

1. TAD \$Berretacoin

1.1. Renombre de tipos

id_bloque, id_transacción, id_comprador, id_vendedor, id_usuario, monto son \mathbb{Z}

Transacción es struct<trans: id_transacción, comp: id_comprador, vend: id_vendedor, mont: monto>

1.2. TAD

```
TAD $Berretacoin {
  obs bloques : dict<id_bloque, seq<Transacción>>
  proc crearBerretacoin () : $Berretacoin
    asegura {res.bloques= {}}

  proc agregarBloque (inout bc : $Berretacoin, in s : seq<Transacción>) :
    requiere {bc = BC0}
    requiere {(∀t:Transacción) (t ∈ s → esValida(t))}
    requiere {1 ≤ |s| ≤ 50}
    requiere {ordenadaCrecientemente(s)}
    requiere {¬hayIdsRepetidos(s)}
    requiere {todosIdsNuevos(bc.bloques, s)}
    requiere {|bc.bloques| < 3000 → esCreacion(s[0]) ∧ ¬recibioAntes(s[0].vend, bc.bloques) ∧
    (∀i:ℤ) ((0 < i < |s| →L s[i].comp ≠ 0))}
    requiere {|bc.bloques| ≥ 3000 → ¬(∃t:Transacción) (t ∈ s ∧ t.comp = 0)}
    requiere {nadieGastoDeMas(bc.bloques, s)}
    asegura {bc.bloques = setKey(BC0.bloques, |BC0.bloques| + 1, s)}

  proc maximosTenedores (in bc : $Berretacoin) : seq<id_usuario>
    requiere {bc.bloques ≠ {}}
    asegura {sinRepetidos(res) ∧ (∀id:id_usuario) (id ∈ res ↔ tieneMasOIgualQueTodos(id, bc.bloques))}

  proc montoMedio (in bc : $Berretacoin) : ℝ
    requiere {bc.bloques ≠ {}}
    asegura {res = montoTotal(bc.bloques) / #transacciones(bc.bloques)}

  proc cotizaciónAPesos (in bc : $Berretacoin, in s : seq<ℤ>) : seq<ℤ>
    requiere {todosPositivos(s)}
    requiere {bc.bloques ≠ {}}
    requiere {|s| = |bc.bloques|}
    asegura {|res| = |s|}
    asegura {(∀i:ℤ) (0 ≤ i < |res| →L res[i] = s[i] * ∑j=0|bc.bloques[i]|-1 bc.bloques[i][j].mont)}}
}
```

2. Predicados

esValida: verifica si una transacción es válida.

```
pred esValida (t : Transacción) {  
    t.trans > 0 ∧ t.comp ≥ 0 ∧ t.vend > 0 ∧ t.mont > 0 ∧ t.comp ≠ t.vend  
}
```

ordenadaCrecientemente: verifica si la secuencia dada está ordenada de forma creciente.

```
pred ordenadaCrecientemente (s : seq⟨Transacción⟩) {  
    (∀i, j : ℤ) (0 ≤ i < j < |s| →L s[i].trans < s[j].trans)  
}
```

hayIdsRepetidos: verifica si hay ids de transacción repetidos en una secuencia de transacciones.

```
pred hayIdsRepetidos (s : seq⟨Transacción⟩) {  
    (∃t, r : Transacción) (t ∈ s ∧ r ∈ s ∧ t ≠ r ∧ t.trans = r.trans)  
}
```

todosIdsNuevos : verifica que dada una secuencia s de transacciones, ningún id de transacción de las transacciones de s ya existiese en bloques.

```
pred todosIdsNuevos (bloques : dict⟨id_bloque, seq⟨Transacción⟩⟩, s : seq⟨Transacción⟩) {  
    (∀c : id_bloque) (c ∈ claves(bloques) →L (∀i : ℤ) (0 ≤ i < |bloques[c]| → (∀t : Transacción) (t ∈ s →L t.trans ≠  
    bloques[c][i].trans)))  
}
```

esCreacion: verifica si la transacción dada es de creación.

```
pred esCreacion (t : Transacción) {  
    t.comp = 0 ∧ t.mont = 1  
}
```

recibioAntes: dado un id de usuario, verifica si ese usuario fue vendedor en la primera transacción de algún bloque en bloques.

```
pred recibioAntes (usuario : id_usuario, bloques : dict⟨id_bloque, seq⟨Transacción⟩⟩) {  
    (∃c : id_bloque) (c ∈ claves(bloques) ∧L (∃st : seq⟨Transacción⟩) (st = bloques[c] ∧ st[0].vend = usuario))  
}
```

