
# INFORME DEL TALLER 1

**MAYRA ALEJANDRA SANCHEZ SALINAS (2040506)**

## FUNDAMENTOS DE PROGRAMACIÓN FUNCIONAL Y CONCURRENTE

**JUAN FRANCISCO DIAZ FRIAS**

**UNIVERSIDAD DEL VALLE**
**SANTIAGO DE CALI, VALLE DEL CAUCA**
**2022**

## CONTENIDO

1. **Informe de procesos**
   - **Triángulo de pascal**

Los números en el borde del triángulo son todos 1, y cada número dentro del triángulo es la suma de los dos números que están encima. Debe escribir una función que calcule los elementos del triángulo de Pascal mediante un proceso recursivo.

Haga este ejercicio implementando la función pascal

C son las filas y R las columnas, contando desde 0 y devuelve el número en ese punto en el triángulo. Por ejemplo, pascal(0,2)=1, pascal(1,2)=2 y pascal(1,3)=3.

Esto fue el procedimiento en el package.scala:

```scala
/**
 * Ejercicio 1 (Triangulo de pascal)
 */
def pascal (c: Int, r: Int): Int={
  /* c = columnas y r= filas
  c & r > 0 y la longitud de c < r*/
  if (c < 0 || r < 0 || c > r)throw new IllegalArgumentException("No es posible, debe ser c<r")else{
    if (c == 0 || c == r) 1 else {
      pascal(c - 1, r - 1) + pascal(c, r - 1)
    }
  }
}
```

Considero que este es una función recursiva ya que se llama la misma función así misma en pascal (c-1,r-1) + pascal (c,r-1)

- **Balanceo de paréntesis**

Escriba una función recursiva que verifique el equilibrio de paréntesis en una cadena de Strings, que representamos como List[Char] (lista de caracteres), no como un String. Balanceo de paréntesis se refiere a cada paréntesis de apertura le corresponde un paréntesis de cierre en su debido orden. Por ejemplo, la función deberá devolver verdadero para las siguientes cadenas:

- (if (esCero x) max (/ 1 x))
- Yo se lo dije (que no está (aún) listo). (Pero él no estaba escuchando)

En cambio, la función debería devolver falso para las siguientes cadenas de caracteres:

- :-)
- ())(

El último ejemplo nos demuestra que no es suficiente simplemente verificar que una cadena contiene el mismo número de paréntesis de apertura y cierre. Desarrolle este ejercicio implementando la función balanceado:

```
/**
 * Ejercicio 2 (Balanceo de parentesis)
 */
def balanceoParentesis (chars: List[Char]): Boolean ={
  def parentesis (lista: List[Char], cont: Int): Int={
    if (lista.isEmpty && cont >= 0) cont else{
      if (cont<0)cont else{
        if (lista.head == ')') parentesis(lista.tail , cont - 1)else{
          if (lista.head == '(') parentesis(lista.tail , cont + 1)else{
            parentesis(lista.tail, cont)
          }
        }
      }
    }
  }
 parentesis(chars,0)==0
}
```

Esta función cumple con ser recursiva en la función paréntesis ya que se vuelve a llamar la función evaluando el balanceo de paréntesis.

- **Cambio de monedas**

  Escriba una función recursiva que cuente de cuántas maneras diferentes se puede lograr una cantidad de pesos, usando unas determinadas denominaciones de monedas. Por ejemplo, hay 3 formas de completar 400 pesos, si se tienen monedas con denominaciones de 100 y 200: (1) 4 monedas de 100 pesos, (2) 2 monedas de 100 pesos y 1 moneda de 200 pesos, y (3) 2 monedas de 200 pesos.

Realice este ejercicio implementando la función cambioMonedas. Esta función requiere una cantidad para cambiar y una lista de denominaciones únicas para las monedas.

```
/**
 * Ejercicio 3 (Cambio de monedas)
 */
def cambioMonedas(total: Int, denom: List[Int]): Int = {
  if(total == 0)
    1
  else if(total > 0 && !denom.isEmpty)
    cambioMonedas(total - denom.head, denom) + cambioMonedas(total, denom.tail)
  else
    0
}
```

## 2. Informe de corrección

- **Argumentación sobre la corrección**
  - **Pascal**

    **Argumentación:** Si se ingresa pascal (3,4) se verifica primero que: c < 0, o sea 3 < 0 lo cual es falso, después que r < 0, o sea 4 < 0 lo cual es falso también y por último verificar que c no sea mayor que r, por lo que 3 > 4 entonces es falso.

