Examen de **Tecnología de Programación** **14 de Septiembre 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 (máxima nota entre las prácticas entregadas durante el curso y el práctico) → 40% de la nota.

Ejercicio 1 [2 puntos]

A continuación verás dos programas realizados en el lenguaje C++. Por cada uno de los programas hay más de un fichero, cuyo nombre aparece encima de su correspondiente código. Asume que todos los ficheros están en la misma carpeta. De cada programa, responde a:

- Se compila con el comando `g++ -std=c++11 main.cc` (con gcc versión > 4.8.1) ¿Compilaría correctamente o daría algún error? Ignora errores tipográficos.
- 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

NOTA: Este programa, por concisión, podría tener *memory leaks*. Ignóralos, no afectan a tu respuesta.

foo.h

```
#include <iostream>
class Foo {
  int t;
public:
  Foo(int t) : t(t) { }
  virtual void sil()
  { (this->t) += 1; }
  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() override
  {  (this->t) *=2 ;    }
  void output() const
  {  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();           }
```

Programa B

NOTA: El operador += del tipo de dato std::string concatena el la cadena de texto que se pasa como parámetro al final.

foo.h

```
template<typename T>
class Foo {
public:
 Foo() { }
 virtual T
transform(const T& t)
  const = 0;
};
```

bar.h

```
#include "foo.h"
#include <string>
class Bar : public Foo<int>,
        public Foo<std::string> {
 int times;
public:
 Bar(int t) : times(t) { }
 int transform(const int& i)
                  const override
 {    return i*times;          }
  std::string transform(const
   std::string& s) const override
 {
```

main.cc

```
#include "bar.h"
#include <iostream>

template<typename T>
void transformprint(const T& t,
           const Foo<T>& foo) {
 std::cout<<foo.transform(t)
                  <<std::endl;
}

int main(int argc,
                char** argv) {
 Bar bar(3);
```

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

```
  std::string r = s;
  for (int i=1;i<times;++i) r+=s;
  return r;
}                              };
```

```
transformprint<int>(3, bar);
transformprint<std::string>
    (std::string("la"), bar);
}
```

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

Ejercicio 2 [3 puntos]

Diferentes lenguajes de programación permiten definir **funciones (o métodos) genéricos**. A continuación te presentamos tres ejemplos en tres lenguajes de programación diferentes:

Java

```
class Min {
public static <T>
   T min(T t1, T t2)
   {
      if (t1<t2) return t1;
      else       return t2;
   }
}
```

C++

```
template<typename T>
T min(T t1, T t2)
{
   if (t1<t2) return t1;
   else       return t2;
}
```

Haskell

```
min :: t -> t -> t
min a b
   | a < b     = a
   | otherwise = b
```

Sobre este tema (y basado en los ejemplos anteriores) realiza las siguientes tareas:

(a) **Define** formalmente el concepto de función (o método) genérico. Tu definición deberá ser independiente de un lenguaje específico.

(b) Las implementaciones de las funciones genéricas varían entre diferentes lenguajes. De los tres ejemplos que te presentamos **¿cuáles compilan y cuáles no?** Ten en cuenta tan sólo la función genérica presentada y no su instanciación posterior. Ignora errores tipográficos y la posible definición previa de la función **`min`** en cada lenguaje.

(c) **Razona y justifica** por qué compila este código tan similar en unos casos sí y en otros no. Relaciónalo con la implementación de las funciones genéricas en cada lenguaje.

(d) Elije un lenguaje en el que la función genérica no compile y **haz las modificaciones** que consideres sobre el código **para que compile** correctamente. Replica todo el (breve) código de la función original con las modificaciones hechas.

(e) **Explica brevemente** por qué los cambios realizados en (d) permiten que el código compile. Relaciona tu explicación con tu respuesta del apartado (c)

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

Ejercicio 3 [3 puntos]

Se dice que un número natural es poderoso si, **para todo** divisor suyo *p* que es primo, se cumple que *p* al cuadrado (p^2) también es divisor del número.

Ejemplos:

- El número 36 es un número poderoso, ya que, de todos sus divisores, los únicos que son primos son 2 y 3, y se cumple que 4 (2^2) y 9 (3^2) también son divisores de 36.
- Los números primos, por definición, no son números poderosos.
- El número 1 sí que es un número poderoso.

Se pide: Implementa en Haskell las siguientes funciones:

- **`poderosos`** – Sin ningún parámetro, devuelve una lista infinita con todos los números poderosos.
- **`npoderosos n`** – Utilizando la función anterior, devuelve los **`n`** primeros números poderosos.
- **`poderososHasta mx`** – Utilizando la primera función, devuelve todos los números poderosos hasta un máximo que se pasa como parámetro **`mx`**.

Se permite la implementación de otras funciones que sirvan para resolver el problema. Además del correcto funcionamiento de las funciones, se valorará la concisión de la solución.

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

Ejercicio 4 [2 puntos]

Una empresa puede tener distintos tipos de personal: un único director, encargados, empleados fijos, empleados por horas y becarios. La empresa contrata empleados, paga su sueldo a final de mes, e incluso les da la bienvenida por las mañanas.

En el lenguaje orientado a objetos de tu elección (C++ o Java), realiza las siguientes tareas:

- Define la clase **`Empresa`**, que almacena todos sus empleados.
- Define clases para representar los distintos tipos de **empleados**, teniendo en cuenta que todos tienen información común como nombre y DNI, a todos se les puede saludar mediante un método denominado **`hola()`**, pero luego cada uno presenta ciertas particularidades en cuanto al pago de su salario (que se describen mas adelante).
- Implementa en la clase **`Empresa`** el método **`contratar(… nuevoEmpleado)`**, que añade un nuevo empleado a la empresa. Ten en cuenta que sólo puede haber un director.
- Implementa en la clase **`Empresa`** el método **`saludar()`**, que saluda por la mañana a todos los empleados.
- Implementa en la clase **`Empresa`** el método **`pagar()`**, que paga el salario mensual a todos los empleados a los que les corresponde, de la siguiente forma:
  - El director no cobra un sueldo, vive de las rentas de los beneficios.
  - Los empleados (tanto fijos como por horas) cobran su sueldo mediante un método **`pagar(… cantidad)`**.
  - Los encargados y empleados fijos cobran un sueldo fijo que se puede saber mediante una llamada a su método **`sueldo()`**.
  - Los empleados por horas cobran dependiendo de las horas trabajadas y el valor de la hora de trabajo. Ambas características se pueden consultar mediante sendos métodos **`horasTrabajadas()`** y **`precioHora()`**.
  - Los becarios no cobran, obviamente.
- Para las clases que representan diferentes tipos de empleados no es necesario implementar los métodos descritos arriba, pero sí declararlos (sólo su interfaz) en la definición de la clase correspondiente.

Notas:

- Se valorará positivamente la reutilización de código y la utilización a discreción de las posibilidades de cada lenguaje.
- Se valorará negativamente la inclusión de código inútil o sobrante.