
UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA

CS1103
Programación Orientada a Objetos 2
Unidad 2 - Semana 3 - Templates y Metaprogramming

Rubén Rivas

Varadic Templates

Ejercicio # 1

Generar la función **suma** realice la suma de todos sus parámetros.

Ejemplos:

```
auto total = suma(1, 2, 3, 4); // total == 10
```

```
auto total = suma(100, 200, 300); // total == 10
```

Ejercicio # 2

Generar la función **sum_product** de cantidad de parámetros variado y que realice la multiplicación de los parámetros pares con los parámetros impares y que devuelva la suma de todos estos productos. (¿Cómo se puede asegurar que los tipos siempre serán numericos?)

Ejemplos:

```
sum_product(10, 2, 5, 4, 10); // El resultado es:10*2 + 5*4 + 10 = 50
```

```
sum_product(3, 5, 2, 10); // El resultado es:3*5 + 2*10 = 35
```

Ejercicio # 3

Generar la función **compare** donde la cantidad de parámetros sea variada y que realice la comparación de los parámetros pares con los parámetros impares y devuelva true si todos los pares son iguales con los impares y false en caso contrario.

Ejemplos:

```
compare(10, 10, 20, 20, 10); //El resultado: 10==10 && 20==20 && 10 → true
```

```
compare(3, 5, 2, 10); // El resultado es:3==5 && 2==10 → false
```

Ejercicio # 4

Desarrollar una función **rand** similar a la funcion **randint** de Python de modo que genere un numero aleatorio en un rango de first y last pero de del tipo especificado en el parámetro de template.

Ejemplos

```
cout << rand<int>(1, 100); // Genera numero entero aleatorio entre 1 y 100
```

```
cout << rand<double>(1.5, 5); // Genera numero double aleatorio entre 1.5 y 5
```

Ejercicio # 5

Implementar la función print al estilo de python que permita imprimir un número variado de valores, que permita por medio del ultimo parametro bool imprimir en lineas separada los valores.

Ejemplos:

```
print("El resultado es:", 10); // El resultado es:10
```

```
print(true, "A", "B"); // A y B lo imprime en lineas separada
```

Ejercicio # 6

En la librería estándar existe una función no miembro que **size** que permite calcular el tamaño de cualquier contenedor, al estilo de esa función crear una función **size_container** que permita de forma genérica calcular la cantidad de cualquier contenedor incluso un arreglo estático.

Ejemplos:

```
int arr[] = {3, 4, 5, 6};
cout << "size: "<< size_container(arr); // size: 4
```

Ejercicio # 7

En **Python** existe la función **input** que permite ingresar de un valor del teclado y lo almacena en una variable, al estilo de **Python** desarrollar la función input en C++ utilizando templates. El valor por default almacenado será un **std::string.**

```
auto nombre = input("Ingrese su nombre: ");
auto nota_1 = input<int>("Ingrese la nota 1: ");
auto nota_2 = input<float>("Ingrese la nota 2: ");

cout << "Alumno: " << nombre << " Nota: " << (nota_1 + nota_2) / 2;
```

Function Templates

Ejercicio # 1

Existen 3 tipos de funciones de valor absoluto **`abs`**, **`labs`**, y **`fabs`**. Esas funciones se diferencia solamente por el tipo, seria mejor tener una sola función template para el valor absoluto que se llame **`absolute`**. de modo que pueda ser usada por tipos como int, long y double.

Ejercicio # 6

Escribir una función template **`rotate_range`** que permite rotar los valores de un contenedor una cantidad determinada, esta función recibe 2 parametros un contenedor, y un valor entero, en caso sea positivo los valores rotará hacia la derecha, en caso negativo los valores rotaran a la izquierda:

```
// ejemplo # 1:
vector<int> v1 = {1, 2, 3, 4, 5, 6};
auto v3 =rotate_range(v1, 2);           // {5, 6, 1, 2, 3, 4}

// ejemplo # 2:
list<int> v1 = {10, 20, 30, 40, 50};
auto v3 = rotate_range(v1, -2);         // {30, 40, 50, 10, 20}
```

