Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza
1542

Examen de Teoría

7 de septiembre de 2022

Tecnología de Programación
Grado en Ingeniería Informática
Duración: 2 horas 30 minutos

RECUERDA: La evaluación final consta de las siguientes pruebas:

Prueba escrita	→	60 % de la nota (mínimo 5 para aprobar)
Prácticas y Problemas	→	40 % de la nota

Ejercicio 1 **[2 puntos]**

A continuación tienes dos programas, escritos en C++ y en Haskell, que realizan la misma función:

C++

```
int add(int x,int y)
{ return x+y; }

int inc(int n)
{ return add(1,n); }

int main() {
  cout << inc(3) << endl;

  return 0;
}
```

Haskell

```
add :: Int -> Int -> Int
add x y = x+y

inc = add 1


main = do
  print $ inc 3
```

La funcionalidad de los dos programas es la misma, pero están implementados de forma distinta:

1. ¿ En qué se diferecian las dos implementaciones ?
2. ¿ Qué concepto/propiedad de los dos lenguajes (que uno usa y el otro no) es responsable de esa diferencia ?
   Define ese concepto.
3. ¿ Se puede usar el mismo tipo de implementación en el otro lenguaje ?

Ejercicio 2 **[2 puntos]**

Elije **dos lenguajes** entre los vistos en la asignatura (C++, Java y Haskell). Para cada uno de los dos lenguajes elegidos, realiza las siguientes tareas:

1. **Implementa** (en ese lenguaje) una **única función** (o un método estático si el lenguaje no permite funciones) que, dada una lista, devuelva como resultado la suma de todos los elementos de la lista. Dicha (única) función deberá permitir listas de diferentes tipos de datos, y la suma será del mismo tipo de datos que los elementos de la lista. Si el lenguaje ya incluye en su biblioteca estándar una función que tenga exactamente esta funcionalidad, no puedes utilizarla directamente, sino que tendrás que hacer tu propia implementación.

2. **Explica brevemente** cuál es el **mecanismo del lenguaje** que has utilizado para que la función implementada sirva para listas de diferentes tipos de datos, devolviendo el tipo de datos adecuado, e identifica dónde se ve en el código que has desarrollado.

3. **Responde:** ¿Qué ocurriría si la función implementada fuera invocada por una lista de un tipo de dato para el que no existe implementación del operador de suma? Explica **en qué momento y dónde detectaría el compilador** que no existe dicho operador de suma.

Ejercicio 3 **[3 puntos]**

Para calcular los valores de las funciones trigonométricas se utilizan series de potencias del tipo:

$$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \dots$$
$$cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} \dots$$

El coste de calcular esas series se debe a la evaluación de las potencias y de los factoriales. Además, deben repetirse en las dos series, aunque no se usen directamente; por ejemplo, para calcular el valor x^5 necesario para el seno, es necesario calcular antes x^4, que no se usa directamente en la serie. Pero sí sería útil para calcular el valor del coseno. Y en muchas ocasiones se necesitan los dos valores, tanto el seno como el coseno.

Para optimizar el cálculo de esas funciones se pide implementar en Haskell una función

```
sincos n x
```

que implemente el cáculo de la siguiente forma:

- Devuelve simultáneamente los dos valores de $sin(x)$ y $cos(x)$.
- Se calculan mediante las series que se definen arriba, utilizando el número de términos `'n'` que indica el primer parámetro de la función.
- Los térmimos de la serie se calculan de forma incremental (cada uno se calcula a partir del valor del anterior, no desde cero), minimizando el número de operaciones.

Se valorará la concisión, brevedad y eficiencia de la solución propuesta.
Se permite definir nuevas funciones Haskell, pero recuerda que se tendrá en cuenta la eficiencia de la solución, que podría verse afectada por las funciones que reutilicen entre apartados.

Ejercicio 4 **[3 puntos]**

Para el desarrollo de un nuevo **microcontrolador de tipo pila**, se te pide que implementes una máquina virtual que emule dicho controlador y que permita experimentar con el juego de instrucciones y hacer análisis de eficiencia sobre el mismo. Un microcontrolador de pila consta como memoria a corto plazo de una única pila, y todas las instrucciones del juego de instrucciones están diseñadas para coger sus operandos de la pila (desapilándolos, *pop*) y para devolver su resultado en la propia pila (apilándolos, *push*). El microcontrolador trabaja en punto flotante de 32 bits (tipo `float`).

En un lenguaje orientado a objetos a tu elección (C++ o Java) representa:

- La **pila** del microcontrolador. No es necesario implementar una pila de cero, sino que se puede usar otra estructura de datos proporcionada por el lenguaje (una lista, por ejemplo) para representarla.
- Las **instrucciones** del juego de instrucciones. Cada una de dichas instrucciones deberá proporcionar información de cuánto tiempo tarda (ciclos de reloj, un entero), y deberá poder ejecutarse. En su ejecución, cada tipo de instrucción tendrá un comportamiento diferente. Deberás representar las siguientes instrucciones:
  - `push` <c> - apila la constante <c>(parámetro de la instrucción) en la pila. Requiere 1 ciclo de reloj.
  - `add` - desapila dos valores de la pila y apila su suma. Requiere 4 ciclos de reloj.
  - `mul` - desapila dos valores de la pila y apila su producto. Requiere 14 ciclos de reloj.
- Un **programa**, una lista de instrucciones que se ejecutan secuencialmente desde la primera hasta la última, con el siguiente comportamiento:
  - **Calcular** el tiempo total (en ciclos de reloj, un entero) del programa.
  - **Ejecutar** el programa completo, que, a partir de una pila de entrada, la transforme pasando por todas las instrucciones.

Por último, implementa un programa principal que cree un programa que contenga, como mínimo, una instrucción de cada tipo, y que además muestre por pantalla el tiempo total en ciclos de ejecución y ejecute el programa completo a partir de una pila vacía.

Además del comportamiento que te pedimos explícitamente, puedes definir todos los métodos y clases que consideres necesarios. Puedes usar (y se recomienda) usar tipos de datos de las bibliotecas estándar del lenguaje elegido (`std::list<...>` en C++ y `LinkedList<...>` en Java).

Se valorará especialmente la escalabilidad del sistema, como por ejemplo la posibilidad de añadir instrucciones al juego de instrucciones sin tener que modificar fragmentos de código común a todas ellas. También se valorará que no exista código repetido o métodos innecesarios.