



Este apunte contiene la especificación de los los TADs que se pueden usar en la materia dándolos por especificados:

```
TAD Conjunto $\langle T \rangle$  {  
  obs elems: conj $\langle T \rangle$   
  proc conjVacío () : Conjunto $\langle T \rangle$  {  
    asegura {res.elems =  $\langle \rangle$ }  
  }  
  
  proc pertenece (in c : Conjunto $\langle T \rangle$ , in e : T) : Bool {  
    asegura {res = true  $\leftrightarrow$  e  $\in$  c.elems}  
  }  
  
  proc agregar (inout c : Conjunto $\langle T \rangle$ , in e : T) {  
    requiere {c = C0}  
    asegura {c.elems = C0.elems  $\cup$   $\langle e \rangle$ }  
  }  
  
  proc sacar (inout c : Conjunto $\langle T \rangle$ , in e : T) {  
    requiere {c = C0}  
    asegura {c.elems = C0.elems -  $\langle e \rangle$ }  
  }  
  
  proc unir (inout c : Conjunto $\langle T \rangle$ , in c' : Conjunto $\langle T \rangle$ ) {  
    requiere {c = C0}  
    asegura {c.elems = C0.elems  $\cup$  c'.elems}  
  }  
  
  proc restar (inout c : Conjunto $\langle T \rangle$ , in c' : Conjunto $\langle T \rangle$ ) {  
    requiere {c = C0}  
    asegura {c.elems = C0.elems - c'.elems}  
  }  
  
  proc intersecar (inout c : Conjunto $\langle T \rangle$ , in c' : Conjunto $\langle T \rangle$ ) {  
    requiere {c = C0}  
    asegura {c.elems = C0.elems  $\cap$  c'.elems}  
  }  
  
  proc agregarRápido (inout c : Conjunto $\langle T \rangle$ , in e : T) {  
    requiere {c = C0  $\wedge$  e  $\notin$  c.elems}  
    asegura {c.elems = C0.elems  $\cup$   $\langle e \rangle$ }  
  }  
  
  proc tamaño (in c : Conjunto $\langle T \rangle$ ) :  $\mathbb{Z}$  {  
    asegura {res = |c.elems|}  
  }  
}
```

```

TAD Diccionario $\langle K, V \rangle$  {
  obs data: dict $\langle K, V \rangle$ 
  proc diccionarioVacío () : Diccionario $\langle K, V \rangle$  {
    asegura {res.data = {}}
  }

  proc está (in d : Diccionario $\langle K, V \rangle$ , in k : K) : Bool {
    asegura {res = true  $\leftrightarrow$  k  $\in$  d.data}
  }

  proc definir (inout d : Diccionario $\langle K, V \rangle$ , in k : K, in v : V) {
    requiere {d = D0}
    asegura {d.data = setKey(D0.data, k, v)}
  }

  proc obtener (in d : Diccionario $\langle K, V \rangle$ , in k : K) : V {
    requiere {k  $\in$  d.data}
    asegura {res = d.data[k]}
  }

  proc borrar (inout d : Diccionario $\langle K, V \rangle$ , in k : K) {
    requiere {d = D0  $\wedge$  k  $\in$  d.data}
    asegura {d.data = delKey(D0.data, k)}
  }

  proc definirRápido (inout d : Diccionario $\langle K, V \rangle$ , in k : K, in v : V) {
    requiere {d = D0  $\wedge$  k  $\notin$  d.data}
    asegura {d.data = setKey(D0.data, k, v)}
  }

  proc tamaño (in d : Diccionario $\langle K, V \rangle$ ) :  $\mathbb{Z}$  {
    asegura {res = |d.data|}
  }
}

```

```

TAD Cola⟨T⟩ {
  obs s: seq⟨T⟩
  proc colaVacía () : Cola⟨T⟩ {
    asegura {res.s = ⟨⟩}
  }

  proc vacía (in c : Cola⟨T⟩) : Bool {
    asegura {res = true ↔ c.s = ⟨⟩}
  }

  proc encolar (inout c : Cola⟨T⟩, in e : T) {
    requiere {c = C0}
    asegura {c.s = concat(C0.s, ⟨e⟩)}
  }

  proc desencolar (inout c : Cola⟨T⟩) : T {
    requiere {c = C0 ∧ c.s ≠ ⟨⟩}
    asegura {c.s = subseq(C0.s, 1, |C0.s|)}
    asegura {res = C0[0]}
  }

  proc proximo (in c : Cola⟨T⟩) : T {
    requiere {c = C0 ∧ c.s ≠ ⟨⟩}
    asegura {res = C0.s[0]}
  }
}

