Especificacion de TADs

Ejercicio 1 Especificar en forma completa el TAD `NumeroRacional` que incluya al menos las operaciones aritmeticas basicas

TAD `NumeroRacional` {

`obs n:` $\mathbb{Z}$

`obs d:` $\mathbb{Z}$

`proc suma (in a,b: NumeroRacional) : NumeroRacional`

`requiere` $\{true\}$

`asegura` $\{res = \frac{a.n*b.d+a.d*b.n}{b.d*a.d}\}$

`proc resta (in a,b: NumeroRacional) : NumeroRacional`

`requiere` $\{true\}$

`asegura` $\{res = \frac{a.n*b.d-a.d*b.n}{b.d*a.d}\}$

`proc multiplicacion (in a,b: NumeroRacional) : NumeroRacional`

`requiere` $\{true\}$

`asegura` $\{res = \frac{a.n*b.n}{b.d*a.d}\}$

`proc division (in a,b: NumeroRacional) : NumeroRacional`

`requiere` $\{b \neq 0\}$

`asegura` $\{res = \frac{a.n*b.d}{b.n*a.d}\}$

}

Ejercicio 2: Especificar TADs para las siguientes figuras geometricas. Tiene que contener las operaciones rotar, trasladar y escalar, y una mas propuesta por usted.

- Rectangulo (2D)
- Esfera (3D)

TAD `Rectangulo` {

`obs alto:` $\mathbb{R}$

`obs ancho:` $\mathbb{R}$

`obs posicion:` $\langle \mathbb{R} \times \mathbb{R} \rangle$

`obs angulo:` $\mathbb{R}$

`proc nuevoRectangulo (in h:` $\mathbb{R}$`, in w: float, in x:`$\mathbb{R}$`,in y:`$\mathbb{R}$`) : Rectangulo`

`requiere` $\{w \geq 0 \wedge h \geq 0\}$

`asegura` $\{res.ancho = w \wedge res.alto = h \wedge res.posicion = (x, y) \wedge r.angulo = 0\}$

`proc escalar (inout r: Rectangulo, mh:`$\mathbb{R}$`, mw:`$\mathbb{R}$`)`

`requiere` $\{r = R_0\}$

`asegura` $\{(mw \geq 0 \wedge mh \geq 0) \wedge_L r.posicion = R_0.posicion\}$

`asegura` $\{r.alto = |R_0.alto * mh| \wedge r.ancho = |R_0.ancho * mw|\}$

`asegura` $\{mw < 0 \wedge_L r.posicion_1 = R_0.posicion_1 + R_0.ancho * mw\}$

`asegura` $\{mh < 0 \wedge_L r.posicion_2 = R_0.posicion_2 + R_0.alto * mh\}$

`proc rotar (inout r: Rectangulo, in rad:` $\mathbb{R}$`)`

`requiere` $\{r = R_0\}$

`asegura` $\{r.angulo = R_0.angulo + rad\}$

`proc trasladar (inout r: Rectangulo, in pos:` $\langle x, y \rangle$`)`

`requiere` $\{r = R_0\}$

`asegura` $\{r.posicion = R_0.posicion + pos\}$

`proc area (in r: Rectangulo) :` $\mathbb{R}$

`asegura` $\{res = r.alto \cdot r.ancho\}$

}

TAD Esfera {

obs radio: $\mathbb{R}$

obs centro: $\langle \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rangle$

obs semieje: $\langle \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rangle$

proc nuevaEsfera (in r: $\mathbb{R}$, in pos:$\langle \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rangle$) : Esfera

asegura $\{res.radio = |r|\}$

asegura $\{res.centro = pos\}$

proc escalar (inout esfera: Esfera, in e: $\mathbb{R}$)

requiere $\{esfera = esfera_0\}$

asegura $\{esfera.radio = ESFERA_0.radio * |e|\}$

proc trasladar (inout esfera: Esfera, in pos:$\langle x, y, z \rangle$)

requiere $\{esfera = ESFERA_0\}$

asegura $\{esfera.centro = ESFERA_0.centro + pos\}$

proc rotar (inout esfera: Esfera, in angulosrad: $\langle \alpha, \beta, \gamma \rangle$)

requiere $\{esfera = ESFERA_0\}$

asegura $\{esfera.semieje = ESFERA_0.semieje + angulosrad\}$

}

Ejercicio 3. Especifique el TAD DobleCola$\langle T \rangle$, en el que los elementos pueden insertarse al principio o al final y se eliminan por el medio.