nadieGastoDeMas: verifica que, para cada transacción, el comprador tenga saldo suficiente antes de realizarla.

```
pred nadieGastoDeMas (bloques : dict⟨id_bloque, seq⟨Transacción⟩⟩, s : seq⟨Transacción⟩) {  
    (∀i : ℤ) (0 ≤ i < |s| →L s[i].comp = 0 ∨ calculaSaldo(s[i].comp, bloques, subseq(s, 0, i)) ≥ s[i].monto)  
}
```

sinRepetidos: verifica que no haya elementos repetidos dentro de una secuencia.

```
pred sinRepetidos (s : seq⟨ℤ⟩) {  
    (∀i, j : ℤ) ((0 ≤ i, j < |s| ∧L s[i] = s[j]) →L i = j)  
}
```

tieneMasOigualQueTodos: verifica que un id que es usuario de bloques tiene mayor o igual cantidad de monedas que todos los demás usuarios.

```
pred tieneMasOigualQueTodos (usuario : id_usuario, bloques : dict⟨id_bloque, seq⟨Transacción⟩⟩) {  
    esUsuario(usuario, bloques) ∧ (∀id : id_usuario) (esUsuario(id, bloques) →  
    calculaSaldo(usuario, bloques, ⟨⟩) ≥ calculaSaldo(id, bloques, ⟨⟩))  
}
```

esUsuario: verifica si, dado un id de usuario, este se encuentra en alguna transacción de los bloques.

```
pred esUsuario (usuario : id_usuario, bloques : dict⟨id_bloque, seq⟨Transacción⟩⟩) {  
    (∃c : id_bloque) (c ∈ claves(bloques) ∧L (∃t : Transacción) (t ∈ bloques[c] ∧ (t.comp = usuario ∨ t.vend = usuario)))  
}
```

todosPositivos: verifica que todos los valores de una secuencia sean positivos.

```
pred todosPositivos (s : seq⟨ℤ⟩) {  
    (∀i : ℤ) (0 ≤ i < |s| →L s[i] > 0)  
}
```

3. Auxiliares

calculaGasto: dado un usuario calcula sus gastos en los bloques y secuencia ingresados.

```
aux calculaGasto (usuario : id_usuario, bloques : dict⟨id_bloque, seq⟨Transacción⟩⟩, st :
seq⟨Transacción⟩) : monto =

$$\sum_{c \in \text{claves}(\text{bloques})} \sum_{i=0}^{|\text{bloques}[c]|-1} \text{if } \text{bloques}[c][i].\text{comp} = \text{usuario} \text{ then } \text{bloques}[c][i].\text{mont}$$

else 0 fi +  $\sum_{i=0}^{|\text{st}|-1} \text{if } \text{st}[i].\text{comp} = \text{usuario} \text{ then } \text{st}[i].\text{mont} \text{ else } 0 \text{ fi};$ 
```

calculaGanacia: dado un usuario calcula sus ganancias en los bloques y secuencia ingresados.

```
aux calculaGanancia (usuario : id_usuario, bloques : dict⟨id_bloque, seq⟨Transacción⟩⟩,
st : seq⟨Transacción⟩) : monto =

$$\sum_{c \in \text{claves}(\text{bloques})} \sum_{i=0}^{|\text{bloques}[c]|-1} \text{if } \text{bloques}[c][i].\text{vend} = \text{usuario} \text{ then } \text{bloques}[c][i].\text{mont}$$

else 0 fi +  $\sum_{i=0}^{|\text{st}|-1} \text{if } \text{st}[i].\text{vend} = \text{usuario} \text{ then } \text{st}[i].\text{mont} \text{ else } 0 \text{ fi};$ 
```

calculaSaldo: resta de las dos funciones anteriores

```
aux calculaSaldo (usuario: id_usuario, bloques: dict⟨id_bloque, seq⟨Transacción⟩⟩,
st: seq⟨Transacción⟩) : monto =
calculaGanancia(usuario, bloques, st) - calculaGasto(usuario, bloques, st);
```

montoTotal : dado bloques, suma los montos de todas las transacciones.

```
aux montoTotal (bloques : dict⟨id_bloque, seq⟨Transacción⟩⟩) : monto =

$$\sum_{c \in \text{claves}(\text{bloques})} \sum_{i=0}^{|\text{bloques}[c]|-1} \text{if } \text{esCreacion}(\text{bloques}[c][i]) \text{ then } 0 \text{ else } \text{bloques}[c][i].\text{mont} \text{ fi};$$

```

#transacciones : dado bloques devuelve el total de transacciones que hubo.

```
aux #transacciones (bloques : dict⟨id_bloque, seq⟨Transacción⟩⟩) :  $\mathbb{Z}$  =

$$\sum_{c \in \text{claves}(\text{bloques})} \sum_{i=0}^{|\text{bloques}[c]|-1} \text{if } \text{esCreacion}(\text{bloques}[c][i]) \text{ then } 0 \text{ else } 1 \text{ fi};$$

```