1. Ejercicio 1

Especificar en forma completa el TAD *NumeroRacional* que incluya al menos las operaciones aritméticas básicas (suma, resta, división, multiplicación).

```
TAD numeroRacional
    obs a : float   o real?
    obs b : float

    proc suma (in a: float, in b: float): float
        asegura res= a+b

    proc resta (in a: float, in b: float): float
        asegura res= a-b

    proc división (in a: float, in b: float): float
        asegura if b=0 then res=0 else res= a/b

    proc multiplicación (in a: float, in b: float): float
        asegura res= a*b
```

2. Ejercicio 2

Especificar TADs para las siguientes figuras geométricas. Tiene que contener las operaciones: rotar, trasladar y escalar y una más propuesta por mi.

2.1. Rectángulo (2D)

```

TAD Rectangulo {
    obs altura: real
    obs base: real
    obs angulo: real
    obs centro: tupla <real,real>

    proc crearRectangulo (in altura: real, in base: real, in angulo, in centro): rectangulo {
        requiere: {altura >= 0 && base >= 0}
        asegura: {res.altura = altura && res.base = base && res.angulo = angulo && res.centro == centro}
    }

    proc asignarAngulo (inout r: rectangulo, in angulo: real): rectangulo {
        requiere: {r = old(r) && 0 <= angulo < 2pi}
        asegura: {r.angulo = angulo}
        asegura: {r.base = old(r).base && r.altura = old(r).altura}
        asegura: {r.angulo = old(r).angulo && r.centro = old(r).centro}
    }

    proc rotar (inout r: rectangulo, in angulo: real): rectangulo {
        requiere: {r = old(r) && 0<= old(r).angulo + angulo <2pi}
        asegura: {r.angulo = old(r).angulo + angulo}
        asegura: {r.centro = old(r).centro }
        asegura: {r.base = old(r).base && r.altura = old(r).altura}
    }

    proc trasladar (inout r: rectangulo, in p: punto): rectangulo {
        requiere: {r = old(r)}
        asegura: {r.centro = punto}
        asegura: {r.base = n * old(r).base && r.altura = n * old(r).altura && r.angulo = old(r).angulo}
    }

    proc trasladar2 (inout r: Rectangulo, in diferencia: tupla <real, real>): rectangulo {
        requiere: {r == old(r)}
        asegura: {r.centro == old(r).centro + diferencia}
        asegura: {r.base == old(r).base && r.altura == old(r).altura && r.angulo == old(r).angulo}
```

```
        }

        proc escalar (inout r: Rectangulo, in n: real): Rectangulo {
            requiere: {r == old(r)}
            asegura: {r.altura == n * old(r).altura && r.base == n * old(r).base}
            asegura: {r.angulo == old(r).angulo && r.centro == old(r).centro}
        }

        proc esCuadrado (in r: rectangulo): Bool {
            requiere: {True}
            asegura: {res = true <--> r.altura = r.base}
        }
    }
```

2.2. Esfera (3D)

```
    TAD esfera{
        obs radio: real
        obs centro: tupla <real, real, real>
        obs rotación: tupla <real, real, real>

        proc crearEsfera (in centro: trupla <real, real, real>, in radio: real): esfera {
            requiere: {radio >= 0}
            asegura: {res.radio = radio && res.centro = centro}
        }

        proc rotar (inout e: esfera, in coordenadas: tupla <real, real, real>): esfera {
            requiere: {e = old(e)}
            requiere: {0 <= e.rotación + coordenadas < 2pi}
            asegura: {e.rotación = old(e).rotacion + coordenadas}
            asegura: {e.radio = old(e).radio}
            asegura: {e.centro = old(e).centro}
        }

        proc trasladar (inout e: esfera, in p: tupla <real, real, real>): esfera {
            requiere: {e = old(e)}
            asegura: {e.centro = p}
            asegura: {e.radio = old(e).radio}
            asegura: {e.rotacion = old(e).rotacion}
        }

        proc trasladar2 (inout e: Esfera, in diferencia: tupla<real,real,real>): Esfera {
            requiere: {e == old(e)}
            asegura: {e.centro == old(e).centro + diferencia}
            asegura: {e.radio == old(e).radio}
            asegura: {e.rotacion == old(e).rotacion}
        }

        proc escalar (inout e: esfera, in n: real): esfera {
            requiere: {e = old(e)}
            asegura: {e.centro = old(e).centro}
            asegura: {e.radio = n * old(e).radio}
            asegura: {e.rotacion = old(e).rotacion}
        }

        proc areaDeEsfera (in radio: real): real {
            requiere: {radio >= 0}
            asegura: {res = 4 * pi * radio ** 2}
        }

    }
```

3. Ejercicio 3

Especifique el TAD *DobleCola*$<T>$, en el que los elementos pueden insertarse al principio o al final y se eliminan por el medio. Ejemplo:

```
c := DobleCola<int>.NuevaDobleCola()

encolarAdelante(c, 1)           // c = <1>
encolarAdelante(c, 2)           // c = <2, 1>
encolarAtrás(c, 3)              // c = <2, 1, 3>

desencolar(c)                   // devuelve 1, c = <2, 3>
desencolar(c)                   // devuelve 2, c = <3>
desencolar(c)                   // devuelve 3, c = <>
```

```
TAD DobleCola <T> {
    obs q: seq <T>

    proc crearDobleCola (inout c: DobleCola): c {
        requiere {}
        asegura: {}
    }

    aux primerElemento (elems: seq <T>):
        if |elems| = {} then

    proc encolarAdelante (inout s: seq <T>, in e: T) {
        requiere: {s = old(s)}   -- comparar el estado previo (s0) con el estado post
        asegura: {s.q = concat (<e>, old(s).q)}
    }

    proc encolarAtrás (inout s: seq <T>, in e: T) {
        requiere: {s = old(s)}
        asegura: {s.q = concat (ols(s).q, <e>)}
    }

    proc desencolar (inout s: seq <T>): T {
        requiere: {s = old(s)}
        requiere: {s.q =/ {}}
        asegura: {s.q = subseq (old(s).q , 1, |old(s).q|)}
        asegura: {res = old(s)[medioDeLaSeq]}
    }

    aux medioDeLaSeq (s: seq <T>): T
        if s mod 2 = 0 then () else

}
```

4. Ejercicio 4

Especifique el TAD *DiccionarioConHistoria*. El mismo guarda, para cada clave, todos los valores que se asociaron con la misma a lo largo del tiempo (en orden).

```
TAD DiccionarioConHistoria {
    obs
}
```