Examen de **Tecnología de Programación** **13 de Septiembre de 2016**
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 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. 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, ¿por qué no compila? ¿dónde daría el error de compilación?
- 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

```
template<typename T>
class Foo {
   T t;

public:
  Foo(const T& _t) :
         t(_t) { }

  void mar(const T& t2)
   {  t=t+t2;   }

   T sil() const
   {  return t;   }
};
```

bar.h

```
#include "foo.h"

class Bar : public Foo<int>
{
public:
   Bar() : Foo<int>(0) { }
};
```

main.cc

```
#include "bar.h"
#include <iostream>
int main(int argc, char** argv) {
 Bar bar;
 bar.mar(3); bar.mar(2);
 std::cout<<bar.sil()<<std::endl;

 Foo<double> ffoo(0.3);
 ffoo.mar(3.1); ffoo.mar(1.6);
 std::cout<<ffoo.sil()<<std::endl;

 Foo<const char*> cfoo("ba");
 cfoo.mar("tm"); cfoo.mar("an");
 std::cout<<cfoo.sil()<<std::endl;
}
```

Programa B

NOTA: El programa B podría tener *memory leaks*. Ignorarlos, no afectarían al resultado.

foo.h

```
class Foo {
public:
  Foo() { }

 virtual int value()
const = 0;
};
```

bar.h

```
#include "foo.h"
class Bar : public Foo {
  Foo* foo;
public:
  Bar(Foo* f) : foo(f) { }
  int value() const override
  { if (foo == nullptr)
       return 0;
  else return foo->value()+1;
 }                          };
```

main.cc

```
#include "bar.h"
#include <iostream>
int main(int argc, char** argv) {
  Foo* f = new Bar(nullptr);
  std::cout<<f->value()<<std::endl;
  f = new Bar(new Bar(nullptr));
  std::cout<<f→value()<<std::endl;
  f = new Bar(new Foo());
  std::cout<<f->value()<<std::endl;
}
```

Ejercicio 2 [3 puntos]

Numerosos lenguajes de programación modernos de alto nivel permiten diferentes mecanismos para lograr el **polimorfismo**. Diferentes paradigmas de programación ofrecen diferentes mecanismos para lograrlo. Sobre este tema, realiza las siguientes areas:

(a) **Define** con tus propias palabras el concepto de polimorfismo (dí **qué es**, no cuándo se utiliza ni para qué sirve).

(b) Los lenguajes orientados a objetos vistos en la asignatura (C++ y Java) permiten polimorfismo. **Explica** brevemente qué mecanismo de dichos lenguajes permite **polimorfismo por inclusión**, y qué otro mecanismo permite **polimorfismo paramétrico**, y cómo lo permiten.

(c) **Ilustra** mediante un **ejemplo** en un lenguaje orientado a objetos a tu elección (C++ o Java) simultáneamente los dos tipos de polimorfismo identificados en (b). Dicho ejemplo deberá ser lo más pequeño posible (preferiblemente con solo dos o tres clases involucradas) y deberá incluir un pequeño programa principal.

(d) **Justifica** dónde aparecen las modalidades de polimorfismo descritas de (b) en tu ejemplo de (c).

(e) De los dos tipos de polimorfismo de (b), sólo uno de ellos está representado mediante un mecanismo de **Haskell** (programación funcional). **¿Cuál?**

(f) **Ilustra** mediante un **ejemplo** en Haskell el polimorfismo descrito en (e). Dicho ejemplo deberá incluir la definición de un tipo de dato, una función que la utilice, y una función principal desde la que se utilice el tipo de dato de forma polimórfica.

(g) **Justifica** dónde se puede apreciar el polimorfismo en el ejemplo de (f).

Ejercicio 3 [3 puntos]

Un **árbol familiar** almacena todos los miembros de una familia que descienden de forma directa de una persona determinada, y de paso podemos almacenar su año de nacimiento:

Familia Skywalker:

```
Shmi - 72 BBY
\-----    Anakin - 42 BBY
     \-----    Luke - 19 BBY
          \-----    Ben - 26 ABY
     \-----    Leia  -  19 BBY
          \-----    Jaina - 9 ABY
          \-----    Jacen - 9 ABY
          \-----    Ben - 1 ABY
          \-----    Anakin - 10 ABY
```

Realiza en Haskell las siguientes tareas:

- Define un **tipo de dato** para representar el **año de nacimiento** de un personaje del árbol, teniendo en cuenta que el año no se representa sólo mediante la cifra, sino además puede situarse “antes de la Batalla de Yavin” (BBY, *before Battle of Yavin*) o “despues” (ABY, *after Battle of Yavin*), que es el punto de referencia de la cronología. Dicho tipo de dato debe ser comparable de forma fácil para detectar que fecha es anterior o posterior a otra, por lo que deberas definir las funciones necesarias para ello.
- Define un **tipo de dato** para almacenar el **árbol familiar**.
- **Implementa una función** que dado un árbol familiar como el que acabas de definir devuelva el nombre del **miembro más joven** de la familia.
- **Implementa una función** que dado el nombre de un personaje devuelva el nombre de su **antecesor en el árbol**. Si el nombre está repetido, la función debe devolver todas las posibilidades:

```
parent skywalker “Anakin”  =  [“Shmi” , “Leia” ]
```

Ejercicio 4 [2 puntos]

Implementa en un lenguaje orientado a objetos (C++ o Java) un tipo de datos que permita representar conjuntos. Un conjunto es una colección de elementos en la que los elementos no pueden estar repetidos.

Define los siguientes elementos:

- Una clase/interfaz que permita almacenar conjuntos **de cualquier tipo de datos** que cumpla las restricciones necesarias para implementar el resto de funcionalidad que se te pida.
- Un **método** para **añadir un elemento** al conjunto, verificando que no está ya incluido.
- Un **método** que compruebe que **un conjunto dado es correcto**, es decir, que no existen elementos repetidos.
- Un **método** que compruebe si un elemento dado **pertenece** al conjunto.

Puedes usar todos los elementos del lenguaje necesarios, excepto un tipo de datos conjunto en el caso de que ya exista, obviamente.