ejemplo, pilas o colas (stack, queue, priority_queue).

Contenedores Asociativos: almacenan elementos como pares clave-valor y están organizados de manera que permiten búsquedas rápidas basadas en las claves (set, map, multiset, multimap).

Contenedores Asociativos Desordenados: son prácticamente iguales que los asociativos, con la salvedad de que no se guardan los elementos de forma ordenada.

Contenedores Secuenciales

Array

Características:

Uso estático de memoria, no son redimensionables, eficiente uso de memoria y de acceso a los elementos.

Se puede usar un offset del puntero para acceder a los elementos siguientes.

```cpp
#include <iostream>
#include <array>

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

    for (int n : numeros) {
        std::cout << n << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

Vector

Características:

Uso dinámico de memoria, son redimensionables (pide una cantidad fija de memoria cada ciertos intervalos de tamaño del vector), eficiente uso de memoria y de acceso a los elementos.

Manejo de memoria lo realiza un objeto de tipo allocator asociado al vector.

Se puede usar un offset del puntero para acceder a los elementos siguientes.

Es relativamente eficiente agregar o eliminar elementos al final del vector, no así los que están en medio o al comienzo.

Vector

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

int main() {
    std::vector<int> numeros = {1, 2, 3, 4, 5};
    numeros.push_back(6); // Agrega un elemento al final
    numeros[0] = 10; // Modifica el primer elemento

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

    std::cout << std::endl;

    return 0;
}
```

Deque

Características:

Tamaño dinámico, puede ser contraído o expandido eficientemente tanto al comienzo como al final.

No almacena los elementos en espacios adyacentes de memoria (offset sobre el puntero da *undefined behavior*).

Acceso a elementos en tiempo constante.

Más eficiente que vectores para secuencias grandes (se complica encontrar un gran espacio de memoria).

Funcionan peor que las list o forward_list para insertar o eliminar elementos que no están ni al comienzo ni al final, y tienen menos ***consistent iterators***.

Deque

```cpp
#include <iostream>
#include <deque>

int main() {
    std::deque<int> numeros = {1, 2, 3, 4, 5};
    numeros.push_front(0);
    numeros.push_back(6);

    for (int n : numeros) {
        std::cout << n << " ";
    }
    std::cout << std::endl;
    return 0;
}

```

Forward List

Características:

Son básicamente listas simplemente enlazadas.

Permiten inserción y eliminación en tiempo constante en cualquier lugar de la secuencia, inclusive si se elimina un rango completo de elementos.

Los elementos pueden ser almacenados en cualquier lugar de la memoria, el elemento anterior posee un puntero al elemento siguiente.

Forward List

Desventajas:

No posee acceso directo a posiciones de la secuencia, sino que se tiene que recorrer.

Consumen un poco más de memoria que otros contenedores.

No posee método size, se calcula con *distance* (tiempo lineal).

No posee member type *reverse_iterator*.

```cpp
#include <iostream>
#include <forward_list>

int main() {
    std::forward_list<int> numeros = {1, 2, 3, 4, 5};
    numeros.push_front(0);

    auto it = numeros.begin(); // Iterador al primer elemento (0)
    numeros.insert_after(it, 10); // Insertar 10 después de 0

    std::cout << "Elementos de la forward_list: ";
    for (int n : numeros) {
        std::cout << n << " ";
    }
    std::cout << std::endl;

    return 0;
}✨
```

List

Características:

Lista doblemente enlazada.

Se puede recorrer en ambos sentidos (posee *reverse_iterator*).

Mismas ventajas y desventajas que *forward_list*

```cpp
#include <iostream>
#include <list>

int main() {
    std::list<int> numeros = {1, 2, 3, 4, 5};
    numeros.push_back(6);
    numeros.push_front(0);

    for (int n : numeros) {
        std::cout << n << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

Contenedores Adaptadores

Stack

Características:

Estructura LIFO (pila). Útil cuando los elementos sólo deben ser tomados de un extremo del contenedor.

Utiliza un contenedor secuencial por debajo, puede ser: vector, deque, list (por defecto es deque).

También se puede usar una clase de contenedor definida especialmente, siempre y cuando tenga los métodos empty, size, back, push_back, pop_back.

```cpp
#include <iostream>
#include <stack>

int main() {
    std::stack<int> pila;
    pila.push(1);
    pila.push(2);
    pila.push(3);

    while (!pila.empty()) {
        std::cout << pila.top() << " ";
        pila.pop();
    }
    std::cout << std::endl;
    return 0;
}
```

Queue

Características:

Estructura FIFO (cola). Útil cuando los elementos deben ser insertados en un extremo de la secuencia y extraídos del otro.

Los elementos son agregados al final de la secuencia, y extraídos del frente.

Utiliza un contenedor secuencial por debajo, puede ser: deque, list (por defecto es deque).

También se puede usar una clase de contenedor definida especialmente, siempre y cuando tenga los métodos empty, size, front, back, push_back, pop_front.

```cpp
#include <iostream>
#include <queue>

int main() {
    std::queue<int> cola;
    cola.push(1);
    cola.push(2);
    cola.push(3);

    while (!cola.empty()) {
        std::cout << cola.front() << " ";
        cola.pop();
    }
    std::cout << std::endl;
    return 0;
}
```

Priority Queue

Características:

Es una cola en la que el primer elemento es el “más grande”, basado en función establecida en su creación.

Se pueden insertar elementos en cualquier momento, pero sólo se puede extraer el mayor.

El contenedor subyacente puede ser vector, deque o uno custom. Por defecto es vector.