    Después pasamos a verificar si c == 0 ó c == r ( si eso se llega a cumplir el resultado es 1 ) pero en este caso no es así, por lo que se recurre a la siguiente operación:

    pascal (c-1,r-1) + pascal (c,r-1)

    la cual es una llamada a la función que se acaba de analizar

pascal (c-1,r-1) + pascal (c,r-1), se reemplazan los valores
pascal (3-1,4-1) + pascal (3,4-1), se resuelve
(*) pascal (2,3) + pascal (3,3), ahora se vuelve a analizar la función pascal

¿Cuál es el resultado pascal (2,3) y pascal (3,3)?

pascal (2,3) = 3 y pascal (3,3)=1

por lo que pascal (*) = pascal (2,3) + pascal (3,3) es 4

  - **Balanceado de paréntesis**

Si se ingresa balanceoParentesis(“mi(mama_me(mima)”.toList), al ingresar a la función `balanceoParentesis(chars: List[Char]): Boolean =` se define de una vez el tipo de dato que ingresa y el tipo de dato de salida en este caso booleano

Se crea la funcion `def parentesis (lista: List[Char], cont: Int): Int=` y se crean dos parámetros lista y cont, entonces esta función es la que ayuda verificar lo siguiente:

- ➔ `if (lista.isEmpty && cont >= 0) cont,` que ayuda a verificar si la lista está vacía por lo que el contador arroja que es 0, y si no se cumple entonces pasa al siguiente if
- ➔ `if (cont<0)cont` que verifica si el parámetro cont es mayor a 0 entonces arroje su valor y si no entonces el siguiente if
- ➔ `if (lista.head == ')') parentesis(lista.tail , cont - 1)else` viene a comprobar si en la lista se encuentra algún paréntesis de cerradura y si es asi llama a la función interna paréntesis para devolver la lista sin el primer elemento y así ir restando un uno al conteo, después pasa a verificar el siguiente
- ➔ `if (lista.head == '(') parentesis(lista.tail , cont + 1)else` en este condicional se verifica de la misma manera que el punto anterior pero con el paréntesis de apertura
- ➔ `parentesis(lista.tail, cont)` vuelve a llamar la función paréntesis pero esta vez la list sin su primer elemento y con su conteo actual
- ➔ `parentesis(chars,0)==0` finalmente esta es una llamada a la función paréntesis asignándole la entrada de chars como la lista y el 0 como cont

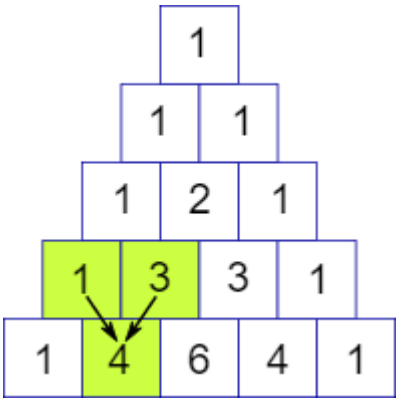
- **Cambio de monedas**

En este caso `cambioMonedas ( 300, List ( 5 , 10 , 20 , 50 , 100 , 200 , 500 ))` se ingresa un valor para el dinero (300) y los billetes que tenemos en una lista que son (5,10,20,50,100,200,500) lo que hace el algoritmo es:

1. Se debe evaluar la cantidad de dinero en este caso que es 300, si fuera 0 devolvería 1 ya que la única solución es usar cero monedas, si fuera menor que 0 no se podria ya que no es posible
2. Pasa a ver las cantidades de la lista y se evalúa de manera que cuente las formas posibles de usar toda esa lista para pagar esos 300

Ejemplo: (1) 60 billetes de 5, (2) 30 billetes de 10, (3) 15 billetes de 20 y así sucesivamente y en total son 1022 formas posibles de pagar con esa cantidades

- **Casos de prueba**
  - **Triángulo de pascal**




El triángulo de pascal como se puede observar es una pirámide rodeada de unos y la suma de los pares de la fila es el siguiente número bajo de ellos 2

Con el programa realizado en el package del paquete Recursión se puede encontrar cual es el número ubicado en el triángulo.

Teniendo en cuenta que las columnas deben ser menores que las filas.

Por ejemplo:

- pascal (1, 2) siendo 1 ( c ) columna y 2 ( r ) fila podemos ver en la imagen que corresponde al número 2

```
import Recursion.{pascal,balanceado,cambioM
// Pruebas pascal
pascal( 1 , 2 )
```

```
val res0: Int = 2
```

- pascal (3,4) siendo 3 ( c ) columna y 4 ( r ) fila podemos ver en la imagen que corresponde al número 4

```
pascal( 3 , 4 )
```

```
val res1: Int = 4
```

- pascal(0,2) siendo 0 ( c ) columna y 2 ( r ) fila podemos ver en la imagen que corresponde al número 1

```
pascal( 0 , 2 )
```

```
val res2: Int = 1
```