TAD DobleCola {

obs elems: $seq\langle T \rangle$

proc DobleColaVacia () : DobleCola

asegura $\{res = \langle\rangle\}$

proc encolarFinal (inout doblecola: DobleCola, in e: T)

requiere $\{doblecola = DOBLECOLA_0\}$

asegura $\{doblecola.elems = concat(DOBLECOLA_0.elems, \langle e \rangle)\}$

proc encolarInicio (inout doblecola: DobleCola, in e: T)

requiere $\{doblecola = DOBLECOLA_0\}$

asegura $\{doblecola.elems = concat(\langle e \rangle, DOBLECOLA_0.elems)\}$

proc desencolar (inout doblecola: DobleCola) : T

requiere $\{doblecola = DOBLECOLA_0\}$

requiere $\{cola.elems \neq \langle\rangle\}$

asegura $\{res \notin doblecola.elems\}$

asegura $\{res = DOBLECOLA_0.elems\left[\left\lceil \frac{|DOBLECOLA_0|}{2} \right\rceil\right]\}$

}

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 `DiccionarioConHistoriaT`,$seq\langle K\rangle$ {

`obs data: dict<T,`$seq\langle K\rangle$`>`

`proc DiccionarioConHistoriaT,`$seq\langle K\rangle$ `Vacio () : DiccionarioConHistoriaT,`$seq\langle K\rangle$
`asegura` $\{res.data = \{\}\}$

`proc estaLaLlave (in dic:DiccionarioConHistoriaT,`$seq\langle K\rangle$ `, in e: T) : Bool`
`asegura` $\{res = true \longleftrightarrow e \in dic.data\}$

`proc definir (inout dic: DiccionarioConHistoriaT,`$seq\langle K\rangle$ `,in k: T, in e: K)`
`requiere` $\{DIC_0 = dic \wedge_L k \in RES_0.data\}$
`asegura` $\{dic.data[k][0] = setKey(DIC_0.data, k, concat(DIC_0.data[k], e))\}$

`proc consultarHistorial (in dic: DiccionarioConHistoriaT,`$seq\langle K\rangle$ `, in k: T) :` $seq\langle K\rangle$
`asegura` $\{res = res.data[k]\}$

`proc borrar (inout dic DiccionarioConHistoriaT,`$seq\langle K\rangle$ `,in k: T)`
`requiere` $\{dic = DIC_0\}$
`requiere` $\{k \in dic.data\}$
`asegura` $\{dic.data = delKey(DIC_0, k)\}$

`proc tamaño (in dic: DiccionarioConHistoriaT,`$seq\langle K\rangle$ `) :` $\mathbb{Z}$
`asegura` $\{res = |dic.data|\}$

}

Ejercicio 5. Modifique el TAD ColaDePrioridad¡T¿para que, si hay muchos valores iguales al maximo, la operacion desapilarMax los desapile a todos.

TAD `ColaDePrioridad<T>` {

`---Toda la implementacion normal de ColaDePrioridad<T>---`

`proc desapilarMax (inout cola:ColaDePrioridad<T>)` : $seq\langle T\rangle$
`requiere` $\{cola = COLA_0\}$
`asegura` $\{\nexists\ e \in cola.data, \exists e' \in COLA_0.data : tienePriMax(COLA_0.data, e') \wedge_L cola.data[e] = COLA_0.data[e']\}$
`asegura` $\{(\forall e \in COLA_0.data)\ (tienePriMax(COLA_0.data, e) \wedge_L e \in res) \vee$
$(\neg tienePriMax(COLA_0.data, e) \wedge_L e \notin res)\}$
`asegura` $\{|res| = |COLA_0.data| - |cola.data|\}$

}

No estoy seguro de si se puede usar el operando $\in$ con diccionarios, ni si estoy asegurando que TODOS los elementos que saco de la cola sean incluidos en la sequencia de resultado, por ejemplo si tengo dos elementos en la cola con el mismo nombre y la misma prioridad, ambos van a la secuencia con la definicion que di? creo que no, que como uno ya pertenece el otro no necesariamente se agrega. ¿Comparar la cantidad que saco y el largo de la secuencia puede ser una opcion valida?

