

# Trabajo Práctico 1

## Especificación de TADs

13 de abril de 2025

AED

### Grupo Almendra

Integrante	LU	Correo electrónico
Puodziunas, Bruno	309/23	puodziunasb@gmail.com
Ozzan Prieto, Luana Constanza	1444/23	luanaozzan@gmail.com
Piputto, Lucas Ignacio	1345/24	lucaspiputto@gmail.com
Yu, Patricio	1247/24	yupatricio0@gmail.com

**Facultad de Ciencias Exactas y Naturales**  
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (+54 11) 4576-3300

<http://www.exactas.uba.ar>

# 1. Renombres de tipo

En lugar de *tuplas* utilizaremos *structs* para una mejor legibilidad de la especificacion a lo largo del documento.

## 1.1. Tipo: *usuario*

El usuario consta de un *id* y una cantidad de *monedas*, ambos enteros positivos, realizaremos el siguiente renombre de tipo:

<i>usuario</i> <b>ES</b> struct $\langle id : \mathbb{Z}, monedas : \mathbb{Z} \rangle$
---

## 1.2. Tipo: *transaccion*

Por definicion del problema sabemos que una transaccion es una cuadru-pla que consta de un *id\_transaccion*, *id\_comprador*, *id\_vendedor*, *monto*, todos enteros positivos, realizaremos el siguiente renombre de tipo:

<i>transaccion</i> <b>ES</b> struct $\langle id\_transaccion : \mathbb{Z}, id\_comprador : \mathbb{Z}, id\_vendedor : \mathbb{Z}, monto : \mathbb{Z} \rangle$
---

## 1.3. Tipo: *bloque*

Un bloque tiene un *id\_bloque* que se representa con un entero positivo y puede tener hasta 50 *transacciones*:

<i>bloque</i> <b>ES</b> struct $\langle id\_bloque : \mathbb{Z}, transacciones : seq(transaccion) \rangle$
--

## 2. Definicion del TAD

```

TAD Berretacoin {
  obs bloques: seq⟨bloque⟩
  obs usuarios: seq⟨usuario⟩

  proc nuevoBerretacoin (in usuarios: seq⟨usuario⟩) : Berretacoin {
    requiere { |usuarios| ≥ 1 }
    requiere { (∀i : Z) ((0 ≤ i < |usuarios|) →L ((usuarios[i].id >
0) ∧L ¬(hayRepetidos(usuarios, usuarios[i])) ∧L (usuarios[i].monedas = 0))) }
    asegura { res.bloques = ⟨⟩ }
    asegura { res.usuarios = usuarios }
  }

  proc agregarBloque (inout b: Berretacoin, in transacciones: seq⟨transaccion⟩) {
    requiere { b = B0 }
    requiere { |transacciones| ≤ 50 }
    requiere { ((transacciones[0].id = 0) ∧ (∀i : Z) ((0 ≤ i < |transacciones| - 1) →L
(transacciones[i].id + 1 = transacciones[i + 1].id))) ∨ (transacciones = ⟨⟩) }
    requiere { (∀i : Z) ((0 ≤ i < |transacciones|) →L (transacciones[i].id_comprador ≠
transacciones[i].id_vendedor)) }
    requiere
{ (∀i : Z) ((0 ≤ i < |transacciones|) →L ((¬existiaUsuario(B0, transacciones[i].comprador)) →L ((∃j :
Z) ((0 ≤ j < i) →L (transacciones[i].comprador = transacciones[j].vendedor)))) ) }
    requiere { (∀i : Z) ((0 ≤ i < |b.usuarios|) →L ((∀j : Z) (((0 ≤ j < |transacciones|) ∧ ((b.usuarios[i].id =
transacciones[j].id_comprador) ∨ (b.usuarios[i].id = transacciones[j].id_vendedor)) →L
(ifThenElse(existiaUsuario(B0.usuarios, b.usuarios[i].id), B0.usuarios[i].monto, 0) +
montoRecibido(transacciones, usuarios[i], j) - montoGastado(transacciones, usuarios[i], j)) ≥ 0)))) ) }
    requiere { (|B0.bloques| ≤ 3000) →L ((|transacciones| > 0) ∧ (transacciones[0].monto =
1) ∧ (noGanoMonedaGratis(B0.bloques, transacciones[0].id_vendedor) ∧ (∀i : Z) ((1 ≤ i <
|transacciones|) →L (transacciones[i].comprador_id ≠ 0)))) ) }
    requiere { (|B0.bloques| > 3000) →L ((∀i : Z) ((0 ≤ i < |transacciones|) →L
(transacciones[i].comprador_id ≠ 0))) }
    asegura { True }
  }

  proc maximosTenedores (in b: Berretacoin) : seq⟨usuario⟩ {
    asegura
{ (∀i : Z) ((0 ≤ i < |b|) →L ((b.usuarios[i] ∈ res) ⇔ (esMaximoTenedor(b.usuarios, b.usuarios[i])))) }
  }

  proc montoMedio (in b: Berretacoin) : R {
    requiere { True }
    asegura { res = 0 ⇔ |b.bloques| = 0 }
    asegura
{ res = (montoTotalOperadoBloques(b.bloques)) / (cantTotalDeOperacionesBloques(b.bloques)) }
  }

  proc cotizacionAPesos (in cotizaciones: seq⟨Z⟩) : seq⟨Z⟩ {
    requiere { |cotizaciones| = |b.bloques| }
    requiere { (∀i : Z) ((0 ≤ i < |cotizaciones|) →L (cotizaciones[i] > 0)) }
    asegura { |res| = |cotizaciones| }
    asegura { (∀i : Z) ((0 ≤ i < |cotizaciones|) →L (res[i] =
cotizaciones[i] * montoTotalOperado(bloques[i].transacciones))) }
  }

  pred hayRepetidos (s: seq⟨usuario⟩, u: usuario) {
    ( ∑i=0|s|-1 ifThenElseFi(s[i].id = u.id, 1, 0) ) ≥ 2
  }

  pred nuevoUsuarioValido (s: seq⟨usuario⟩, u: usuario) {
    (u.monedas = 0) ∧ ((∀i : Z) ((0 ≤ i < |s|) →L (s[i].id ≠ u.id)))
  }

