Auxiliar 5 pauta

Pregunta 1

En esta pregunta usted deberá crear una función que determine la cantidad de puntos que pertenecen a una cierta circunferencia. Para esto deberá seguir los siguientes pasos:

1. Cree la estructura **punto2D**, que contenga 2 parámetros (x, y).
2. Programe la función **distancia(p1, p2)**, que dado dos puntos2D determine la distancia entre ambos puntos.
3. Programe la función **pertenece(c, r, p)**, que dado un punto2D *p*, determine si este pertenece a la circunferencia de radio *r* y centro *c* (también de tipo punto2D). Un punto pertenece a una circunferencia, si este se encuentra dentro de la circunferencia o sobre su contorno.
4. Programe la función **cuantosPertenecen(c, r, l)**, que dado una lista de punto2D (*l*), determine la cantidad de punto2D que pertenecen a la circunferencia de radio *r* y centro *c*.

```
# P1.1

import estructura

# punto2D: x (float) y (float)
estructura.crear("punto2D", "x y")
```

```
# P1.2

import math

# distancia: punto2D punto2D -> float
# Calcula la distancia entre dos puntos
# ej.: distancia(punto2D(6,3), punto2D(3,-1)) debe retornar 5
def distancia(p1, p2):
    assert type(p1) == punto2D and type(p2) == punto2D

    return math.sqrt((p1.x - p2.x)**2 + (p1.y - p2.y)**2)


assert abs(distancia(punto2D(6,3), punto2D(3,-1)) - 5) <= 0.001
assert abs(distancia(punto2D(4,-3), punto2D(6,2)) - 5.385) <= 0.001
```

```
# P1.3
```

```
# pertenece: punto2D float punto2D -> boolean
# determina si un determinado punto pertenece a una circunferencia dada
# ej.: pertenece(punto2D(0,0), 10, punto2D(0,9)) debe retornar True
def pertenece(c, r, p):
    assert type(c) == punto2D and type(p) == punto2D and r > 0

    return distancia(c,p) <= r

assert pertenece(punto2D(0,0), 10, punto2D(0,9))
assert not pertenece(punto2D(0,0), 10, punto2D(-5,15))
```

```
# P1.4

from lista import *

# cuantosPertenecen: punto2D float list(punto2D) -> int
# determina cuantos puntos de una lista pertenecen a una cierta circunferencia
# ej.: cuantosPertenecen(punto2D(0,0), 10, lista(punto2D(0,9), lista(punto2D(2,12), listaVaci
def cuantosPertenecen(c, r, l, contador=0):
    assert type(c) == punto2D and r > 0 and (type(l) == lista or vacia(l))

    if vacia(l): # caso base
        return contador
    else: # caso recursivo
        if pertenece(c, r, cabeza(l)):
            contador += 1
        return cuantosPertenecen(c, r, cola(l), contador)


L1 = crearLista(punto2D(0,9), crearLista(punto2D(2,15), crearLista(punto2D(-3,5), listaVacia)
L2 = crearLista(punto2D(7,-9), crearLista(punto2D(-5,15), listaVacia))

assert cuantosPertenecen(punto2D(0,0), 10, L1) == 2
assert cuantosPertenecen(punto2D(0,0), 10, L2) == 0
assert cuantosPertenecen(punto2D(0,0), 10, listaVacia) == 0
```

Pregunta 2

Programe la función recursiva **evaluarPolinomio(l, x)**, la cual recibe una lista de números reales (que representan los coeficientes de un polinomio), y un número *x*, retornando el valor del polinomio al evaluarlo en *x*.

Por ejemplo, sea la lista:

L = crearLista(0, crearLista(-1, crearLista(1, listaVacia)))

la cual representa al polinomio *-x + x^2*, entonces **evaluarPolinomio(L, 2)** debe retornar 2.

```
# P2

import math

# evaluarPolinomio: lista(float) float -> float
# Retorna el valor del polinomio al evaluardo en cierto numero
# ej.: evaluarPolinomio(crearLista(1, crearLista(5, listaVacia)), 2) debe retornar 11
def evaluarPolinomio(l, x, evaluacion=0, exp=0):
    assert type(l) == lista or vacia(l)

    if vacia(l): # caso base
        return evaluacion
    else: # caso recursivo
        evaluacion += cabeza(l) * math.pow(x,exp)
        return evaluarPolinomio(cola(l), x, evaluacion, exp+1)

L1 = crearLista(1, crearLista(5, listaVacia))
L2 = crearLista(1, crearLista(0, crearLista(-3, listaVacia)))

assert evaluarPolinomio(L1, 2) == 11
assert evaluarPolinomio(L2, 3) == -26
assert evaluarPolinomio(listaVacia, 1231234) == 0
```