**CS2013 - Programación III**

**Práctica Calificada #2 (PC2)**

**2023 - 1**

Profesor: Rubén Rivas

---

## Librería Estándar – 6 puntos

Diseñar y desarrollar el template de función **join_containers** que permita unir varios contenedores (en una cantidad variada de contenedores) y que genere un contener que por default sea vector.
Los contenedores podrian ser: **list**, **vector**, **std::deque**, **std::forward_list**.

***Si se tiene los siguientes containers :***
*list lst = {1, 2, 3, 4, 5}*
*vector vec = {6, 7, 8, 9, 10, 11, 12}*
*deque deq = {13, 14}*

***Y si se llama:***
*result = **join_containers**(lst, vec, deq)*

***result contendria:***
*{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}*

### Casos de uso #1

```
vector v1 = {1, 2, 3};
list l1 = {4, 5, 6};
vector v2 = {7, 8, 9};
deque d1 = {10, 11, 12};
auto res = join_containers(v1, l1, v2, d1);
copy(begin(res), end(res), ostream_iterator<int>(cout, " "));
```

### Casos de uso #2

```
vector v = {1, 2, 3};
list<int> l;
forward_list f = {7, 8, 9};
auto res = join_containers(v, l, f);
copy(begin(res), end(res), ostream_iterator<int>(cout, " "));
```

**Complejidad Algorítmica – 6 puntos**

Dada la siguiente función determinar la complejidad algoritmica, incluir en su respuesta el procedimiento.

```
bool recursive_binary_search(
        std::vector<int>::iterator start,
        std::vector<int>::iterator end, int value) {
    if (start >= end) {
        return false;
    }

    std::vector<int>::iterator mid = start + (end - start) / 2;

    // Búsqueda binaria anidada
    if (std::binary_search(start, mid, value)) {
        return true;
    }

    // Recursión en la mitad superior
    return recursive_binary_search(mid + 1, end, value);
}
```

**<u>Patrones</u> – 6 puntos**

Dada la siguiente descripción, identifique que patrones podrías ser usados para la implementación y justifique seleccionando el párrafo donde se podría utilizar:

*Se desea desarrollar un sistema de seguimiento de flota de vehículos que tenga las siguientes características:*

- *Nuestro sistema necesita crear objetos que representen diferentes tipos de vehículos (camiones, coches, motos). Dependiendo de los datos de entrada, el sistema decidirá qué tipo de objeto vehículo crear.*
- *Aseguramos que sólo exista una única instancia de la base de datos de vehículos en todo el sistema para evitar conflictos de datos. Esta única instancia proporciona un punto de acceso global a la base de datos de vehículos.*
- *Los vehículos pueden tener diferentes tipos de sistemas de seguimiento, por ejemplo, GPS o GLONASS. Nuestro diseño separa el concepto de un vehículo de su sistema de seguimiento, permitiendo que el sistema de seguimiento de un vehículo pueda cambiarse en tiempo de ejecución.*
- *Podemos tratar un grupo de vehículos (una flota) de la misma manera que tratamos a un único vehículo. Esto nos permite realizar operaciones sobre la flota entera, como calcular la ubicación media de todos los vehículos en la flota, de la misma manera que encontraríamos la ubicación de un solo vehículo.*
- *El sistema recibe notificaciones cada vez que la ubicación de un vehículo cambia. Cuando un vehículo cambia su ubicación, todos los componentes del sistema que están registrados para recibir estas actualizaciones (como una interfaz de usuario que muestra la ubicación del vehículo en un mapa, o un sistema de alertas que se dispara si un vehículo sale de una zona designada) son notificados y pueden tomar acción.*

**Programación Concurrente – 6 puntos**

Escribir la función template **parallel_variance** que retorne la variancia de los elementos del contenedor.

$$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N}$$

Donde:

- $\boldsymbol{\sigma^2}$: variancia
- $\boldsymbol{\mu}$: promedio
- $\boldsymbol{N}$: cantidad de elementos
- $\boldsymbol{X_i}$: elemento $i$

El contenedor puede almacenar valores enteros o doubles.

El contenedor podrian ser cualquier tipo secuencial y su recorrido será a traves de sus interadores.

***Si se tiene el container con los siguientes valores:***
*cnt1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}*

***Y si se llama:***
*result = **parallel_variance**(begin(cnt1), end(cnt1))*

***result contendria:***
*11.916*

**Caso de uso #1**

```
int n = 0; cin >> n;
vector<double> vec(n);
for(auto& item: vec) cin >> item;
auto res = parallel_variance(begin(vec), end(vec));
cout << res;
```

**Caso de uso #2**

```
int n = 0; cin >> n;
vector<int> vec(n);
for(auto& item: vec) cin >> item;
auto res = parallel_variance(begin(vec), end(vec));
cout << res;
```

**Caso de uso #3**

```
int n = 0; cin >> n;
deque<int> deq(n);
for(auto& item: deq) cin >> item;
auto res = parallel_variance(begin(deq), end(deq));
cout << res;
```