**CS2013 - Programación III**

**Simulacro Práctica Calificada #2 (PC2)**

**2023 - 1**

Profesor: Rubén Rivas

---

## Librería Estándar – 6 puntos

Diseñar y desarrollar el template de función **find_uncommon_elements** que permita generar a partir de 2 contenedores un contenedor que contenga los valores no comunes entre ellos. Ejemplo:

```
container_1 = { 1, 4, 15, 19, 20, 11, 22, 1 };

container_2 = { 1, 1, 14, 5, 13, 19, 20, 22 };

container_3 = {4, 5, 11, 13, 14, 15};
```

**Casos de uso**

```
// Caso de uso #1
int n = 0;
cin >> n;
vector<int> vec1(n);
vector<int> vec2(n);
for(auto& item: vec1)
   cin >> item;
for(auto& item: vec2)
   cin >> item;
vector<int> res = find_uncommon_elements<vector>(vec1, vec2);
for(const auto& item: res)
   cout << item << " ";
```

```
// Caso de uso #2
deque<int> deq1(n);
list<int> lis1(n);
for(auto& item: deq1)
cin >> item;
for(auto& item: lis1)
   cin >> item;
deque<int> res = find_uncommon_elements<deque>(deq1, lis1);
for(const auto& item: res)
   cout << item << " ";
```

```
// Caso de uso #3
deque<int> deq1(n);
list<int> lis1(n);
for(auto& item: deq1)
cin >> item;
for(auto& item: lis1)
   cin >> item;
list<int> res = find_uncommon_elements<list>(deq1, lis1);
for(const auto& item: res)
   cout << item << " ";
```

## Complejidad Algorítmica – 6 puntos

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

```
void example_function(std::vector<int>& ns) {
    std::sort(begin(ns), end(ns));
    int n = size(ns);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n * n; j *= 2) {
            auto it = std::lower_bound(begin(ns), end(ns), j);
            if (it != end(ns) && *it == j) {
                std::cout << "Found " << j << std::endl;
            }
        }
    }
}
```

**Patrones – 6 puntos**

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

*Supongamos que estamos desarrollando un juego de aventuras en 2D. Queremos diseñar una solución que permita crear y gestionar diferentes tipos de personajes, enemigos, objetos y niveles de manera eficiente.*

*Nuestra solución se basa en un enfoque modular para la creación y gestión de los objetos del juego. Utilizamos una clase principal que actúa como un generador centralizado para crear los diversos elementos del juego, como los personajes, los enemigos y los objetos. Esto nos permite añadir nuevos elementos al juego de manera sencilla y adaptarnos a diferentes escenarios y requisitos.*

*Además, hemos separado la forma en que interactuamos con los objetos del juego de su implementación subyacente. Hemos definido interfaces comunes que especifican las acciones y características que se pueden aplicar a los objetos, como actualizar su estado, dibujarlos en la pantalla y manejar las colisiones. De esta manera, podemos tratar los diferentes objetos de juego de manera uniforme, independientemente de su tipo o función específica.*

*Para construir estructuras de juego más complejas, hemos utilizado una técnica de agrupación que nos permite combinar y organizar los objetos en una estructura jerárquica. Esto nos facilita la manipulación de múltiples objetos al mismo tiempo, así como la creación de niveles de juego con diferentes configuraciones y desafíos.*

.

**Programación Concurrente – 6 puntos**

Escribir la función template **find_in_collection** que debe retornar todos los valores en un vector que cumplan las condiciones definidas por el lambda utilizado como parametro.

**Caso de uso #1**

```
vector<int> vec = generate_container<int, 10000>(-100, 100);
auto result = find_in_collection(begin(vec), end(vec),
              [](auto value){ return value % 2 == 0; });
cout << result << endl;
```

**Caso de uso #2**

```
list<long int> lst = generate_container<long int, 10000>(-100, 100);
auto result = find_in_collection(begin(lst), end(lst),
              [](auto value){ return abs(value) > 50; });
cout << result << endl;
```