```cpp
#include <iostream>
#include <queue>

int main() {
    std::priority_queue<int> colaPrioridad;
    colaPrioridad.push(10);
    colaPrioridad.push(30);
    colaPrioridad.push(20);
    colaPrioridad.push(5);

    while (!colaPrioridad.empty()) {
        std::cout << colaPrioridad.top() << " ";
        colaPrioridad.pop();
    }
    std::cout << std::endl;
    return 0;
}

```

Contenedores Asociativos

Set

Características:

Se accede a los elementos a través de su valor. No pueden haber valores repetidos.

Suelen ser implementados como *Binary Search Trees*.

Los elementos no pueden ser modificados, sólo insertados o eliminados. Además, se ordenan siguiendo el criterio definido en el constructor.

Acceder a los elementos es más lento que en *unordered_set*.

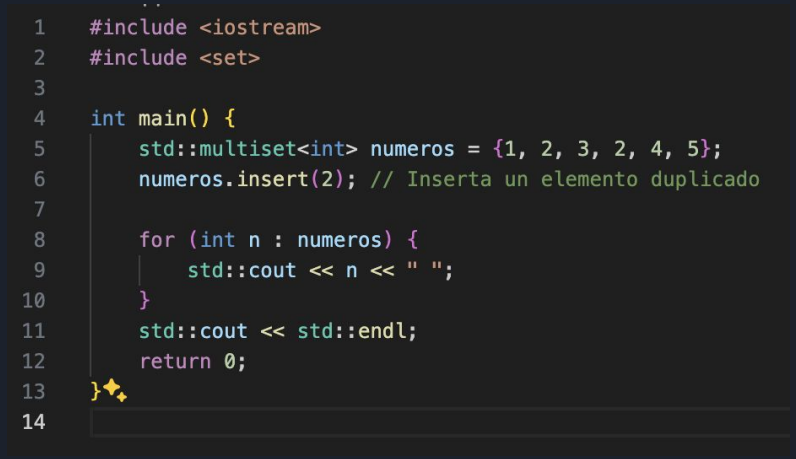
Qué pasa acá?

```cpp
#include <iostream>
#include <set>

int main() {
    std::set<int> numeros = {5, 1, 3, 2, 4};
    numeros.insert(2); // Intentar insertar un elemento repetido

    for (int n : numeros) {
        std::cout << n << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

Multiset

Características:

Mismas características que set, principal diferencia es que permite que existan múltiples elementos con el mismo valor.



```cpp
#include <iostream>
#include <set>

int main() {
    std::multiset<int> numeros = {1, 2, 3, 2, 4, 5};
    numeros.insert(2); // Inserta un elemento duplicado

    for (int n : numeros) {
        std::cout << n << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

Map

Características:

Poseen una *key* y un *value*, y se guardan ordenados, basándose en la *key* y en un criterio establecido en el constructor.

Los map son generalmente más lentos que los unordered_map para acceder a elementos, y se puede acceder a ellos mediante el operador [].

Se suelen implementar como *Binary Search Trees*.

```cpp
#include <iostream>
#include <map>

int main() {
    std::map<std::string, int> edad = {{"Juan", 30}, {"Ana", 25}, {"Luis", 35}};
    edad["Pedro"] = 28;

    for (const auto& par : edad) {
        std::cout << par.first << ": " << par.second << std::endl;
    }
    return 0;
}
```

Multimap

Características:

En características, son casi lo mismo que los map, solo que permiten múltiples elementos con el mismo valor.

```cpp
#include <iostream>
#include <map>

int main() {
    std::multimap<std::string, int> notas = {{"Juan", 85}, {"Ana", 90}, {"Juan", 88}};
    notas.insert({"Ana", 92});

    for (const auto& par : notas) {
        std::cout << par.first << ": " << par.second << std::endl;
    }
    return 0;
}
```

Contenedores Asociativos No Ordenados

Unordered Set

Características:

Similares a los set. Elementos no ordenados y no permite replicados.

El contenedor construye una tabla hash y agrupa los elementos en *buckets*, así la búsqueda es más rápida.

Complejidad promedio O(1).

La función de hash se le puede entregar.

```cpp
#include <iostream>
#include <unordered_set>

int main() {
    std::unordered_set<int> numeros = {5, 1, 3, 2, 4};
    numeros.insert(2);

    for (int n : numeros) {
        std::cout << n << " ";
    }
    std::cout << std::endl;
    return 0;
}

```

Unordered Multiset

Características:

Es como el Unordered Set, pero permite valores repetidos.

Está en <unordered_set>, no tiene header propio.

Unordered Map

Características:

Similares a los map. Elementos no ordenados y no permite replicados.

El contenedor construye una tabla hash y agrupa los elementos en *buckets*, así la búsqueda es más rápida.

Complejidad promedio O(1).

```cpp
#include <iostream>
#include <unordered_map>

int main() {
    std::unordered_map<std::string, int> edad = {{"Juan", 30}, {"Ana", 25}, {"Luis", 35}};
    edad["Pedro"] = 28;

    for (const auto& par : edad) {
        std::cout << par.first << ": " << par.second << std::endl;
    }
    return 0;
}
```

Unordered Multimap

Características:

Es como el Unordered Map, pero permite valores repetidos.

Está en <unordered_map>, no tiene header propio.