Problemas de funciones y procedimientos

version 1.1 - 16/11/2020

1. Números Perfectos. (perfectos)

Se dice que un número es *perfecto* si es igual a la suma de sus divisores sin incluirle a él mismo. La función *esPerfecto* determina si un número natural *n* es perfecto o no.
Ejemplos:

> esPerfecto(6) = true, ya que sumaDivisores(6) = 1+2+3 = 6
> esPerfecto(8) = false, ya que sumaDivisores(8) = 1+2+4 = 7
> donde *sumaDivisores(n)* calcula la suma de los divisores de *n* en *[1, n-1]*.

Para describir el problema de una manera más cercana a la definición del programa:

> Un número natural *n* es *perfecto* si la suma de sus divisores en *[1, n-1]* es igual a él.

Se pide: desarrollar la función *esPerfecto* y un procedimiento *escribirPerfectos* que escriba los *n* primeros números perfectos.

2. Números Triangulares. (triangulares)

Se dice que un número es *triangular* si se puede expresar como la suma de un número determinado de números naturales consecutivos empezando en el 1. La función *esTriangular* determina si un número natural *n* es triangular o no.
Ejemplos:

> esTriangular(10) = true, ya que 10 = 1+2+3+4
> esTriangular(7) = false

Se pide: desarrollar la función *esTriangular* y un procedimiento *escribirTriangulares* que escriba los *n* primeros números triangulares.

Pautas:

Otra manera de enunciar este problema es diciendo:

> Un número natural *n* es *triangular* si existe un número *k, k IN [1, n]*, tal que la suma de los números entre *1* y *k* sea igual a *n*. Determinar si un número *n* es *triangular*.

Si partimos de la función *suma(a, b)* (*f0* en un ejercicio anterior):

> Calcula la suma de los números en *[a, b]*.

podemos utilizarla para redefinir el problema como:

> Un número natural *n* es *triangular* si existe un número *k, k IN [1, n]*, tal que *suma(1, k) = n*. Determinar si un número *n* es *triangular*.

3. Búsquedas de mayores. (mayores)

Desarrolla un módulo que tenga las tres funciones siguientes:

- existeMayor: determina si existe un número en una lista que sea mayor que un número n.
- todosMayores: determina si todos los números de una lista son mayores que un número n.
- posicionMayor: calcula la posición del primer número de una lista que es mayor que un número n; si no existe ninguno, devuelve el valor -1.

Y sus correspondientes versiones recursivas:

- existeMayorRec.
- todosMayoresRec.
- posicionMayorRec.

Prepara aparte un módulo de pruebas llamado mayoresTest.

4. Vocales. (vocales)

Se quiere desarrollar unas funciones para procesar frases de textos. Se considera una frase como una colección de palabras y una palabra como una colección de caracteres. Se supone que en una palabra solamente aparecen caracteres alfabéticos que pueden ser en minúsculas o en mayúsculas indistintamente.
Desarrolla un módulo que tenga las dos funciones siguientes:

- cuantasVocalesPalabra: calcula cuántas vocales tiene una palabra.
- cuantasVocalesFrase: calcula cuántas vocales tiene una frase.

Y sus correspondientes versiones recursivas:

- cuantasVocalesPalabraRec.
- cuantasVocalesFraseRec.

Nota: Aconsejable definir y utilizar también una función auxiliar esVocal.

5. Consonantes. (consonantes)

Se quiere desarrollar unas funciones para procesar frases de textos. Se considera una frase como una colección de palabras y una palabra como una colección de caracteres. Se supone que en una palabra solamente aparecen caracteres alfabéticos que pueden ser en minúsculas o en mayúsculas indistintamente.
Desarrolla un módulo que tenga las dos funciones siguientes:

- cuantasConsonantesPalabra: calcula cuántas consonantes tiene una palabra.
- cuantasConsonantesFrase: calcula cuántas consonantes tiene una frase.

Y sus correspondientes versiones recursivas:

- cuantasConsonantesPalabraRec.
- cuantasConsonantesFraseRec.

Nota: Aconsejable definir y utilizar también una función auxiliar esConsonante.