Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

14 de junio de 2019

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 → 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. Para cada uno de los programas, responde a las siguientes preguntas:

- Se compila con el comando `g++ -std=c++11 main.cc` (gcc versión ≥ 5). ¿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:
  int i;
public:
 Foo(int _i) : i(_i) {}

 virtual int yokey() const
 {
  return i+1;
 }
 int sil() const
 {
  return 2*this->yokey();
 }
};
```

bar.h

```
#include "foo.h"
class Bar : public Foo
{
public:
 Bar(int _i) :
             Foo(_i+3) { }
 int sil() const
 {
   return i-2;
 }
 int yokey() const override
 {
   return -1*this->sil();
 }
};
```

main.cc

```
#include <iostream>
#include "bar.h"
using namespace std;
int main() {
 Foo* foo = new Foo(1);
 cout<<foo->sil()<<" | "
     <<foo->yokey()<<endl;
 Bar* bar = new Bar(2);
 cout<<bar->sil()<<" | "
     <<bar->yokey()<<endl;
 Foo* foobar = new Bar(3);
 cout<<foobar->sil()<<" | "
     <<foobar->yokey()<<endl;

}
```

Programa B

foo.h

```
#include <string>
class Foo {
public:
 std::string meta() const
 {
  return std::string("FOO");
 }
};
```

bar.h

```
#include "foo.h"
class Bar : public Foo {
public:
 std::string meta() const
 {
  return std::string("BAR");
 }
};
```

main.cc

```
#include "bar.h"
#include <iostream>
using namespace std;
void inher(Foo* foo) {
 cout<<foo->meta()<<endl;
}
template<typename F>
void templ(F* f) {
 cout<<f->meta()<<endl;
}
int main() {
 Foo foo;
 Bar bar;
 inher(&foo);
 inher(&bar);
 templ(&foo);
 templ(&bar);
}
```

Ejercicio 2 [3 puntos]

La **programación genérica** es un mecanismo de diferentes lenguajes de programación que establece una forma de conseguir polimorfismo.

- **(a)** **Define**, con tus propias palabras y de forma independiente a un lenguaje de programación específico, el concepto de programación genérica.
- **(b)** Explica cómo se consigue **polimorfismo** mediante programación genérica.
- **(c)** **Ilustra** la programación genérica mediante un **ejemplo** en un lenguaje de programación a tu elección la programación genérica. Añade un programa principal en el que se pueda apreciar el polimorfismo emergente de la programación genérica.
- **(d)** **Justifica** en tu ejemplo de **(c)** tanto tu definición de **(a)** como tu explicación de **(b)**.
- **(e)** Replica tu **ejemplo** de **(c)** en el mismo lenguaje de programación pero **sin utilizar programación genérica**.
- **(f)** Explica qué dificultades y limitaciones has encontrado para diseñar tu código para **(e)** frente a tu ejemplo de **(c)**.

En tu solución, señala claramente y sin ambigüedades qué apartado estás resolviendo en cada caso con la letra correspondiente, de la **(a)** a la **(f)**.

Ejercicio 3 [2 puntos]

Los **L-Systems** o **Sistemas de Lindenmayer** son sistemas de reglas de re–escritura que se utilizan para dibujar elementos naturales, como plantas o copos de nieve. Un L-System se define mediante un alfabeto de símbolos, a los que se les atribuye una representacion gráfica:

F, **G**, ... : moverse hacia adelante una distancia unidad dibujando una linea
+ : girar un ángulo fijo en sentido antihorario
– : girar un ángulo fijo en sentido horario

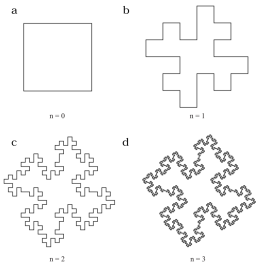
Una regla de re–escritura especifica como se sustituye cada símbolo, por ejemplo:

$$\mathbf{F} \rightarrow \mathbf{F+F+F+F}$$

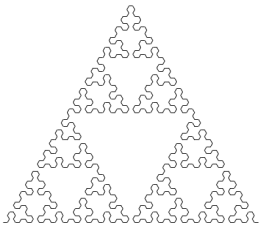
Los simbolos para los que no exista regla se quedan como están.

A partir de una cadena inicial y una serie de reglas podemos obtener la definición de nuestro gráfco repitiendo iterativamente la aplicación de las reglas. En cada paso se aplican todas las reglas en el orden en que estén definidos los símbolos:

Alfabeto: F + –
Inicio: F+F+F+F
Reglas: F → F–F+F+FF–F–F+F
Angulo: 90 grados



Alfabeto: F G + –
Inicio: F
Reglas: F → F+G+F
G → G–F–G
Angulo: 60 grados



Se pide, utilizando el lenguaje Haskell, dada la función:

