1. Crear una versión template de la función **`Fibonacci()`** para que retorne **`long`**, **`float`**, etc, en lugar de siempre **`int`**s.
2. Crear una clase template que implemente una lista simplemente ligada que acepte elementos de cualquier tipo derivado de una clase **`Link`** que contiene la información necesaria para linkar los elementos. Esto se llama una lista intrusiva.
3. Utilizando la lista del ejercicio anterior, crear una clase de lista ligada que acepte elementos de cualquier tipo.
4. Terminar de implementar la clase template **`String`**.
5. Escribir un programa que lea pares **`(key, value)`** e imprima la suma de **`value`**s para cada **`key`** distinto.
6. Crear una clase **`UniquePtr`** con comportamiento similar a la de la biblioteca standard.
   - Se inicializa con un puntero.
   - El puntero será **`delete`**d al salir el objeto de scope.
   - No puede ser copiado, pero puede ser movido.

   Comparar con el **`unique_ptr`** de la biblioteca standard (Stroustrup 34.3.1).
7. Hacer un programa que ordene un array de objetos de alguna **`struct`** usando la función **`qsort()`** de C y la **`sort()`** de C++. Crear la función de comparación para C y una clase similar para C++. Compilar con opciones de optimización ejecutar y comparar los tiempos de ejecución entre C y C++.
8. Crear una clase **`CheckedArray`** que verifique los límites de los índices.
9. Crear una clase **`CheckedVector`** derivada de **`std::vector`** que verifique los límites de los índices.

10. Crear una clase template vector<T> mínima, capaz de almacenar contenedores de instancias de T. Genera una especialización que esté optimizado para un vector de bool (para que cada bool ocupe un bit).