PRÁCTICO Nº 8

Introducción

El objetivo de este práctico es la comprensión de la metodología de programación llamada recursión mediante ejercicios de complejidad creciente. También se trata de mostrar los pros y contras de esta técnica.

Ejercicio 1

La metodología de resolución del siguiente ejercicio será la de ejecutar "manualmente" las funciones de cada parte, para que se entienda mejor cómo funciona la recursión. Luego deberá verificar sus conclusiones con el uso de la computadora.
Indique cuál es el resultado devuelto por las siguientes funciones cuando se les pasa el valor 5 como parámetro.

Parte a)

```
function y = Recursiva(x)
if x==0
     y = 1;
else
     if mod(x,2) == 0
         y = 1 + Recursiva(x-2);
     else
        y = 2 + Recursiva(x-1);
     end
end
```

Parte b)

```
function y = Recursiva(x)
if x==0 | x==1
     y = 3;
else
     a = Recursiva(x-1);
     b = Recursiva(x-2);
     y = a + b;
end
```

Ejercicio 2

Escriba una función recursiva que reciba un entero positivo como parámetro y retorne la suma de todos los enteros desde cero hasta él.

Ejercicio 3

Escriba una función recursiva que reciba un entero positivo y retorne la suma de todos los enteros impares desde cero hasta él.

Ejercicio 4

Escriba una función recursiva que reciba dos enteros $m > 0$ y $n >= 0$ y calcule m elevado a la n.

Ejercicio 5

Escriba una función recursiva que reciba un vector de N enteros y devuelva la suma de todos sus elementos.

Ejercicio 6

Escriba una función recursiva que reciba un vector de N enteros y un entero X, y devuelva la suma de todos los elementos del vector que son mayores que X.

Ejercicio 7

a. Escriba una función recursiva que reciba un vector de enteros y devuelva el máximo valor almacenado en el vector.
   i. Realice una estrategia donde se compare v(1) con el máx{v(2:length(v))}.
   ii. Realice una estrategia donde se compare v(length(v)) con el máx de v(1:length(v)-1)}
   iii. Realice una estrategia donde se compare el máx{v(1:j)} con el máx{v(j+1:length(v))}, con j un valor cercano a la mitad de la cantidad de elementos de v.

b. Escriba una función recursiva que reciba un vector de enteros y devuelva dos variables conteniendo el valor máximo y el valor mínimo del vector.

Ejercicio 8

Escriba una función recursiva que reciba un vector de enteros y un entero *X*, y devuelva la primera posición dentro del vector en la cual se encuentra *X*, o (-1) si *X* no se encuentra en el vector.

Ejercicio 9

a. Implemente una función **iterativa** que verifique si dos vectores son iguales (mismos elementos en el mismo orden).
b. Implementar una función **recursiva** que verifique si dos vectores son iguales (mismos elementos en el mismo orden).
c. Implementar una función **recursiva** que recibe dos vectores y verifica si uno es prefijo del otro.

Ejercicio 10

Implementar en Octave una función recursiva llamada S*eparar*P*aresImpares*. Dicha función recibe como único parámetro de entrada un vector de enteros; y devuelve como parámetros de salida dos vectores agrupando los números pares en el primer parámetro de salida, y los números impares en el segundo parámetro de salida. Se tiene que respetar el orden relativo entre los elementos de los vectores.

Ejemplo:

```
>>[a,b] = SepararParesImpares([1,8,6,2,1,7,9,3,5,1,9,2,7])

a =
        [8,6,2,2]
b =
        [1,1,7,9,3,5,1,9,7]
```

Ejercicio 11

Implementar en Octave una función recursiva llamada *JuntarParesImpares*. La misma recibe exactamente dos parámetros de entrada: el primer parámetro es un vector de enteros pares, y el segundo parámetro es un vector de enteros impares. La función deberá intercalar los elementos de ambos vectores para generar un vector en donde su primer elemento es par, el siguiente es impar, el siguiente es par... y así análogamente hasta que se terminen los elementos de uno de los dos vectores de entrada. Desde ese punto en delante todos los elementos del vector resultado van a ser solo pares o solo impares.

Ejemplo:

```
>>resultado = JuntarParesImpares([8,6,2,2],[1,1,7,9,3,5,1,9,7])
resultado =
        [8,1,6,1,2,7,2,9,3,5,1,9,7]
```

Ejercicio 12

a) Implemente en Octave la función recursiva *SepararRec* que, dado un vector v y un número *X*, devuelva dos vectores:
   → *IZQ*, un vector con todos los elementos de v menores o iguales a *X*.
   → *DER*, un vector con todos los elementos de v mayores a *X*.

b) Implemente en Octave la función recursiva *Ordenar* (conocida como Quicksort) que, dado un vector *v* cualquiera, lo devuelva ordenado. El ordenamiento debe realizarse de la siguiente manera:
   i. Se toma el primer elemento de v, denominado v_1.
   ii. Utilizando la función de la parte a) sobre el resto del vector v se obtienen dos vectores:
      - IZQ es el vector con los elementos menores o iguales a v_1.
      - DER es el vector con los elementos mayores a v_1.
   iii. El vector ordenado se obtiene concatenando el resultado de ordenar IZQ, v_1, y el resultado de ordenar DER.

Ejercicio 13 (opcional)

El objetivo de este ejercicio es aprender a utilizar la herramienta ***dispVariables***. Esta función de Octave permite depurar programas, para entender su funcionamiento y encontrar posibles errores.

Dada la siguiente función:

```
function ret = multiplos(numero)
	dispVariables;
	if numero == 0
		ret = 1;
	else
		if mod(numero,3) == 0
			x = multiplos(numero/3);
			ret = x + 3;
		elseif mod(numero,3) == 1
			y = multiplos((numero-1)/3);
			ret = y * 2;
		else
			z = multiplos((numero-2)/3);
			ret = z ^ 2;
		end
	end
	dispVariables;
```

a. Indicar qué se muestra en pantalla al ejecutar multiplos(15).

b. Comentar la primera línea en donde aparece ***dispVariables***. Indicar qué se muestra en pantalla al ejecutar la función multiplos(15).

c. Comentar la última línea en donde aparece ***dispVariables***. Indicar qué se muestra en pantalla al ejecutar la función multiplos(15).