- pascal(1,2) siendo 1 ( c ) columna y 2 ( r ) fila podemos ver en la imagen que corresponde al número 2

```
pascal( 1 , 2 )
```

```
val res3: Int = 2
```

- pascal(1,3) siendo 1 ( c ) columna y 3 ( r ) fila podemos ver en la imagen que corresponde al número 3

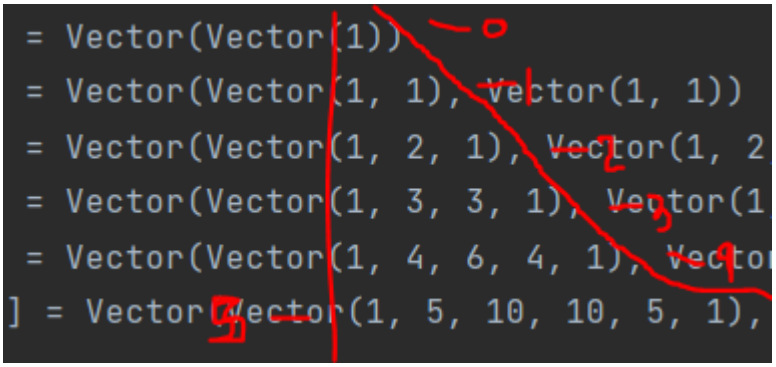
```
pascal( 1 , 3 )
```

```
val res4: Int = 3
```

Con las siguientes funciones se realizó el triángulo de pascal de forma que se pueda videncia en el worksheet

```
def filaTriangulo( fila : Int) =
  for (col<-0 to fila) yield pascal(col,fila)

def triang( filas : Int) =
  for (fila<-0 to filas) yield filaTriangulo(filas)

triang( filas = 0)
triang( filas = 1)
triang( filas = 2)
triang( filas = 3)
triang( filas = 4)
triang( filas = 5)
```

Y con esto su resultado es:




Inicia desde la fila 0

Termina en la fila 5, ya que en el worksheet, la función *triang* va hasta las fila 5

**Mi opinión**: creo que con estos casos de prueba se puede verificar que tanto la función de pascal y la función del triangulo cumple con su propósito, si se desea comprobar que pascal cumple con más parejas de números, simplemente se debe tener en cuenta que, pascal (c,r), c debe ser menor que r para todos los casos

- **Balanceo de paréntesis**

El balanceo de paréntesis consiste en que a cada par de entesis de apertura le corresponde un paréntesis de cierre en su debido orden.

Tenemos 5 casos de prueba en los cuales vamos a verificar el balanceo de paréntesis:

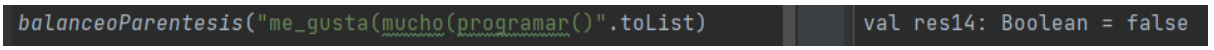
1. 
```
// Pruebas balanceo

balanceoParentesis("if_(zero?_x)_max_(/_1_x))".toList)
```
```
val res11: Boolean = false
```
En esta primera prueba hace falta 1 paréntesis de apertura para que la respuesta sea true

2. 


   En este caso vemos () pero luego esta ) y ( por lo que el resultado nos da falso ya que faltaría el paréntesis inicial y final correspondiente

3. 


   En esta prueba podemos ver un solo paréntesis de cierre por lo que el resultado es falso porque falta el paréntesis inicial

4. 


   Aquí existe solo la cerradura de 1 paréntesis por lo que si queremos que sea true deberían haber 2 más

5. 

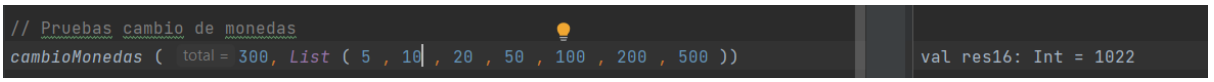

   Finalmente en el último caso de prueba la respuesta es true, ya que están los pares de paréntesis ()

**Mi opinión**: creo que con estos casos de prueba se puede verificar que tanto la función de balanceoParentesis y su función interna parentesis cumplen con su objetivo, el cual es analizar el texto ingresado y verificar si cada par de paréntesis está completo

- **Cambio de monedas**

El cambio de monedas consiste en que se ingresa cierta cantidad de dinero que debemos pagar y una lista que contiene billetes (la cual es la que tenemos) y la máquina nos debe arrojar las formas posibles de pagar ese dinero con los billetes

Tenemos 5 casos de prueba en los cuales vamos a verificar el cambio de monedas:

1. 


2. 


3. 


4. 


5. 


**Mi opinión:** Con estos casos de prueba se puede ver que se cumple el objetivo del programa que es analizar la lista de números de forma que no importe su cantidad