UNIVERSIDAD NEBRIJA

ALUMNO:

Asignatura:	Programación II (Grado de Ingeniería Informática – 1er curso).				
Curso: **2019/2020**		Examen:	Parcial	Fecha: 09-04-2020	
Semestre:	Segundo	Convocatoria:	Ordinaria		

Parte Teórica (2 puntos)

- **Se debe entregar en papel.**
- **La parte teórica necesita ser aprobada para que se corrija la parte práctica. En caso de suspender la parte teórica la nota total del examen será de 0 puntos.**
- **Cada respuesta correcta suma 1 punto.**
- **Cada respuesta incorrecta resta 0.25 puntos.**
- **En caso de aprobar esta parte se tendrán 2 puntos sobre el total de 10 puntos del examen.**

Pregunta 1

```
#include <iostream>

class Foo{
public:
 Foo();
 void print();
private:
 int a;
 int b;
};

Foo::Foo(){
 a=5;
 b=7;
 std::cout << b << "\n";
}

void Foo::print(){
 std::cout << "a: " << a << "\n";
 std::cout << "b: " << b << "\n";
}
int main() {
 Foo fi;
 fi.print();
 return 0;
}
```

¿Qué mostrará el programa por pantalla?

UNIVERSIDAD NEBRIJA

ALUMNO:

Asignatura: Programación II (Grado de Ingeniería Informática – 1er curso).

Curso: **2019/2020** Examen: Parcial Fecha: 09-04-2020
Semestre: Segundo Convocatoria: Ordinaria

Pregunta 2

```
#include <iostream>

class Foo{
 friend bool operator>(Foo const & a,
Foo const &b);
public:
 Foo();
 void print();
private:
 int a;
 int b;
};

Foo::Foo(){
 a=5;
 b=7;
}

void Foo::print(){
 std::cout << "a: " << a << "\n";
 std::cout << "b: " << b << "\n";
}

bool operator>(Foo const & a, Foo const
&b){
 return a.b > b.a;
}

int main() {
 Foo fi;
 Foo fo;
 if(fi > fo) fi.print();
 else std::cout << "nada";
 return 0;
}
```

¿Qué mostrará el programa por pantalla?

UNIVERSIDAD NEBRIJA

ALUMNO:

Asignatura:	Programación II (Grado de Ingeniería Informática – 1er curso).		
Curso: **2019/2020**	Examen:	Parcial	Fecha: 09-04-2020
Semestre: Segundo	Convocatoria:	Ordinaria	

Pregunta 3

```
#include <iostream>

template<class A>
A foo(A b){
   if(b > 3) return 3*b;
   else throw std::string{"error"};
}


int main() {
   try{
       std::cout << foo(5) << "\n";
       std::cout << foo(2.2) << "\n";
       std::cout << foo(6) << "\n";
   }catch(std::string e){
       std::cout << e << "\n";
   }
   std::cout << "fin\n";
}
```

¿Qué mostrará el programa por pantalla?

UNIVERSIDAD NEBRIJA

ALUMNO:

Asignatura:	Programación II (Grado de Ingeniería Informática – 1er curso).			
Curso: **2019/2020**		Examen:	Parcial	Fecha: 09-04-2020
Semestre:	Segundo	Convocatoria:	Ordinaria	

Pregunta 4

```
#include <iostream>
using namespace std;

template<typename T>
T foo(T in)
{
 return in;
}

template<typename T>
T foo(T* in)
{
 return 1;
}

int main()
{
 int dato=8;
 std::cout << foo<int>(&dato) << "\n";
 return 0;
}
```

¿Qué mostrará el programa por pantalla?

UNIVERSIDAD NEBRIJA

ALUMNO:

Asignatura:	Programación II (Grado de Ingeniería Informática – 1er curso).				
Curso: **2019/2020**		Examen:	Parcial	Fecha: 09-04-2020	
Semestre:	Segundo	Convocatoria:	Ordinaria		

Parte Práctica (8 puntos)

La puntuación de cada parte será todo o nada. Si está tal cual se pide será la puntuación indicada, en caso contrario será 0.

Se desea realizar una clase templatizada **Lista.** Esta clase debe permitir tener listas de cualquier tipo de elementos (es decir, vamos a hacer una clase Lista que hace lo mismo que *std::vector*)

Por ejemplo

```
Lista<int> listaEnteros;
Lista<std::string> ListaStrings;
```

La clase **Lista** debe tener las siguientes funciones:

- **push_back**: para añadir un nuevo elemento **(1 punto)**.
- **forEach**: para realizar una operación sobre cada uno de los elementos de la **Lista** sin modificarlos a través de una función lambda. La función lambda debe recibir el elemento y su índice en la lista. **(1.5 punto)**.
- **find**: para buscar la primera coincidencia según un criterio de búsqueda pasado a través de una función lambda. La función lambda debe recibir el elemento y su índice en la lista. Si no encuentra ninguna coincidencia lanzará una excepción **(1.5 punto)**.
- **filter**: para buscar todas las coincidencias según un criterio de búsqueda pasado a través de una función lambda (devuelve una Lista). La función lambda debe recibir el elemento y su índice en la lista. **(1.5 punto)**
- **map**: para aplicar una transformación a todos los elementos a través de una función lambda (devuelve una Lista). La función lambda debe recibir el elemento y su índice en la lista. **(1.5 punto)**

Además se debe sobrecargar el **operador <<** para poder mostrar por pantalla todos los elementos de la lista a través de un cout. **(1 punto)**