## 2. Estructura de Datos: `Set`

La estructura `unordered_set` es una coleccion de elementos únicos sin un orden definido, implementada internamente mediante una tabla hash. Forma parte de la biblioteca `<unordered_set>` de la STL (Standard Template Library) de C++.

Es ideal cuando se requiere:

- Almacenar datos sin duplicados.
- Acceder, insertar y eliminar elementos en tiempo constante promedio.
- No mantener ningun orden especifico de los elementos.

### Funciones Utiles

- `insert(x)` — Inserta el valor $x$ si no existe.
- `erase(x)` — Elimina el valor $x$ si existe.
- `count(x)` — Retorna 1 si $x$ esta en el set, 0 si no.
- `find(x)` — Devuelve un iterador al elemento $x$ o `end()` si no existe.
- `size()` — Cantidad de elementos almacenados.
- `empty()` — Retorna `true` si esta vacio.
- `clear()` — Elimina todos los elementos.

### Ejemplo de Uso en C++

Listing 1: Ejemplo basico con unordered_set

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

using namespace std;

int main() {
    unordered_set<int> numeros;

    // Insertar elementos
    numeros.insert(10);
    numeros.insert(20);
    numeros.insert(30);
    numeros.insert(20); // No se inserta porque ya existe

    // Recorrer el set
    cout << "Elementos del set:" << endl;
    for (int x : numeros) {
```

```
18          cout << x << " ";
19      }
20      cout << endl;
21
22      // Buscar elementos
23      if (numeros.count(20)) {
24          cout << "El elemento 20 esta en el set." << endl;
25      }
26
27      // Eliminar
28      numeros.erase(10);
29
30      // Verificar tamano
31      cout << "Tamano actual: " << numeros.size() << endl;
32
33      return 0;
34  }
```

## Ventajas y Desventajas

**Ventajas:**

- Operaciones de insercion, busqueda y eliminacion eficientes: tiempo promedio $\mathcal{O}(1)$.
- Evita automaticamente duplicados.
- Facil de usar y sintaxis intuitiva.

**Desventajas:**

- No mantiene orden entre los elementos. Esto puede evitarse usando otra biblioteca: set. Es parecida pero mantiene el orden. Como desventaja de esto, la búsqueda pasa a ser $\mathcal{O}(log(n))$.
- El tiempo puede degradarse a $\mathcal{O}(n)$ en el peor caso debido a muchas colisiones. Esto se debe a que la estructura interna de un set posee una función de Hash, que puede no ser inyectiva.
- No se puede acceder por indice.

## Complejidad de Operaciones

- **Insercion:** $\mathcal{O}(1)$ promedio, $\mathcal{O}(n)$ peor caso.
- **Busqueda:** $\mathcal{O}(1)$ promedio, $\mathcal{O}(n)$ peor caso.
- **Eliminacion:** $\mathcal{O}(1)$ promedio, $\mathcal{O}(n)$ peor caso.

## Aplicaciones Comunes

- Almacenamiento de claves unicas.
- Verificacion de existencia de elementos.
- Eliminacion de duplicados en listas.
- Optimizacion en algoritmos que requieren comprobacion rapida de pertenencia.