

CS2013 - Programación III
Práctica Calificada #2 (PC2)
2022 - 1

Profesor: Rubén Rivas

Complejidad Algorítmica - 6 puntos

Diseñar y desarrollar el template de función **remove_duplicates** que en un tiempo lineal $O(n)$ permita remover los valores duplicados de un contenedor.

```
template <typename Container>
void remove_duplicates(Container& cnt);
```

Casos de uso:

```
int n = 0;
cin >> n;
vector<int> vec(n);
for(auto& item: vec)
    cin >> item;
remove_duplicates(vec);
for(const auto& item: vec)
    cout << item << " ";
cout << endl;
```

Programación Concurrente - 10 puntos

Escribir la función template **find_max_triple_product** que reciba un contenedor secuencial genérico que procese de forma concurrente utilizando la función **get_number_of_threads**.

La función debe retornar el máximo valor obtenido por el producto de 3 de los valores contenidos en el contenedor, si hubiese 3 o menos valores la función debe retornar simplemente la multiplicación de todos. Para el caso de `std::list` sobrecargar la función template de modo que no sea concurrente.

Cantidad de thread:

```
int get_number_of_threads(int sz, int rng) {
    int max_threads = (sz + rng - 1) / rng;
    int k_threads = static_cast<int>(thread::hardware_concurrency());
    return min(k_threads != 0? k_threads: 2, max_threads);
}
```

Caso de uso #1

```
vector<long long int> vec;  
int n;  
while (cin >> n) vec.push_back(n);  
auto result = find_max_triple_product(vec);  
cout << result << endl;
```

Caso de uso #2

```
list<long double> lst;  
long double n;  
while (cin >> n) lst.push_back(n);  
auto result = find_max_triple_product(lst);  
cout << fixed << setprecision(2) << result << endl;
```

Patrones de Diseños - 4 puntos

1. Las operaciones aritméticas como: "**1 + 20**" o "**x + 20**" o "**x + 20 * y**", están conformados por términos u operaciones y usualmente conforma expresiones de esta forma:

```
<expresión> ::= <termino> | <operación>  
<operación> ::= <expresión> operador <expresión>  
<operación> ::= operador <expresión>
```

Donde en **operaciones aritméticas**:

```
<operación> ::= <sumar> | <restar> | ...  
<termino> ::= <numero> | <variable> | <constante> | ...
```

Este mismo patrón se utiliza para realizar **operaciones lógicas**, donde:

```
<operación> ::= <and> | <or> | ...  
<termino> ::= <bool> | <variable> | <constante> | ...
```

Se pide determine el tipo de patrón(es) de diseño que recomendaría y dibujar el/(los) diagrama(s) de clases que representen esos patrones seleccionados que permitan resolver algunos de los problemas de diseño de un programa que permita implementar la representación de operaciones aritméticas u operaciones lógicas.

Barranco, 3 de junio 2022.