Examen de **Tecnología de Programación** **26 de Junio de 2017**
Grado en Ingeniería Informática.
Escuela de Ingeniería y Arquitectura.
Universidad de Zaragoza

Duración total del examen: ***2 horas 30 minutos***

NOTA RECORDATORIA: La evaluación final consta de las siguientes pruebas:

- Prueba escrita → 60% de la nota.
- Prácticas, la máxima nota entre las prácticas entregadas y defendidas durante el curso y el examen de prácticas → 40% de la nota.

Ejercicio 1 [2 puntos]

A continuación verás dos programas realizados en el lenguaje C++. Por cada uno de los programas podrá haber más de un fichero: el nombre del fichero aparecerá encima de su correspondiente código. Asume que todos los ficheros están en la misma carpeta. Por brevedad y concisión, se han utilizado punteros estándar y los programas podrían tener *memory leaks*, que deberán ser ignorados pues no afectan a la respuesta del ejercicio. De cada uno de los programas, responde a las siguientes preguntas:

- Se compila con el comando `g++ main.cc -std=c++11` ¿Compilaría correctamente o daría algún error? Ignora errores tipográficos, y asume un compilador razonablemente moderno con soporte completo para el estándar C++11.
- En caso de que en tu respuesta anterior indiques que no compila, ¿dónde ocurre el error de compilación? ¿por qué no compila?
- En caso de que en tu primera respuesta indiques que compila, ¿qué sacaría el ejecutable por pantalla? No es necesario que lo justifiques.

Programa A

foo.h

```
#include <iostream>
class Foo {
protected:
   int t;
public:
   Foo(int t) :
      t(t) { }

  void sil()
 { (this->t)+=1; }

 virtual void
        output() const
 {  std::cout<<
      (this→t)<<
         std::endl; }
};
```

bar.h

```
#include "foo.h"
class Bar : public Foo {


public:
   Bar(int t) :
      Foo(2*t) { }

   void sil()
   {  (this->t)*=2;    }

 void output() const override
 {  std::cout<<
      "|"<<(this→t)<<"|"<<
      std::endl; }
};
```

main.cc

```
#include "bar.h"
int main(int argc, char** argv) {
   Foo* foo = new Foo(1);
   foo->sil(); foo->sil();
   foo->output();

   Bar* bar = new Bar(2);
   bar->sil(); bar->sil();
   bar->output();

   Foo* foobar = new Bar(3);
   foobar->sil(); foobar->sil();
   foobar->output();
}
```

Duración total del examen: ***2 horas 30 minutos***

Programa B

foo.h

```
template<typename T>
class Foo {
protected:
 T t;
public:
 Foo(const T& t) :
     t(t) { }
 T meta() const
  {  return t;   }
};
```

bar.h

```
#include "foo.h"
template<typename T>
class Bar {
 Foo<T>* foo;
public:
 Bar(Foo<T>* foo) :
    foo(foo) { }
 T tolo() const
 { return foo->meta(); }
};
```

main.cc

```
#include "bar.h"
#include <iostream>
int main(int argc, char** argv) {
 Bar<int> ibar(new Foo<int>(42));
 std::cout<<ibar.tolo()<<std::endl;

 Bar<float> fbar(new Foo<float>(4.8f));
 std::cout<<fbar.tolo()<<std::endl;

 Bar<double> dbar(new Foo<int>(1024));
 std::cout<<dbar.tolo()<<std::endl;
}
```

Ejercicio 2 [3 puntos]

La **programación genérica** es un mecanismo muy útil de reutilización de código, que en lenguajes orientados a objetos puede aplicarse tanto a clases como a métodos. Sobre este tema realiza las siguientes tareas:

(a) **Define** el concepto de **programación genérica**. Haz que tu definición sea independiente de un lenguaje concreto.

(b) Siendo coherente con tu definición el (a) y dentro del paradigma de orientación a objetos, **define** los conceptos de **clase genérica** y **método genérico**.

(c) **Implementa** un lenguaje orientado a objetos a tu elección una **clase genérica** que tenga (entre otros elementos que consideres necesarios) un **método genérico**. El ejemplo debe de ser completo, es decir: la clase debe incluir todos los atributos y métodos para poder, potencialmente, compilar.

(d) **Desarrolla un programa principal** que utilice la clase anterior, instanciando tanto la clase como el método correspondiente.