```
rewrite :: Char -> String
```

que define las reglas de re–escritura para cada símbolo de nuestro alfabeto, implementar la función **`lsystem`**, que a partir del conjunto de símbolos para los que aplicaremos las reglas de sustitucion (que además define el orden en que las aplicamos), esa función que define las reglas, la cadena inicial y el número de veces que tenemos que aplicarlas, devuelve la cadena resultante:

```
lsystem :: String -> (Char -> String) -> String -> Int -> String
```

Por ejemplo:

```
rewrite 'F' = "F+G+F"
rewrite 'G' = "G-F-G"
rewrite  c  = [c]
> lsystem "FG+-" rewrite "F" 0
"F"
> lsystem "FG+-" rewrite "F" 1
"F+G-F-G+F"
> lsystem "FG" rewrite "F" 2
"F+G-F-G+F+G-F-G-F+G-F-G+F-G-F-G+F+G-F-G+F"
```

Ejercicio 4 [3 puntos]

La empresa fabricante de microprocesadores **Dintel** te ha encargado un emulador para el juego de instrucciones de su próximo microprocesador. En un lenguaje orientado a objetos a tu elección (C++ o Java) debes representar el juego de instrucciones y los elementos del microprocesador. Es una máquina de 64 bits (registros, entrada y salida aceptan 64 bits simultáneamente), que puedes representar con el tipo de datos que consideres (`uint64_t` en C++, `unsigned long` en Java).

Tu emulador deberá representar los siguientes **componentes** de la máquina:

- Una **entrada** (el teclado) desde el que sólo se pueden leer elementos.
- Una **salida** (la pantalla) al que sólo se pueden escribir elementos.
- Un **registro**, en el que se pueden leer y escribir elementos (es de entrada y salida)
- La **máquina** completa, que contenga un número indeterminado de registros, un teclado como entrada y una pantalla como salida.

Con respecto al **juego de instrucciones**, deberás poder representar lo siguiente:

- `mov %a %b` - mueve un dato desde la entrada `%a` hasta la salida `%b`.
- `inc %a` - incrementa en uno el valor de `%a` (que debe ser de entrada y de salida).
- Un **programa** completo, que consiste en un número indeterminado de instrucciones que se ejecutan secuencialmente.

Tu código deberá no compilar cuando se guarde un valor en un elemento que no sea de salida (`mov` que guarde el resultado en el teclado como parámetro `%b`, por ejemplo) o si se intenta leer un valor de un elemento que no sea de entrada (`mov` que lea un valor de la pantalla como parámetro `%a`, por ejemplo). Este error de compilación deberá ser generado por el sistema de tipos del lenguaje y no por comandos de aserción estática. Por último, tu emulador deberá de ser capaz de **ejecutar** un programa completo.

Se valorará positivamente la concisión y principalmente la **escalabilidad** (posibilidad de añadir nuevos elementos a la máquina tanto de entrada como de salida como ambas y nuevas instrucciones al juego de instrucciones) de la solución propuesta.