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Ejercicio # 7

La librería estándar de C++ brinda un contenedor simple conocido como **std::pair**, que permite almacenar solo 2 valores de diferentes. Su sintaxis es **std::pair<type1, type2>** ejemplo:

```
std::pair<int, string> p1 = {1321, "Jose Perez"};
```

Usualmente este tipo de container que se utiliza en **std::maps** o para 2 valores de una función, uno de las desventajas es que para acceder a los dos valores **std::pair** se utilizan 2 campos (**first & second**) que usualmente no tienen un significado en el contexto de un problema.

```
std::cout << p1.first <<  " " << p1.second;
```

De otro lado lenguajes como python brindan un mecanismo conocido como **unpack** que permite asignar valores de un contenedor a variable con nombres significativos.

Desarrollar un function template que permite asignar los valores de un **std::pair** directamente a variables con nombres significativos, Ejemplo.

```
int key; string name;
unpack(key, name) = p1;
std::cout << key <<  " " << name;
```

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Ejercicio # 9

Escribir un function template que permita leer un número de parámetros variados y que genere a partir de esa listas de parámetros un vector (por defecto) o un contenedor seleccionado.
Ejemplo:

```
auto c1 = generar_contenedor(1, 2, 3, 4);
for (const auto& item: c1)
        std::cout << item << " ";
std::cout << endl;

auto c2 = generar_contenedor<list>(10, 20, 30, 40);
for (const auto& item: c1)
        std::cout << item << " ";
std::cout << endl;
```

Ejercicio # 10

Diseñar un function template que permita leer un número variado de contenedores de un mismo tipo y que retorne el tamaño mínimo.

Ejemplo:

```
vector<int> v1 = { 11 };
vector<int> v2 = { 21, 22, 23, 1, 2 };
vector<int> v3 = { 31, 32, 33, 4 };
cout << min_size(v2, v3) << endl;       // 4
cout << min_size(v1, v2, v3) << endl;   // 1
```

Ejercicio # 12

Escribir un template para la versión iterativa de la busqueda binaria.

Ejercicio # 13

Escribir un template para la versión recursiva de la busqueda binaria.

Class Templates

Ejercicio # 1

Definir una estructura genérica Linked List que cuente con los siguientes métodos básicos:

```
void push_front(T item); // Agrega un item en el frente
void push_back(T item);  // Agrega un item al final
T pop_front();           // Remueve un item del frente y lo retorna
T pop_back();            // Remueve un item del final y lo retorna
T front();               // Retorna el valor del frente
T back();                // Retorna el valor del final
```

Ejercicio # 2

Definir una estructura genérica max-heap que cuente con los siguientes métodos básicos:

```
T find_max();              // Busca el valor mayor
void insert(T item);       // Inserta un item al heap
void delete();             // Remueve un item del final y lo retorna
void print();              // lista todos los elementos del heap
```

Ejercicio # 3

Python cuenta el generador **range** que permite generar una secuencia de valores en un rango definido por 3 parámetros (start, stop, step). Generar en C++ un template que permita emular **range**.

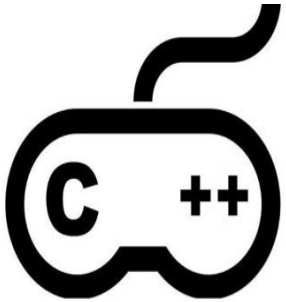
```
for (const auto& item: range(10)) // 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  cout << item << " ";
cout << endl;
for (const auto& item: range(2, 12)) // 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
  cout << item << " ";
cout << endl;
for (const auto& item: range(3, 20, 3)) // 3, 6, 9, 12, 15, 18
  cout << item << " ";
cout << endl;
```

Explorando lo aprendido

- ¿Qué es el variadic template?
- ¿Por qué es útil?
- ¿Que es metaprogramación?
- ¿Cuales son los mecanismos de la metaprogramación?

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Bibliografía:



- **C++ Templates, The Complete Guide**; 2018; David Vandevoorde, Nicolai M. Josuttis, Douglas Gregor
- **C++ Primer, Fifth Edition; 2013**; Stanley B. Lippman, Josée Lajoie, Barbara E. Moo