Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

14 de septiembre de 2018

Examen de **Tecnología de Programación**

Grado en Ingeniería Informática

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

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

- Prueba escrita → 60% de la nota (mínimo 5 para aprobar)
- Prácticas (máxima nota entre prácticas entregadas 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 cuyo nombre aparece encima de su código. 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` (gcc versión $>$ 4.8.1). ¿Compilaría correctamente o daría algún error? Ignora errores tipográficos y potenciales *memory leaks*.
- Si tu respuesta es que no compila ¿dónde ocurre el error de compilación? ¿por qué no compila?
- Si tu primera respuesta es que sí que compila ¿qué sacaría el ejecutable por pantalla? No lo justifiques.

Programa A

foo.h

```
class Foo {
protected:
 double f;

public:
 Foo(double f_):
   f(f_) {}

 void sil()
 {  f/=2.0;    }

 virtual double meta() const
 {  return f;  }
};
```

bar.h

```
#include "foo.h"
class Bar: public Foo
{
public:
 Bar(double t) :
   Foo(t+1.0) { }

 void sil()
 {  f*=2.0;    }

 double meta() const
         override
 {  return f-1.0; }
};
```

main.cc

```
#include <iostream>
#include "bar.h"
using namespace std;

void mira(Foo* foo) {
   foo->sil();
   cout<<foo->meta();
   cout<<endl;
}

int main() {
  mira(new Foo(2.0));
  mira(new Bar(4.0));
}
```

Programa B

foo.h

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

bar.h

```
template <typename T>
class Bar {
 T t;
public:
 Bar(const T& t_) :
    t(t_) {}

 int mitzvah() const {
   return t.mitzvah()+1;
 }
};

template<typename T>
Bar<T> bar(const T& t) {
   return Bar<T>(t);
}
```

main.cc

```
#include "bar.h"
#include "foo.h"
#include <iostream>
using namespace std;

template<typename T>
void show(const T& t) {
  cout<<t.mitzvah()<<endl;
}

int main() {
  show(Foo());
  show(bar(Foo()));
  show(bar(bar(Foo())));
  show(bar(bar(bar(Foo()))));
}
```

Ejercicio 2 [3 puntos]

La **programación genérica** es un mecanismo de diferentes lenguajes que permite nuevas formas de polimorfismo. En concreto, nos permite definir **tipos genéricos de datos**. Sobre este tema, realiza las siguientes tareas:

- **(a)** **Define**, con tus propias palabras y de forma independiente a un lenguaje de programación específico, el concepto de tipo genérico de datos.
- **(b)** La utilización práctica de un tipo genérico de datos consta de dos fases: su **definición** y su **instanciación**. Explica en dos frases en qué consiste cada una de las dos fases.
- **(c)** **Ilustra** mediante un **ejemplo** de código fuente en un lenguaje de programación a tu elección un ejemplo las dos fases involucradas en la utilización de un tipo genérico de datos. Dicho ejemplo deberá ser completo y compilar.
- **(d)** **Justifica** en tu ejemplo dónde se aprecia tu definición de **(a)** y tu explicación de las dos fases de **(b)**.
- **(e)** Escribe otro ejemplo de código fuente en el que la aplicación de las reglas del lenguaje para los tipos genéricos de datos esté forzando un **error de compilación**. Dicho error de compilación deberá depender exclusivamente de la programación genérica, obviando errores léxicos o sintácticos (puntos y comas, identificadores mal escritos) o errores que se correspondan con otras partes del lenguaje de programación elegido.
- **(f)** **Explica** dado tu código de **(e)**, dónde se produce el error de compilación, por qué se produce y cómo se relaciona dicho error con tus respuestas anteriores en **(a)** y **(b)**.

Ejercicio 3 [2 puntos]

Un **sensor de temperatura** guarda los valores proporcionados en diferentes intervalos de tiempo en listas de Haskell. Para hacer un estudio sobre el cambio climático, se quiere saber si la representación de estos valores tiene forma de montículo o no. Una secuencia de números tiene forma de montículo si los números forman una secuencia estrictamente creciente hasta que alcanzan un máximo (que es único) y, a partir de entonces, forman una secuencia estrictamente decreciente. Para que se considere montículo es necesario que, sin contar el valor máximo, tanto la subsecuencia estrictamente creciente como la subsecuencia estrictamente decreciente sean no vacías.

Se pide desarrollar la función en Haskell `esMonticulo` que diga si una lista de números reales tiene estructura de montículo o no. Se podrá suponer que la lista siempre contiene al menos un dato, aunque el número máximo de datos que puede almacenar no está acotado. A continuación se muestran posibles ejemplos de ejecución:

```
*Main> esMonticulo [1,2,3,2,1]
True
*Main> esMonticulo [1]
False
*Main> esMonticulo [1,3]
False
*Main> esMonticulo [3,2,1]
False
*Main> esMonticulo [1,2,2,1]
False
*Main> esMonticulo [1,2,3,4,1]
True
```

Se valorará positivamente la **brevedad y concisión** de las soluciones propuestas. Se puede utilizar cualquier función de Haskell, y también se pueden definir otras funciones propias si se considera necesario.

Ejercicio 4 [3 puntos]

Para una serie de cálculos científicos, se necesita la posibilidad de almacenar **expresiones matemáticas**, incluyendo constantes numéricas y los operadores de suma, resta, multiplicación y división, con las reglas de precedencia habituales:

- Dentro de el mismo nivel de precedecia (por ejemplo varias sumas concatenadas), el orden normal de evaluación de las operaciones es de **izquerda a derecha**.
- La **precedencia de las operaciones** la siguiente: primero se evalúan las multiplicaciones, después las divisiones y después las sumas y las restas.
- Recursivamente se pueden **utilizar paréntesis**, que hacen que la sub-expresión a la que rodean tenga el máximo grado de precedencia (se evalúan antes que todo lo demás).

Por ejemplo:

$$3 + 2 \times 5 = 13$$
$$(3 + 2) \times 5 = 25$$

En un lenguaje orientado a objetos a tu elección (C++ o Java) lleva a cabo las siguientes tareas:

- Diseña una **estructura de clases** que permita almacenar en memoria una expresión matemática, incluyendo constantes numéricas y los cuatro operadores de suma, resta, multiplicación y división. En tiempo de compilación se debe poder elegir el **tipo de dato numérico** que se va a considerar para realizar las operaciones (entero, entero largo, punto flotante de precisión simple o doble...) y permitiendo representar expresiones entre paréntesis.
- Añade, en la clase correspondiente, un método `mostrar()` que permita **mostrar por pantalla** una expresión matemática completa, con los símbolos de todas las operaciones involucradas y los paréntesis que sean necesarios (según la precedencia).
- Añade, en la clase correspondiente, un método `evaluar()` que devuelva un número resultado de **evaluar la expresión matemática**, con el tipo de dato numérico correspondiente. Los métodos `evaluar()` y `mostrar()` deben de ser coherentes entre sí: el resultado comprobar a mano el resultado de una expresión mostrada debe coincidir con el resultado del método de evaluación.

Nota : No se permite que la estructura de datos sea ningún tipo de cadena de texto.