```
// Parte 1 — Axiomatización de funciones recursivas sobre TADs básicos

// Ejercicio 1 (Repaso de funciones sobre secuencias)

// Extienda el tipo Secuencia(α) definiendo las siguientes operaciones:

// a) Duplicar , que dada una secuencia la devuelve duplicada elemento a elemento.
// Por ejemplo, Duplicar(a • b • c • <>) ≡ a • a • b • b • c • c • <>.

TAD SecuenciaExtendida(α)
    igualdad observacional:
    géneros secu_ext(α)
    exporta
    usa
    observadores básicos:
    generadores:

    otras operaciones:
        duplicar: secu(α) → secu_ext(α)   <== ???

    axiomas: ∀s: secu(α), ∀a:α
        duplicar(vacía?(<>)) ≡ <>
        duplicar(a • s) ≡ a • (a • duplicar(s))

// b) • ≤ •, que chequea si una secuencia es "menor o igual" a otra según el orden lexicográfico
para una relación de orden total sobre α dada.
//  Por ejemplo, si representamos palabras como secuencias de letras y el orden de α (las letras)
es el orden del alfabeto, palabra1 ≤ palabra2 debería indicar que palabra1 aparece en el
diccionario antes que palabra2.

    otras operaciones:
        • ≤ • : secu(α) × secu(α) → bool
    axiomas: ∀s,t: secu(α), ∀a,b:α
        <> ≤ <> ≡ true
        <> ≤ • ≡ true
        • ≤ <> ≡ false
        // Comparo el primer término de cada secuencia, y solo si son iguales, sigo preguntando
        // por el resto de la secuencia
        a • s ≤ b • t ≡ if a < b then true else (if b > a then false else s ≤ t fi) fi

// c) Reverso, que dada una secuencia devuelve su reverso (la secuencia dada vuelta).

    otras operaciones:
        reverso: secu(α) → secu(α)
    axiomas: ∀s,t: secu(α), ∀a,b:α
        reverso(<>) ≡ <>
        reverso(a • s) ≡ reverso(s) ∘ a

// d) Capicúa, que determina si una secuencia es capicúa.
// Por ejemplo, Capicúa(a • b • b • a • <>) ≡ Capicúa(1 • <>) ≡ Capicúa(<>) ≡ true.

    otras operaciones:
        capicua: secu(α) → bool
    axiomas:
        capicua(<>) ≡ true
        capicua(a • <>) ≡ true
        capicua(a • (<> ∘ b)) ≡ a = b
        capicua(a • (s ∘ b)) ≡ if a = b then capicua(s) else false fi

// e) EsPrefijo? , que chequea si una secuencia es prefijo de otra.

    otras operaciones:
        esPrefijo?: secu(α) × secu(α) → bool
    axiomas:
        esPrefijo?(<>, t) ≡ true
        esPrefijo?(a•s, <>) ≡ false
        esPrefijo?(a•s, b•t) ≡ if a = b then esPrefijo?(s, t) else false fi

// f) Buscar, que busca una secuencia dentro de otra. Si la secuencia buscada está una o más veces,
la función devuelve la posición de la primera aparición; si no está, la función se indefine.
    otras operaciones:
        buscar: secu(α) × secu(α) → bool
    axiomas:
        buscar(<>, <>) ≡ true
```

```
        buscar(a•s, <>) ≡ false
        buscar(a•s, b•t) ≡ if esPrefijo?(a•s, b•t) then true else esPrefijo?(a•s, t) fi


// g) EstáOrdenada? , que verifica si una secuencia está ordenada de menor a mayor.
    otras operaciones:
        estáOrdenada?: secu(α) → bool
    axiomas:
        estáOrdenada?(<>) ≡ true
        estáOrdenada?(a•<>) ≡ true
        estáOrdenada?(a • (b • s)) ≡ if a < b then estáOrdenada?(b•s) else false fi

// h) InsertarOrdenada, que dados una secuencia so (que debe estar ordenada) y un elemento a (de
género α) inserta a en so de manera ordenada.

    otras operaciones:
        insertarOrdenada: α × secu(α) sₒ → secu(α)   {estáOrdenada(sₒ)}
    axiomas:
        insertarOrdenada(a, <>) ≡ a • <>
        insertarOrdenada(a, b•s) ≡ if a < b then a•(b•s) else insertarOrdenada(a, s) fi


// i) CantidadApariciones, que dados una secuencia y un α devuelve la cantidad de apariciones del
elemento en la secuencia.
    otras operaciones:
        cantidadApariciones: secu(α) × α → nat
    axiomas:
        cantidadApariciones(a, <>) ≡ 0
        cantidadApariciones(a, b•s) ≡ if a=b then 1 + cantidadApariciones(a,s) else
cantidadApariciones(a,s)


// j) EsPermutación? , que chequea si dos secuencias dadas son permutación una de otra.
// Pista: utilizar CantidadApariciones.
// Resuelvo: Cuando cantidad de cada uno de sus elementos
    otras operaciones:
        esPermutación?: secu(α) × secu(α) → bool
        remover: α × secu(α) → secu(α)
    axiomas:
        remover(a, <>) ≡ <>
        remover(a, b•s) ≡ if a=b then remover(a, s) else b•remover(a, s) fi

        esPermutación?(<>, <>) ≡ true
        esPermutación?(a•s, <>) ≡ false
        esPermutación?(<>, b•t) ≡ false
        esPermutación?(a•s, t) ≡ if cantidadApariciones(a, a•s) = cantidadApariciones(a, t) then
                                        esPermutación?(remover(a,s), remover(a,t))
                                    else
                                        false
                                    fi

// k) Combinar , que dadas dos secuencias ordenadas devuelve una secuencia ordenada que resulta de
juntar sus elementos.
// Por ejemplo, Combinar(“acd”, “bef’) ≡ “abcdef”.
    otras operaciones:
        combinar: secu(α) sₒ × secu(α) tₒ → secu(α)      {estáOrdenada(sₒ) ∧ estáOrdenada(tₒ)}
    axiomas:
        combinar(<>, <>) ≡ <>
        combinar(a•s, <>) ≡ a•s
        combinar(<>, b•t) ≡ b•t
        combinar(a•s, b•t) ≡ if a < b then a•combinar(s, b•t) else b•combinar(a•s, t) fi
```