Ejercicio 6. Especifique los TADS indicados a continuacion pero utilizando los observadores propuestos

- Diccionario<K,V> observando con conjuntos (de tuplas)
- Conjunto<T> observando con funciones
- Pila<T> observando con diccionarios
- Punto observando con coordenadas polares

TAD Diccionario<K,V> {
obs datos: conj$\langle K \times V \rangle$

proc diccionarioVacio () : Diccionario<K,V>
requiere $\{true\}$
asegura $\{res.datos = \{\}\}$

proc agregar (inout dic: Diccionario<K,V>, in k: K, in v: V)
requiere $\{dic = DIC_0\}$
asegura $\{(k, v) \in dic.datos\}$

---Se puede hacer siquiera esto? Otro observador con otro conjunto de tuplas y apariciones? raro

}

TAD Conjunto<T> {
obs pertenece: Bool
obs largo: $\mathbb{Z}$

proc conjVacio () : Conjunto<T>
requiere $\{true\}$
asegura $\{res.largo = 0\}$

proc agregar (inout c: Conjunto<T>, in e: T)
requiere $\{c = C_0\}$
asegura $\{c.pertenece(e)\}$

proc sacar (inout c: Conjunto<T>, in e: T)
requiere $\{c = C_0\}$
asegura $\{\neg c.pertenece(e)\}$

proc unir (inout c: Conjunto<T>, in c': Conjunto<T>)
requiere $\{c = C_0\}$
asegura $\{\forall e \in T, c'.pertenece(e) : c.pertenece(e)\}$

proc restar (inout c: Conjunto<T>, in c': Conjunto<T>)
requiere $\{c = C_0\}$
asegura $\{\forall e \in T, c'.pertenece(e) : \neg c.pertenece(e)\}$

proc intersecar (inout c: Conjunto<T>, in c': Conjunto<T>)
requiere $\{c = C_0\}$
asegura $\{\forall e \in T, C_0.pertenece(e) \wedge c'.pertenece(e) \leftrightarrow c.pertenece(e)\}$

proc tamaño (in c: Conjunto<T>) : $\mathbb{N}$
requiere $\{true\}$
asegura $\{\exists! m \in \mathbb{Z} : \sum_{i=1}^{m} \text{if } c.pertenece(e) \text{ then } 1 \text{ else } 0 \text{ fi} = res, e \in T\}$
--- acá cree el observador largo porque muchas cosas raras tenia que hacer
asegura $\{res = c.largo\}$

}

TAD Pila<T> {
obs elems: dict<$\mathbb{N}$,T>

proc pilaVacia () : Pila<T>
requiere $\{true\}$
asegura $\{res.elems = \{\}\}$

proc vacia (in pila: Pila<T>) : Bool
requiere $\{true\}$
asegura $\{res = true \leftrightarrow pila.elems = \{\}\}$

proc apilar (inout pila: Pila<T>, in e: T)
requiere $\{pila = PILA_0\}$
asegura $\{|PILA_0.elems|+1 = |pila.elems| \wedge |pila.elems| \in pila.elems \wedge_L setKey(pila.elems, |pila.elems|, e)\}$

```
    proc desapilar (inout pila: Pila<T>) : T
          requiere {pila = PILA_0}
          requiere {pila.elems ≠ {}}
          asegura {|pila.elems| = |PILA_0.elems| − 1 ∧ |PILA_0.elems| ∉ pila.elems}
          asegura {res = PILA_0.elems[|PILA_0.elems|]}

    proc tope (in pila: Pila<T>) : T
          requiere {pila.elems ≠ {}}
          asegura {res = pila.elems[|pila.elems| − 1]}
}

TAD Punto {
    obs coords: ⟨ℝ × ℝ⟩
    pred igualdad (p1, p2: Punto) {
        (∃k ∈ ℤ : p1.coords_2 = p2.coords_2 + 2kπ) ∧ (p1.coords_1 = p2.coords_1)
    }
    proc crearPunto (in r: ℝ, in α: ℝ) : Punto
          requiere {r ≥ 0}
          asegura {res.coords = (r, α)}

    proc mover (inout p: Punto, in newcoords: ⟨ℝ × ℝ⟩)
          requiere {true}
          asegura {p.coords_1 = newcoords_1 ∧ p.coords_2 = newcoords_2}
}
```

