Auxiliar 7

```
from estructura import *
from lista import *
from abb import *
```

P1

Cree la función ***filtrarXY(abb, x, y)***, que dado un árbol de búsqueda binaria *abb*, retorna un nuevo árbol de búsqueda binaria que contenga solo aquellos valores de *abb* que están en el rango [x, y] (ambos incluidos). Recuerde realizar la receta de diseño y los asserts de su función

```
# filtrarXY: AB num num -> AB
# dado un ABB filtra aquellos valores que se encuentran dentro de un rango dado
# ej.: filtrarXY(AB(3, AB(1, None, None), AB(4, None, None)), 3, 5) retorna
# AB(3, None, AB(4, None, None))

def filtrarXY(abb, x, y, newAbb=None):
  if abb == None:
    return newAbb
  if x <= abb.valor and abb.valor <= y:
    newAbb = insertar(abb.valor, newAbb)
  newAbbIzq = filtrarXY(abb.izq, x, y, newAbb)
  return filtrarXY(abb.der, x, y, newAbbIzq)


# ASSERTS

A = AB(4, AB(2, AB(1, None, None), None), AB(8, AB(5, None, None), AB(10, AB(9, None, None),
B = AB(5, AB(3, AB(1, None, AB(2, None, None)), AB(4, None, None)), AB(8, AB(7, AB(6, None, N

assert filtrarXY(A, 6, 15) == AB(8, None, AB(10, AB(9, None, None), AB(12, None, None)))
assert filtrarXY(A, 1, 10) == AB(4, AB(2, AB(1, None, None), None), AB(8, AB(5, None, None),
assert filtrarXY(B, 3, 7) == AB(5, AB(3, None, AB(4, None, None)), AB(7, AB(6, None, None), N
assert filtrarXY(B, 1, 3) == AB(3, AB(1, None, AB(2, None, None)), None)
assert filtrarXY(None, 1, 5) == None
assert filtrarXY(A, 13, 15) == None
```

P2

Programe la función ***multiplos(abb, k)***, que dado un ABB de números enteros y un entero *k*, retorne una lista enlazada ordenada con todos aquellos valores de *abb* que sean múltiplo de *k*. Recuerde realizar la receta de diseño y los asserts de su función

```
# multiplos: AB int -> list(int)
# retorna una lista ordenada con todos los multiplos de un valor k
# Ej.: multiplos(AB(3, AB(1, None, None), AB(4, None, None)), 3) debe retornar
# lista(3, listaVacia)

def multiplos(abb, k, newList=listaVacia):
  if abb == None:
    return newList
  newListDer = multiplos(abb.der, k, newList)
  if abb.valor % k == 0:
    newListDer = crearLista(abb.valor, newListDer)
  return multiplos(abb.izq, k, newListDer)


# ASSERTS

A = AB(4, AB(2, AB(1, None, None), None), AB(8, AB(5, None, None), AB(10, AB(9, None, None),
B = AB(5, AB(3, AB(1, None, AB(2, None, None)), AB(4, None, None)), AB(8, AB(7, AB(6, None, N

assert multiplos(A, 2) == crearLista(2, crearLista(4, crearLista(8, crearLista(10, crearLista
assert multiplos(A, 3) == crearLista(9, crearLista(12, listaVacia))
assert multiplos(B, 4) == crearLista(4, crearLista(8, listaVacia))
assert multiplos(B, 2) == crearLista(2, crearLista(4, crearLista(6, crearLista(8, crearLista(
assert multiplos(None, 4) == listaVacia
assert multiplos(B, 11) == listaVacia
```

P3

En esta pregunta simularemos una agenda telefónica en formato ABB, por lo que cada valor del árbol será una persona (que tiene nombre, apellido y número telefónico). Usted debe considerar que el árbol se ordena en base al apellido de la persona, si dos personas tienen el mismo apellido entonces habrá que ordenar en base al nombre. Además, no habrán dos personas con mismo nombre y apellido en la agenda telefónica.

En base a lo anterior, usted deberá programar 3 funciones que permitirán simular ciertas funcionalidades de una agenda telefónica:

1. Cree la estructura *Persona*, la cual debe tener *nombre*, *apellido* y *telefono*
2. Cree la función ***agregarContacto(abb, p)***, la cual retorna un nuevo ABB pero con la persona *p* insertada en este.
3. Cree la función ***listaTelefonicaPorApellido(abb, apellido)*** que retorna una lista enlazada ordenada con todos los contactos (personas) que tengan dicho apellido.
4. Cree la función ***ConsultarTelefono(abb, apellido, nombre)***, que dado el nombre y apellido de una persona retorne el número telefónico de dicha persona.

```
# 1.
```

```
# Persona: nombre(str) apellido(str) telefono(str)
estructura.crear('Persona', 'nombre apellido telefono')
```

```
# 2.

# agregarContacto: AB Persona -> AB
# Agrega la persona p a la agenda telefonica
# Ej. agregarContacto(None, Persona('Juan', 'Perez', '123')) retorna
# AB(Persona('Juan', 'Perez', '123'), None, None)

def agregarContacto(abb, p):
  if abb == None:
    return AB(p, None, None)
  if p.apellido < abb.valor.apellido:
    return AB(abb.valor, agregarContacto(abb.izq, p), abb.der)
  elif p.apellido > abb.valor.apellido:
    return AB(abb.valor, abb.izq, agregarContacto(abb.der, p))
  else: # los apellidos son iguales, entonces se decide segun nombre
    if p.nombre < abb.valor.nombre:
      return AB(abb.valor, agregarContacto(abb.izq, p), abb.der)
    elif p.nombre > abb.valor.nombre:
      return AB(abb.valor, abb.izq, agregarContacto(abb.der, p))
    else: # la persona ya existe
      return abb # no hago nada


# ASSERT

p1 = Persona('Juanita', 'Perez', '12345')
p2 = Persona('Juan', 'Castro', '12367')
p3 = Persona('Manuel', 'Castro', '12333')
p4 = Persona('Alejandro', 'Muñoz', '125445')
p5 = Persona('Luis', 'Rodriguez', '125445')
p6 = Persona('Luchita', 'Abarca', '125445')

A = AB(p1, AB(p4, AB(p2, None, None), None), AB(p5, None, None))

assert agregarContacto(None, p1) == AB(p1, None, None)
assert agregarContacto(A, p6) == AB(p1, AB(p4, AB(p2, AB(p6, None, None), None), None), AB(p5
assert agregarContacto(A, p3) == AB(p1, AB(p4, AB(p2, None, AB(p3, None, None)), None), AB(p5
assert agregarContacto(A, p5) == A
```

```
# 3.

# listaTelefonicaPorApellido: AB str -> list(Persona)
# Retorna todos los contactos que tengan cierto apellido
# Ej.: sea A = AB(Persona('Juan', 'Perez', '123'), None, None)
# listaTelefonicaPorApellido(A, 'Perez') debe retornar
# crearLista(Persona('Juan', 'Perez', '123'), listaVacia)
```

```
def listaTelefonicaPorApellido(abb, apellido, newLista=listaVacia):
  if abb == None:
    return newLista
  newLista = listaTelefonicaPorApellido(abb.der, apellido, newLista)
  if abb.valor.apellido == apellido:
    newLista = crearLista(abb.valor, newLista)
  return listaTelefonicaPorApellido(abb.izq, apellido, newLista)


# ASSERT

p1 = Persona('Juanita', 'Perez', '12345')
p2 = Persona('Juan', 'Castro', '12367')
p3 = Persona('Manuel', 'Castro', '12333')
p4 = Persona('Alejandro', 'Muñoz', '125445')
p5 = Persona('Luis', 'Rodriguez', '125445')
p6 = Persona('Luchita', 'Abarca', '125445')

A = AB(p1, AB(p4, AB(p2, AB(p6, None, None), AB(p3, None, None)), None), AB(p5, None, None))

assert listaTelefonicaPorApellido(A, 'Castro') == crearLista(p2, crearLista(p3, listaVacia))
assert listaTelefonicaPorApellido(A, 'Abarca') == crearLista(p6, listaVacia)
assert listaTelefonicaPorApellido(A, 'Pereira') == listaVacia
assert listaTelefonicaPorApellido(None, 'Pereira') == listaVacia
```

```
# 4.

# consultarTelefono: AB str str -> str
# Retorna el numero telefono de una persona
# Ej.: sea A = AB(Persona('Juan', 'Perez', '123'), None, None)
# consultarTelefono(A, 'Perez', 'Juan') debe retornar '123'

def consultarTelefono(abb, apellido, nombre):
  if abb == None:
    return ''
  if abb.valor.apellido == apellido and abb.valor.nombre == nombre:
    return abb.valor.telefono
  elif abb.valor.apellido > apellido:
    return consultarTelefono(abb.izq, apellido, nombre)
  elif abb.valor.apellido < apellido:
    return consultarTelefono(abb.izq, apellido, nombre)
  else:
    if abb.valor.nombre > nombre:
      return consultarTelefono(abb.izq, apellido, nombre)
    elif abb.valor.nombre < nombre:
      return consultarTelefono(abb.der, apellido, nombre)


# ASSERT
```

```
p1 = Persona('Juanita', 'Perez', '12345')
p2 = Persona('Juan', 'Castro', '12367')
p3 = Persona('Manuel', 'Castro', '12333')
p4 = Persona('Alejandro', 'Muñoz', '125445')
p5 = Persona('Luis', 'Rodriguez', '125445')
p6 = Persona('Luchita', 'Abarca', '125445')

A = AB(p1, AB(p4, AB(p2, AB(p6, None, None), AB(p3, None, None)), None), AB(p5, None, None))

assert consultarTelefono(A, 'Castro', 'Juan') == '12367'
assert consultarTelefono(A, 'Castro', 'Manuel') == '12333'
assert consultarTelefono(A, 'Abarca', 'Luchita') == '125445'
assert consultarTelefono(A, 'Castro', 'Pamela') == ''
assert consultarTelefono(None, 'Zuñiga', 'Luis') == ''
```