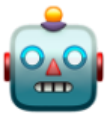

Programacion Competitiva

C++ | UTN

Resumen de STL en C++

Introducción al STL (Standard Template Library)

La STL (Standard Template Library) es un conjunto de clases y funciones en C++ diseñadas para proporcionar estructuras de datos genéricas y algoritmos eficientes. Incluye:

- **Contenedores**: estructuras de datos (como `vector`, `map`, `set`, etc.).
- **Iteradores**: permiten recorrer los elementos de los contenedores.
- **Algoritmos**: funciones genéricas como `sort`, `find`, `count`, etc.
- **Funciones auxiliares**: como `pair`, `make_pair`, `auto`, etc.

auto

Palabra clave que permite al compilador deducir el tipo de una variable automáticamente.

```
auto x = 5;          // int
auto name = "Hola";  // const char*
```

range-based for

Forma simplificada de recorrer contenedores:

```
vector<int> v = {1, 2, 3};
for (auto x : v) {
    std::cout << x << " ";
}
```

Contenedores de STL

1. Vector

- Arreglo dinámico.
- Acceso por índice O(1).

Funciones importantes:

`push_back`, `pop_back`, `size`, `empty`, `clear`, `resize`, `at`, `front`, `back`, `begin`, `end`.

Código ejemplo:

```
vector<int> v = {1, 2, 3};
v.push_back(4);
for (auto x : v) cout << x << " ";
```

---

2. Deque

- Similar a vector pero permite inserción/eliminación por ambos extremos.

Funciones importantes:

`push_front`, `push_back`, `pop_front`, `pop_back`, `front`, `back`, `at`, `size`.

Código ejemplo:

```
deque<int> d = {1, 2};
d.push_front(0);
d.push_back(3);
```

---

3. List

- Lista doblemente enlazada.
- Inserción/eliminación eficiente en cualquier parte (O(1)).

Funciones importantes:

`push_back`, `push_front`, `pop_back`, `pop_front`, `insert`, `erase`, `sort`, `reverse`, `unique`, `merge`.

Código ejemplo:

```
list<int> l = {1, 2, 3};
l.push_front(0);
l.sort();
```

---

4. Set

- Estructura que almacena valores únicos ordenados automáticamente.
- Internamente usa árboles binarios balanceados.

Funciones importantes:

`insert`, `erase`, `find`, `count`, `lower_bound`, `upper_bound`.

Código ejemplo:

```
set<int> s;
s.insert(3);
s.insert(1);
for (auto x : s) cout << x << " "; // 1 3
```

---

5. Map

- Diccionario clave-valor. Las claves son únicas y están ordenadas.

Funciones importantes:

`insert`, `erase`, `find`, `count`, `operator[]`, `at`, `begin`, `end`.

Código ejemplo:

```
map<string, int> m;
m["uno"] = 1;
m["dos"] = 2;
```

---

6. Unordered Set / Map

- Igual que `set` / `map`, pero no mantienen orden. Usan `hash tables`.

- Operaciones promedio en O(1).

Funciones importantes:

Mismas que `set` y `map`, pero sin `lower_bound` ni `upper_bound`.

Código ejemplo:

```
unordered_map<string, int> um;
um["a"] = 10;
```

---

7. Stack (adaptador de contenedor)

- LIFO (Last In First Out).

Funciones importantes:

`push`, `pop`, `top`, `empty`, `size`.

Código ejemplo:

```
stack<int> st;
st.push(1);
st.push(2);
cout << st.top();
```

---

8. Queue (adaptador FIFO)

Funciones importantes:

`push`, `pop`, `front`, `back`, `empty`, `size`.

Código ejemplo:

```
queue<int> q;
q.push(1);
q.push(2);
cout << q.front();
```

---

9. Priority Queue

- Cola con prioridad (heap). Máximo valor por defecto.

Funciones importantes:

`push`, `pop`, `top`, `empty`, `size`.

Código ejemplo:

```
priority_queue<int> pq;
pq.push(3);
pq.push(1);
cout << pq.top(); // 3
```

---

10. Multimap

- Permite claves duplicadas, mantiene orden.

Funciones importantes:

`insert`, `find`, `equal_range`, `count`, `erase`, `lower_bound`, `upper_bound`.

Código ejemplo:

```
multimap<int,string> mm;
mm.insert({1, "uno"});
mm.insert({1, "uno duplicado"});
```

---

11. Bitset

- Arreglo de bits de tamaño fijo, muy eficiente en espacio/tiempo.

Funciones importantes:

`set`, `reset`, `flip`, `any`, `none`, `count`, `test`, `to_string`.

Código ejemplo:

```
bitset<8> b;
b.set(1);
```

```
b[3] = 1;
cout << b; // 00001010
```

Algoritmos en STL

Algoritmos básicos:

- `sort`, `reverse`, `count`, `find`, `min_element`, `max_element`, `accumulate`, `binary_search`, `lower_bound`, `upper_bound`.

Código ejemplo:

```
vector<int> v = {3, 1, 4};
sort(v.begin(), v.end());
```

Algoritmos avanzados:

- `nth_element`: coloca el k-ésimo elemento como si estuviera ordenado.
- `partial_sort`: ordena parcialmente.
- `next_permutation`: genera la siguiente permutación lexicográfica.
- `prev_permutation`: genera la anterior permutación lexicográfica.
- `rotate`: rota elementos.

Ejemplo:

```
vector<int> v = {1, 2, 3};
do {
    for (int x : v) cout << x << " ";
    cout << "\n";
} while (next_permutation(v.begin(), v.end()));
```

Este resumen reúne la teoría esencial, funciones clave y ejemplos prácticos para dominar la STL en C++. Ideal para resolver ejercicios, rendir exámenes y comprender la eficiencia de las estructuras y algoritmos.