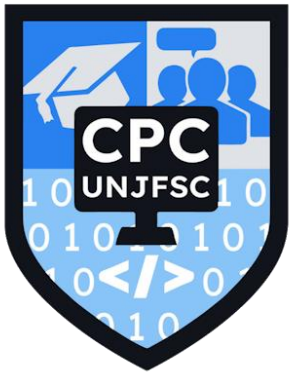
STL: Vectores

Instructor: Zahir la Rosa
13 de Julio, 2024



Standard Template Library (STL)

Introducción

La STL (Librería de Plantillas Estándar) en C++ es un grupo de clases y funciones de uso general que implementan **algoritmos** y **estructuras de datos** comunes. Fue diseñada para proporcionar herramientas reutilizables y eficientes que simplifican muchas tareas comunes de programación. La STL es una parte fundamental de la biblioteca estándar de C++ y se organiza en varios componentes clave:

1. **Contenedores**: Estructuras de datos que almacenan elementos. Los contenedores STL incluyen:
   - **Secuenciales:** como **vector**, **deque**, **list**, y **array**.
   - **Asociativos:** como **set**, **map**, **multiset**, y **multimap**.
   - **No ordenados:** como **unordered_set**, **unordered_map**, etc.

2. **Algoritmos**: Funciones genéricas que operan sobre contenedores. Ejemplos incluyen **sort**, **find**, **for_each**, **transform**, y muchos otros.

3. **Iteradores**: Nos permiten recorrer los elementos de un contenedor. Ejemplos incluyen **begin**, **end**, **rbegin** y **rend**.

4. **Adaptadores:** Ejemplos incluyen **stack**, **queue**, **priority_queue**.

Vectores (std::vector)

Introducción

Es una estructura de datos que representa un array dinámico. A diferencia de los arrays estáticos en C++, los vectores pueden cambiar de tamaño dinámicamente, permitiendo la adición y eliminación de elementos según sea necesario.

Declaración y Constructor

```
#include <vector>       // Es necesario incluir la librería vector
...
vector<int> v1;           // Vector vacío (sin tamaño definido)
vector<int> v2(10);       // Vector con 10 elementos, inicializados a 0
vector<int> v3(10, 5);    // Vector con 10 elementos, inicializados a 5
vector<int> v4 = {3, 2, 4, 4, 6};  // Vector inicializado con una lista de
valores
```

Referencia: https://cplusplus.com/reference/vector/

Capacidad

- **size:** Retorna el tamaño actual del vector.

```
vector<char> v = {'h', 'o', 'l', 'a'};
cout << v.size();  // Muestra 4
```

- **resize:** Cambia el tamaño actual del vector.

```
vector<int> v = {1, 2, 3};
v.resize(7);  // v = {1, 2, 3, 0, 0, 0, 0};

vector<int> v = {1, 2, 3};
v.resize(7, 4);  // v = {1, 2, 3, 4, 4, 4, 4};
```

- **empty:** Retorna un valor booleano identificando si el vector está vacío o no.

```
vector<int> v;
if (v.empty()) cout << "El vector está vacío.";
```

Acceso a Elementos

- **operator[]:** Accede a una posición específica de un vector.

```
vector<int> v = {1, 2, 4, 5, 6};
cout << v[3];  // Muestra 5
```

- **at:** Accede a una posición específica de un vector.

```
vector<int> v = {1, 2, 4, 5, 6};
cout << v.at(3);  // Muestra 5
```

- **front:** Retorna la primera posición del vector.

```
vector<int> v = {1, 2, 4, 5, 6};
cout << v.front();  // Muestra 1
```

- **back:** Retorna la útlima posición del vector.

```
vector<int> v = {1, 2, 4, 5, 6};
cout << v.back();  // Muestra 6
```

Agregar y eliminar elementos

- **push_back:** Añade un elemento al final del vector.

```
vector<int> v = {1, 2, 4, 5, 6};
v.push_back(7);  // v = {1, 2, 4, 5, 6, 7};
```

- **pop_back:** Elimina el elemento al final del vector.

```
vector<int> v = {1, 2, 4, 5, 6};
v.pop_back();  // v = {1, 2, 4, 5};
```

- **clear:** Elimina todos los elementos del vector.

```
vector<int> v = {1, 2, 4, 5, 6};
v.clear();  // v = {};
```

Iteradores

- **begin:** Retorna un iterador al primer elemento del vector.

```
vector<char> v = {'h', 'o', 'l', 'a'};
vector<char>::iterator inicial = v.begin();
cout << *inicial;  // Muestra h
```

- **end:** Retorna un iterador al final del vector que se encuentra fuera del mismo.

```
vector<char> v = {'h', 'o', 'l', 'a'};
vector<char>::iterator despues_del_final = v.end();
vector<char>::iterator final = prev(despues_del_final);
cout << *final;  // Muestra a
```

Iteradores Inversos

- **rbegin:** Retorna un iterador al último elemento del vector (iterar en sentido inverso)

```
vector<char> v = {'h', 'o', 'l', 'a'};
vector<char>::reverse_iterator final = v.rbegin();
cout << *final;  // Muestra a
```

- **rend:** Retorna un iterador al inicio del vector que se encuentra fuera del mismo.

```
vector<char> v = {'h', 'o', 'l', 'a'};
vector<char>::reverse_iterator despupes_del_inicial = v.rend();
vector<char>::reverse_iterator inicial = prev(despupes_del_inicial);
cout << *inicial;  // Muestra h
```

Recorrer Vectores

Usando índices:

```
vector<char> v = {'h', 'o', 'l', 'a'};
for (int i = 0; i < v.size(); i++) {
    cout << v[i] << " ";
}
// Muestra h o l a
```

Usando bucle for de rango:

```
vector<char> v = {'h', 'o', 'l', 'a'};
for (char& elemento : v) {
    cout << elemento << " ";
}
// Muestra h o l a
```

Usando iteradores:

```
vector<char> v = {'h', 'o', 'l', 'a'};
for (vector<char>::iterator it = v.begin(); it != v.end(); it++) {
    cout << *it << " ";
}
// Muestra h o l a
```

```
vector<char> v = {'h', 'o', 'l', 'a'};
for (auto it = v.begin(); it != v.end(); it++) {
    cout << *it << " ";
}
// Muestra h o l a
```