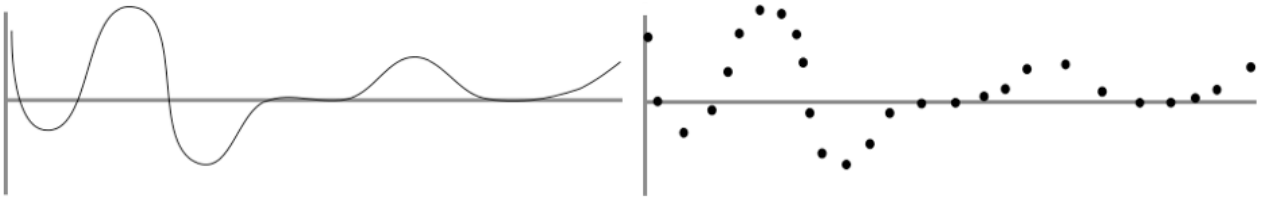
(e) **Explica** sobre tu código en (c) y (d) tanto tu definición de la programación genérica (a) como tus definiciones concretas para clase y método genérico (b).

Duración total del examen: ***2 horas 30 minutos***

Ejercicio 3 [2 puntos]

La parte izquierda de la figura muestra una curva, un tanto extraña, del comportamiento de cierto **proceso químico**. La curva no es realmente conocida, y para obtener información sobre ella se muestrean sus valores durante un cierto periodo de tiempo. Esto hace que se tenga una visión discreta de la curva, como se muestra en la parte derecha de la figura. Los puntos corresponden a los valores que realmente se han tomado durante el tiempo que ha durado el experimento. Los valores medidos son números reales y están guardados en una lista de Haskell.

Una medida de seguridad del proceso químico en un intervalo de tiempo se caracteriza por determinar el **número de veces que la curva del proceso toma el valor 0.0**; la curva pasará por 0.0 si cambia de signo entre dos muestras consecutivas o si toma el valor 0.0. Se cuenta un solo paso por cero si hay dos o más muestras consecutivas iguales a 0.0. En el ejemplo mostrado a continuación la curva pasa exactamente 5 veces por 0.



L = [8.0,0.0,-2.0,-0.2,3.0,8.0,10.0,9.5,8.0,6.3,-0.3,-5.0,-6.0,-4.2,-0.3,0.0,0.0,0.3,0.8,3.5,4.0,0.5,0.0,0.0,0.2,0.5,4.0]

Se pide: diseña en Haskell la función **`curveZeroes :: [Double]` → `Integer`** que dada una lista de números reales devuelva el número de veces que la curva representada pasa por 0. Puedes definirte otras funciones si lo consideras necesario. Se valorará negativamente el código innecesariamente extenso.

Duración total del examen: ***2 horas 30 minutos***

Ejercicio 4 [3 puntos]

Un camping factura a sus usuarios dependiendo de los elementos con que ocupen las parcelas que ofrece. Los distintos Elementos se representan por las siguientes categorías:

- Adulto
- Niño
- Coche
- Tienda
- Caravana

Los distintos elementos se facturan de la siguiente forma:

- cada Persona: 4 € (Adultos y Niños)
- cada Vehículo: 5 € (Coches)
- cada Alojamiento: 6 € (Tiendas y Caravanas)

Por razones de seguridad el camping suministra unos avisadores que emiten una alarma en caso de emergencia (por ejemplo para abandonar el camping en caso de incendio), uno para cada Adulto (en el que se enciende una luz parpadeante) y uno para cada Alojamiento (que emite un aviso acústico). En caso de alarma, se emite un aviso que reciben los Adultos y los Alojamientos.

Cada **Parcela** guarda la información de los elementos que contiene. La información del estado del **Camping** en un momento determinado se almacena como un conjunto de Parcelas.

En el lenguaje orientado a objetos que prefieras (C++ o Java), se pide:

- Definir los tipos de datos para representar la información sobre el estado de ocupación del camping (clases **Camping**, **Parcela** y las demás que consideres necesarias)
- Definir un método que permita calcular el precio diario de una **Parcela**, según los elementos que la ocupen, y otro que calcule los ingresos totales del **Camping** en un día.
- Definir los métodos necesarios que permitan que el **Camping** avise a todo el mundo en caso de emergencia..

Evaluá las modificaciones necesarias a tu solución si ahora se quisieran añadir vehículos especiales como **Autocaravanas** (vehículo con alojamiento incorporado) o **Microbuses** (vehículo mas grande que paga el doble).