Práctica III

*Apartado 1 [**1 punto**]*: implementar las siguientes **declaraciones** de plantilla:

- Función que recibe 2 parámetros de entrada plantilla distintos, y devuelve un tipo de la misma plantilla que una de las entradas
- Función que reciba como parámetro de entrada un vector de tipos templatizados y devuelva un tipo templatizado.

*Apartado 2 [**1 punto**]*: implementar las siguientes clases templatizadas:

- Clase con una variable miembro tipo plantilla y los getter/setter correspondientes.
- Clase que toma dos parámetros diferentes como plantilla. Uno será el argumento del constructor y otro como valor de retorno de su método getter.

*Apartado 3 [**2.5 puntos**]*: implementar una clase de etiqueta Puntero, dicha clase debe simular el comportamiento de un puntero. Para ello debe encapsular una variable puntero templatizada que debe ser inicializada con reserva dinámica de memoria en su constructor, dicha reserva dinámica debe ser liberada en el método que consideréis oportuno. Además debe disponer de un método que devuelva la posición de memoria y debe sobrecargar en su interior el operador * de manera que un objeto instanciado de esta clase permita realizar:

```
std::cout<<*miPuntero<<std::endl;
*miPuntero="el valor que sea";
```

*Apartado 4 [**2.5 puntos**]*: templatizar una clase genérica contenedor que encapsule un std::vector<T>. Dicha clase debe incorporar los constructores y métodos getter/setter que consideréis oportunos, así como un método ordenar que devuelva un std::vector<T> con los elementos del vector templatizado encapsulado en la clase ordenados mediante el método burbuja. Esta clase templatizada debe permitir al menos ordenar números enteros, números decimales y personas por edad.

Ya sabéis que las estructuras en C++ son clases con todos sus miembros públicos. Debéis utilizar la siguiente declaración de persona, completando la definición de sus métodos

```
struct persona{
        int edad=0;
        string nombre;

        persona(int e,string n);
        bool operator >(persona const &per)
};
```

*Apartado 5 [**3 puntos**]*: templatizar una clase pila de manera que podamos utilizarla con distintos tipos de datos. Para ello vamos a encapsular un vector<T> e implementar los métodos push, pop, tos e imprimir

- Método push: debe añadir un elemento en nuestra pila.
- Método pop: debe eliminar un elemento en nuestra pila
- Método tos: debe obtener el último elemento de la pila
- Método imprimir: debe mostrar por terminal el contenido de la pila.

Debéis probar vuestra pila en el programa principal, para ello vais a declarar :

- Una estructura templatizada nodo que encapsule:
  - std::string nombre;
  - int IDnodo;
  - T infoNodo;
- Una estructura color que encapsule:
  - std::string nombre;
  - int R;
  - int G;
  - int B;

Y vais a sobrecargar el operador << para poder mostrar por terminal objetos de tipo nodo. Recordad que nodo es un template, por lo que deberéis templatizar también la sobrecarga para indicar en el parámetro de entrada donde pasáis en nodo a mostrar el tipo templatizado que corresponda.

Con todo esto podréis en vuestro programa principal crear un objeto pilaColores y apilar, desapilar, mostrar TOS, etc de la manera:

```
Pila<nodo<colores>> pilaColores;

pilaColores.push({"Nodo 1",101,{"Rojo",255,0,0}});
```

Como recordatorio: una pila es un TAD lineal LIFO (Last Input First Output) cuyo funcionamiento se asemeja por ejemplo a una pila de platos. Vamos colocando información (nodos o platos :) uno encima de otro y cuando queremos recuperar información de nuestro stack siempre lo hacemos por la parte alta de la pila (TOS, Top Of Stack)