Resumen: C++

José Ángel Olmedo Guevara

¿Qué es una estructura de datos?

Una estructura de datos es una forma de organizar y almacenar datos en memoria para que puedan ser utilizados de manera eficiente. En C++, la biblioteca estándar (STL) incluye una variedad de estructuras de datos útiles.

Estructuras comunes en C++ (STL)

- `vector`: Arreglo dinámico.
- `list`: Lista doblemente enlazada.
- `deque`: Cola doble, inserción al inicio o al final.
- `stack`: Pila (LIFO).
- `queue`: Cola (FIFO).
- `priority_queue`: Cola de prioridad.
- `set`, `multiset`: Conjuntos ordenados.
- `map`, `multimap`: Diccionarios ordenados.
- `unordered_set`, `unordered_map`: Versiones rápidas (hash).

Contenedores comunes en STL

- `std::array<T,N>` (arreglo de tamaño fijo)
- `std::vector<T>` (arreglo dinámico)
- `std::list<T>` (lista doblemente enlazada)
- `std::stack<T>` (pila LIFO)
- `std::queue<T>` (cola FIFO)

Ejemplos de uso

std::array

```cpp
#include <array>
#include <iostream>
using namespace std;

int main() {
    array<int,4> a = {1,2,3,4};
    cout << a[2] << endl; // 3
}
```

std::vector

```cpp
#include <vector>
#include <iostream>
using namespace std;

int main() {
    vector<int> v = {1,2,3};
    v.push_back(4);
    cout << v[1] << endl; // 2
}
```

std::list

```cpp
#include <list>
#include <iostream>
using namespace std;

int main() {
    list<string> L;
    L.push_back("hola");
    L.push_front("adios");
    for(auto &s : L) cout << s << " "; // adios hola
}
```

std::stack

```cpp
#include <stack>
#include <iostream>
using namespace std;

int main() {
    stack<int> S;
    S.push(10);
    S.push(20);
```

```
    cout << S.top() << endl; // 20
    S.pop();
    cout << S.top() << endl; // 10
}
```

std::queue

```
#include <queue>
#include <iostream>
using namespace std;

int main() {
    queue<int> Q;
    Q.push(1);
    Q.push(2);
    cout << Q.front() << endl; // 1
    Q.pop();
    cout << Q.front() << endl; // 2
}
```

Ventajas y Desventajas

std::array

Ventajas: - Tamaño fijo, sin sobrecarga de asignación dinámica. - Acceso O(1).
Desventajas: - No crece o reduce en tiempo de ejecución.

std::vector

Ventajas: - Tamaño dinámico, acceso O(1).
Desventajas: - Inserción/borrado en medio O(n). - Reasignación al crecer.

std::list

Ventajas: - Inserción y eliminación O(1) dado el iterador.
Desventajas: - Acceso por índice O(n). - Overhead por nodos enlazados.

std::stack

Ventajas: - LIFO, muy útil en algoritmos de retroceso y llamadas recursivas.
Desventajas: - Solo acceso al tope, sin iteración directa.

std::queue

Ventajas: - FIFO, ideal para procesamiento por orden de llegada.
Desventajas: - No acceso aleatorio, operaciones limitadas.

Comparación rápida

Necesitas...	**Usa**
Arreglo dinámico	`vector`
Lista enlazada	`list`
Insertar al inicio y final	`deque`
Acceder al último elemento rápido	`stack`
Procesar elementos por orden de llegada	`queue`
Ordenar por prioridad	`priority_queue`
Colección sin duplicados ordenada	`set`
Asociar claves con valores	`map`
Mayor eficiencia con hash	`unordered_map`

Notas adicionales

- STL hace uso de plantillas, lo que permite usar cualquier tipo de dato.
- Muchas estructuras permiten iteradores y algoritmos como `sort()`, `find()`, etc.

Paso por valor y paso por referencia en C++

- **Primitivos**:
  - Se pasan por valor por defecto.
  - Para modificar la variable original, se usan referencias (`int& x`) o punteros (`int* x`).

Ejemplo:

```
int a = 1;
int b = a;
a++;
std::cout << a << std::endl; // 2
std::cout << b << std::endl; // 1
```

- **Objetos**:
  - También se pasan por valor por defecto.
  - Se copia el objeto completamente.
  - Para evitar la copia, se usan referencias (`Objeto& x`) o punteros (`Objeto* x`).

Ejemplo por valor:

```
class Caja {
public:
    int valor;
};

Caja c1;
c1.valor = 5;

Caja c2 = c1;
c1.valor++;

std::cout << c1.valor << std::endl; // 6
std::cout << c2.valor << std::endl; // 5
```

Ejemplo con punteros:

```
Caja* c1 = new Caja();
c1->valor = 5;

Caja* c2 = c1;
c1->valor++;

std::cout << c1->valor << std::endl; // 6
std::cout << c2->valor << std::endl; // 6

delete c1;
```

Compilar y ejecutar un programa (omitir <>):

g++ <nombreArchivo.cpp> -o <nombreArchivo>
./<nombreArchivo>