```

```

}
pred esMaximoTenedor (s: seq⟨usuario⟩, u: usuario) {
  (∀i : ℤ) ((0 ≤ i < |s|) →L (u.monedas ≥ s[i].monedas))
}
pred existiaUsuario (s: seq⟨usuario⟩, id_u: ℤ) {
  (∃i : ℤ) ((0 ≤ i < |s|) ∧L (s[i].id = id_u))
}
pred noGanoMonedaGratis (bloques: seq⟨bloque⟩, id_vendedor: ℤ) {
  (∃i : ℤ) ((0 ≤ i < |bloques|) →L (bloques[i].transacciones[0].vendedor ≠ id_vendedor))
}
aux montoRecibido (transacciones: seq⟨transaccion⟩, u: usuario, idx: ℤ) : ℤ =
  (∑k=0idx ifThenElse(transacciones[k].id_vendedor = u.id, transacciones[k].monto, 0));
aux montoGastado (transacciones: seq⟨transaccion⟩, u: usuario, idx: ℤ) : ℤ =
  (∑k=0idx ifThenElse(transacciones[k].id_comprador = u.id, transacciones[k].monto, 0));
aux montoTotalOperado (transacciones: seq⟨transaccion⟩) : ℤ =
  (∑k=0|transacciones|-1 transacciones[k].monto);
aux montoTotalOperadoBloques (bloques: seq⟨bloque⟩) : ℤ =
  (∑k=0|bloques|-1 montoTotalOperado(transaccionesDesde(ifThenElse(b.bloques[k].id ≤ 3000, 1, 0),
    b.bloques[k].transacciones)));
aux cantTotalDeOperacionesBloques (bloques: seq⟨bloque⟩) : ℤ =
  (∑k=0|bloques|-1 |b.bloques[k].transacciones| - ifThenElse(b.bloques[k].id ≤ 3000, 1, 0));
aux transaccionesDesde (indice: ℤ, transacciones: seq⟨transaccion⟩) : seq⟨transaccion⟩ =
  (subseq(transacciones, indice, |transacciones|));
}

```