Práctica 2: programación funcional en Scala; recursividad

Nuevas tecnologías de la programación

Contenido:

1 Objetivos

En esta primera práctica se trata de trabajar con el lenguaje de programación Scala, definiendo algunas funciones recursivas cuya declaración ya se proporciona. La implementación realizada debe ser capaz de superar un conjunto de pruebas proporcionado con el material de la práctica, usando la librería **scalatest**.

2 Definición de los ejercicios

2.1 Triángulo de Pascal

El siguiente patrón de números se conoce como triángulo de Pascal:

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
        1   6   15  20  15  6   1
      1   7   21  35  35  21  7   1
    1   8   28  56  70  56  28  8   1
  1   9   36  84  126 126 84  36  9   1
1   10  45  120 210 252 210 120 45  10  1
```

Los números en el vértice del triángulo son todos 1 y cada número interior puede obtenerse como la suma de los valores que tiene sobre él. Se trata de escribir una función que calcule los elementos del triángulo de forma recursiva. Para ello se escribirá una función con la siguiente declaración:

```
def calcularValorTrianguloPascal(columna: Int, fila: Int): Int
```

Esta función recibe como argumento una columna y una fila (comenzando por el valor 0) y devuelve el valor almacenado en la posición correspondiente del triángulo. Esto permitiría escribir un método **main** de la forma siguiente:

```
/**
  * Metodo main: en realidad no es necesario porque el desarrollo
  * deberia guiarse por los tests de prueba
  *
  * @param args
  */
def main(args: Array[String]) {
  println("..................... Triangulo de Pascal ...................")

  // Se muestran 10 filas del trinagulo de Pascal
  for (row <- 0 to 10) {
    // Se muestran 10 y 10 filas
    for (col <- 0 to row)
      print(calcularValorTrianguloPascal(col, row) + " ")

    // Salto de linea final para mejorar la presentacion
    println()
  }

  // Se muestra el valor que debe ocupar la columna 5 en la fila 10
  print(calcularValorTrianguloPascal(10, 15))
}
```

que permite obtener la siguiente salida:

```
..................... Triangulo de Pascal ...................
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
3003
```

Aunque se use el **main**, como se ha comentado antes, la forma básica de probar consistirá en usar el conjunto de casos de prueba específico para esta función.

2.2 Balanceo de cadenas con paréntesis

Se trata aquí de escribir una función recursiva que verifique el balanceo de los paréntesis presentes en una cadena de caracteres, representada como **List[Char]** (no como objeto de la clase **String**). Algunas cadenas balancedas son:

- (if (a ¿ b) (b/a) else (a/(b*b)))
- (ccc(ccc)cc((ccc(c))))

Algunas no balanceadas:

- (if (a ¿ b) b/a) else (a/(b*b)))
- (ccc(ccccc((ccc(c))))
- ())()())
- ())(

El último ejemplo pone de manifiesto que no es suficiente verificar que la expresión contiene el mismo número de paréntesis abriendo y cerrando, ya que deben seguir el orden adecuado. La función tendrá la siguiente declaración:

```
1 def chequearBalance(cadena: List[Char]): Boolean
```

Hay tres métodos de la clase **List** que son útiles para realizar este ejercicio:

- **cadena.isEmpty**: comprueba si la lista está vacía
- **cadena.head**: obtiene el primer elemento de la lista
- **cadena.tail**: devuelve una nueva lista sin el primer elemento

Pueden definirse funciones auxiliares, si resulta conveniente.

2.3 Contador de posibles cambios de moneda

Se trata aquí de escribir una función recursiva que determine de cuántas formas posibles puede devolverse una cierta cantidad. Por ejemplo, con monedas de valor 1 y 2 hay 3 formas de cambiar el valor 4:

- $1+1+1+1$
- $1+1+2$
- $2+2$

La función tiene la siguiente declaración:

```
1 def contarCambiosPosibles(cantidad: Int, monedas: List[Int]): Int
```

Aquí pueden usarse también los métodos de utilidad vistos en el ejercicio anterior y relativos a la clase **List**. En este ejercicio ayuda pensar en los casos extremos:

- ¿cuántas formas hay de dar cambio de un valor 0?
- ¿cuántas formas hay de dar cambio de un valor positivo si no tenemos monedas?

3 Implementación

Todo el código puede desarrollarse usando como punto de partida el esqueleto disponible en el material de la práctica, con nombre **Main.Scala** y rellenar el cuerpo de las funciones con las sentencias necesarias. Recordad que es posible incluir funciones dentro de funciones (para el caso en que convenga usar alguna función auxiliar para resolver el problema).

```scala
/**
  * Objeto singleton para probar la funcionalidad del triangulo
  * de Pascal
  */
object Main {

  /**
    * Metodo main: en realidad no es necesario porque el desarrollo
    * deberia guiarse por los tests de prueba
    *
    * @param args
    */
  def main(args: Array[String]) {
    println("..................... Triangulo de Pascal ...................")

    // Se muestran 10 filas del trinagulo de Pascal
    for (row <- 0 to 10) {
      // Se muestran 10 10 columnas
      for (col <- 0 to row)
        print(calcularValorTrianguloPascal(col, row) + " ")

      // Salto de linea final para mejorar la presentacion
      println()
    }

    // Se muestra el valor que debe ocupar la columna 5 en la fila 10
    println(calcularValorTrianguloPascal(10, 15))
    println(calcularValorTrianguloPascal(0, 0))
  }

  /**
    * Ejercicio 1: funcion para generar el triangulo de Pascal
    *
    * @param columna
    * @param fila
    * @return
    */
  def calcularValorTrianguloPascal(columna: Int, fila: Int): Int = {
     // A rellenar
  }

  /**
    * Ejercicio 2: funcion para chequear el balance de parentesis
    *
    * @param cadena cadena a analizar
    * @return valor booleano con el resultado de la operacion
    */
  def chequearBalance(cadena: List[Char]): Boolean = {
     // A rellenar
  }
```

```
51
52    /**
53      * Ejercicio 3: funcion para determinar las posibles formas de devolver el
54      * cambio de una determinada cantidad con un conjunto de monedas
55      *
56      * @param cantidad
57      * @param monedas
58      * @return contador de numero de vueltas posibles
59      */
60    def contarCambiosPosibles(cantidad: Int, monedas: List[Int]): Int = {
61       // A rellenar
62    }
63 }
```

4 Material a entregar

Al final de la realización de la práctica se entregará un archivo comprimido con el contenido completo de la práctica, tal y como se integra en el proyecto con el entorno de desarrollo que hayáis usado. Se incluirá también un pequeño documento indicando el entorno de desarrollo y una breve valoración de la práctica (si los conceptos vistos son novedosos, si os ha parecido de interés, problemas encontrados, etc) en tres o cuatro líneas.

La fecha de entrega se fijará más adelante. La entrega se hará mediante la plataforma **PRADO**.