Ejercicio 7. Especificar TADs para las siguientes estructuras:

- Multiconjunto<T> - Es igual a un conjunto pero con duplicados. Cada elemento tiene asociada una multiplicidad, que es la cantidad de veces que este aparece en la estrucutra. Tiene las mismas operaciones que un conjunto y ademas una que indica la multiplicidad del elemento.

```
TAD Multiconjunto {
      obs elems: dict<T,ℕ>

      proc conjuntoVacio () : Multiconjunto
            requiere {true}
            asegura {res.elems = {}}

      proc agregar (inout mc: Multiconjunto, in e: T)
            requiere {mc = MC_0}
            asegura {e ∈ MC_0.elems ∧_L mc.elems[e] = MC_0.elems[e] + 1}
            asegura {e ∉ MC_0.elems ∧_L setKey(mc, e, 1)}

      proc eliminar (inout mc: Multiconjunto)
            requiere {mc = MC_0}
            asegura {e ∈ MC_0.elems ∧_L MC_0.elems[e] = 1 ∧_L mc.elems = delKey(MC_0, e)}
            asegura {e ∈ M_0.elems ∧_L MC_0.elems[e] > 1 ∧_L mc.elems = setKey(MC_0, e, MC_0.elems[e] − 1)}

      proc multiplicidad (in mc: Multiconjunto, in e: T) : ℤ
            requiere {true}
            asegura {e ∈ mc ∧_L res = mc.elems[e]}
            asegura {e ∉ mc ∧_L res = 0}

      --- Hay mas operaciones pero creo que estas son las mas relevantes
}
```

- Multidict<K,V>: Misma idea pero para diccionarios, cada clave puede estar asociada a multiples valores. Los valores se definen de a uno, pero la operacion obtener debe devolver todos los valores asociados a una determinada clave.

  Sobre la nota(que no copie): una posible implementacion que se me ocurre es la de un taller en el que las keys son los operarios y los values son listas en las que estan los trabajos pendientes que le corresponden a cada uno, en una implementacion mas completa se podria hacer procs para mover los trabajos de la cola (porque la secuencia va a ser basicamente una cola en su comportamiento default) pero que tambien se pueda elegir trabajos especificos que hacer para darles prioridad.

  TAD `Multidict<K,V>` {

  `obs elems: dict<K,`$seq\langle V\rangle$`>`

  `proc multidictVacio () : Multidict<K,V>`

  `requiere` $\{true\}$

  `asegura` $\{res.elems = \{\}\}$

  `proc agregar (inout md: Multidict<K,V>, in k: K, in v: V)`

  `requiere` $\{md = MD_0\}$

  `asegura` $\{k \in MD_0.elems \wedge_L setKey(md.elems, k, concat(\langle v\rangle, MD_0.elems[k]))\}$

  `asegura` $\{k \notin MD_0.elems \wedge_L setKey(md.elems, k, \langle v\rangle)\}$

  `proc borrar (inout md: Multidict<K,V>, in k: K)`

  `requiere` $\{md = MD_0\}$

  `asegura` $\{k \notin MD_0 \wedge_L md = MD_0\}$

  `asegura` $\{k \in MD_0.elems \wedge |MD_0.elems| = 1 \wedge_L delKey(md.elems, k)\}$

  `asegura` $\{k \in MD_0.elems \wedge |MD_0.elems| > 0 \wedge_L$
  $setKey(md.elems, k, subseq(MD_0.elems, 1, |MD_0.elems|[k]))\}$

  `--- Creo que el resto de la implementacion es trivial, podria agregarse un booleando como parametr`
  `de borrar que determine si se borra la key completamente o si se borra valor a valor`

  }

Ejercicio 8. Especifique el TAD contadores que, dada una lista de eventos, permite contar la cantidad de veces que se produjo cada uno de ellos. La lista de eventos es fija. El TAD debe tener una operacion para incrementar el contador asociado a un evento y una operacion para conocer el valor actual del contador de dicho evento.

-Modifique el TAD para que sea posible guardar el valor del contador en un determinado momento del pasado. Si necesita conocer la fecha y hora actual puede pasarla como parametro a los procedimientos. Asuma que las dechas son numeros enteros (por ejemplo la cantidad de segundos desde el 1ro de enero del '70)

TAD `Contadores` {

`obs elems: dict<T,`$\mathbb{Z}$`>`

}