Examen de **Tecnología de Programación** **17 de Junio 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, 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. 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

```
template<typename T>
class Foo {
  T t;
  int n;
public:
  Foo(const T& _t) :
       t(_t),n(1)
              { }

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

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

bar.h

```
#include <iostream>
class Bar {
 int n, d;
public:
 Bar(int _n, int _d) :
      n(_n), d(_d) { }
 Bar operator+(
        const Bar& bas) const
 { return Bar(
  n*bas.d + bas.n*d,   d*bas.d);
 }
 Bar operator/(int i) const
 {  return Bar(n, d*i); }

friend std::ostream&
 operator<<(std::ostream& os,
                const Bar& tolo)
{ os<<tolo.n<<"/"<<tolo.d;
                    return os; }
};
```

main.cc

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

int main(int argc, char** argv) {
 Foo<int>   ifoo(1);
 ifoo.mar(2); ifoo.mar(3);
 std::cout<<ifoo.sil()<<std::endl
;

 Foo<double> ffoo(0.1);
 ffoo.mar(0.2); ffoo.mar(0.3);
 std::cout<<ffoo.sil()<<std::endl
;

 Foo<Bar>   bfoo(Bar(1,10));
 bfoo.mar(Bar(2,10));
 bfoo.mar(Bar(3,10));
 std::cout<<bfoo.sil()<<std::endl
;
}
```

Programa B

foo.h

```
class Foo {
public:
  Foo() { }
  virtual int
            value() const
  { return 1; }

};
```

bar.h

```
#include "foo.h"
class Bar : public Foo {
  Foo* foo;
public:
  Bar(Foo* f) : foo(f) { }

  int value() const override
  { return foo->value()+1; }
};
```

main.cc

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

int main(int argc, char** argv)
{
  Foo* foos[5];
  foos[0]=new Foo();
  for (int i=1;i<5;++i)
     foos[i]=new Bar(foos[i-1]);

  for (Foo* f : foos)
     std::cout<<f->value()<<" ";
  std::cout<<std::endl;

  for (Foo* f : foos)
     delete f;
}
```

Ejercicio 2 [3 puntos]

La **herencia** es fundamental para el paradigma orientado a objetos. Sobre este tema realiza las siguientes tareas:

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

(b) **Elije** un lenguaje orientado a objetos que disponga de herencia.

(c) **Cita tres consecuencias** de la herencia en el lenguaje elegido sobre las clases afectadas por la misma.

(d) **Ilustra** con un ejemplo de código fuente en el lenguaje elegido tanto tu definición de herencia (a) como las tres consecuencias citadas (c).

(e) **Explica** sobre tu ejemplo (d) tanto tu definición de la herencia (a) como las tres consecuencias de la herencia que has elegido (c).

Ejercicio 3 [2 puntos]

La ordenación por mezcla o **mergesort** para listas se basa en:

- dividir la lista a ordenar en dos partes
- ordenar cada una de ellas
- mezclarlas de nuevo tomando siempre el elemento de la lista correspondiente que vaya primero

Implementa las siguientes funciones en Haskell:

split

Recibe una lista y devuelve dos mitades. Si la lista tiene un numero de datos impar una de ellas tiene un elemento mas.

merge

Recibe dos listas, que se suponen ordenadas, y las mezcla generando a su vez una lista ordenada

msort

Recibe una lista y utiliza las funciones anteriores para devolver la lista ordenada

NOTA: no está permitido utilizar NINGUNA de las funciones predefinidas de Haskell, como length, splitAt, take, drop, etc.

Por simplicidad, supondremos que el criterio de ordenación es el operador **'<',** aunque todo el código debe ser genérico.

Se valorara la especificación correcta de los prototipos de las funciones.

Ejercicio 4 [3 puntos]

Los programas de animación en 2D-1/2 (dos dimensiones y media) se usan para generar peliculas de dibujos animados. Se llaman asi porque son un paso intermedio entre dos y tres dimensiones. Permiten manejar objetos planos, fijos y moviles, situados a distintas distancias de la cámara (profundidad), y que se pueden ocultar unos a otros. Para dibujar correctamente los objetos hay que hacerlo en orden de atrás a adelante.

Diseña en el lenguaje orientado a objetos que desees (C++ o Java) una jerarquia de clases/interfaces que permita a un programa de animacion 2D-1/2 representar una escena, teniendo en cuenta los siguientes puntos:

- Todos los objetos de la escena tienen una posicion (representada por dos coordenadas reales x e y) y una profundidad o distancia a la cámara (z). La profundidad se usara despues para ordenarlos de mas lejano a más cercano y dibujarlos en ese orden. Todos los objetos deben poder ser dibujados independientemente de su tipo.
- Pueden existir objetos fijos (fondos, muebles) o móviles (personajes). La posicion de los objetos móviles puede cambiar a lo largo del tiempo, por lo que antes de dibujarlos el objeto debe actualizarla en funcion del tiempo (el propio objeto calcula su nueva posicion) a traves de un metodo **`update(float t)`**.
- Los objetos pueden estar empaquetados en grupos,. Un grupo se comporta como un objeto normal, pero contiene otros objetos. Un grupo determinado solo puede contener objetos de un tipo, o fijos o moviles. Un grupo de objetos fijos es un objeto fijo, y un grupo de objetos moviles es un objeto móvil, de cara a su funcionalidad. Los grupos deben tener un método **`add(….)`** que permita añadir sólo objetos del tipo correcto. Por supuesto, un grupo puede contener otro grupo, ya que son un objeto fijo o movil.
- La escena guarda el conjunto de objetos necesarios para dibujar la animación.

Define las clases y los métodos necesarios para implementar esa funcionalidad.

Para la clase Escena, implementa la siguiente funcionalidad:

- Un metodo **`sort()`** que ordene los objetos de mayor a menor profundidad. No es necesario que implementes la ordenacion completa, pero si que definas el interfaz de la funcion y donde están los datos sin ordenar y ordenados.
- Un metodo **`draw()`** que dibuje la escena.
- Un metodo **`update(float t)`** que actualiza la posicion de todos los objetos para el instante de tiempo t.