```

```

TAD Pila⟨T⟩ {
  obs s: seq⟨T⟩
  proc pilaVacía () : Pila⟨T⟩ {
    asegura {res.s = ⟨⟩}
  }

  proc vacía (in p : Pila⟨T⟩) : Bool {
    asegura {res = true ↔ p.s = ⟨⟩}
  }

  proc apilar (inout p : Pila⟨T⟩, in e : T) {
    requiere {p = P0}
    asegura {p.s = concat(P0.s, ⟨e⟩)}
  }

  proc desapilar (inout p : Pila⟨T⟩) : T {
    requiere {p = P0 ∧ p.s ≠ ⟨⟩}
    asegura {p.s = subseq(P0.s, 0, |P0.s| - 1)}
    asegura {res = P0.s[|P0.s| - 1]}
  }

  proc tope (in p : Pila⟨T⟩) : T {
    requiere {p = P0 ∧ p.s ≠ ⟨⟩}
    asegura {res = P0.s[|P0.s| - 1]}
  }
}

```

```

TAD ColaPrioridad⟨T⟩ {
  obs d: dict⟨T, ℝ⟩
  proc ColaPrioridadVacía () : ColaPrioridad⟨T⟩ {
    asegura {res.d = {}}
  }

  proc vacía (in c : ColaPrioridad⟨T⟩) : Bool {
    asegura {res = true ↔ c.d = {}}
  }

  proc encolar (inout c : ColaPrioridad⟨T⟩, e : T, in pri : ℝ) {
    requiere {c = C0}
    requiere {e ∉ c.d}
    asegura {c.d = setKey(C0.d, e, pri)}
  }

  proc desencolarMax (inout c : ColaPrioridad⟨T⟩) : T {
    requiere {c = C0}
    requiere {c.d ≠ {}}
    asegura {c.d = delKey(C0.d, res)}
    asegura {tienePriMax(C0.d, res)}
  }

  pred tienePriMax (d : dict⟨T, ℝ⟩, e : T) {
    e ∈ d ∧L (∀e' : T)(e' ∈ d →L d[e] ≥ d[e'])
  }
}

```

```

TAD Secuencia⟨T⟩ {
  obs s: seq⟨T⟩
  proc secuenciaVacía () : Secuencia⟨T⟩ {
    asegura {res.s = ⟨⟩}
  }

  proc agregarAdelante (inout s : Secuencia⟨T⟩, in e : T) {
    requiere {s = S0}
    asegura {s.s = concat(⟨e⟩, S0.s)}
  }

  proc agregarAtrás (inout s : Secuencia⟨T⟩, in e : T) {
    requiere {s = S0}
    asegura {s.s = concat(S0.s, ⟨e⟩)}
  }

  proc vacía (in s : Secuencia⟨T⟩) : Bool {
    asegura {res = true ↔ s.s = ⟨⟩}
  }

  proc fin (inout s : Secuencia⟨T⟩) {
    requiere {s = S0}
    requiere {|s.s| > 0}
    asegura {s = tail(S0)}
  }
}

```

```

proc comienzo (inout  $s$  : Secuencia( $T$ )) {
  requiere  $\{s = S_0\}$ 
  requiere  $\{|s.s| > 0\}$ 
  asegura  $\{s = head(S_0)\}$ 
}

proc primero (in  $s$  : Secuencia( $T$ )) :  $T$  {
  requiere  $\{|s.s| > 0\}$ 
  asegura  $\{res = s[0]\}$ 
}

proc último (in  $s$  : Secuencia( $T$ )) :  $T$  {
  requiere  $\{|s.s| > 0\}$ 
  asegura  $\{res = s[|s| - 1]\}$ 
}

proc longitud (in  $s$  : Secuencia( $T$ )) :  $\mathbb{Z}$  {
  asegura  $\{res = |s.s|\}$ 
}

proc obtener (in  $s$  : Secuencia( $T$ ), in  $i$  :  $\mathbb{Z}$ ) :  $T$  {
  requiere  $\{0 \leq i < |s.s|\}$ 
  asegura  $\{res = s[i]\}$ 
}

proc eliminar (inout  $s$  : Secuencia( $T$ ), in  $i$  :  $\mathbb{Z}$ ) {
  requiere  $\{s = S_0\}$ 
  requiere  $\{0 \leq i < |s.s|\}$ 
  asegura  $\{s.s = concat(subseq(S_0.s, 0, i - 1), subseq(S_0.s, i + 1, |S_0.s|))\}$ 
}

proc copiar (in  $s$  : Secuencia( $T$ )) : Secuencia( $T$ ) {
  asegura  $\{res.s = s.s\}$ 
}

proc modificarPosición (inout  $s$  : Secuencia( $T$ ), in  $i$  :  $\mathbb{Z}$ , in  $e$  :  $T$ ) {
  requiere  $\{s = S_0\}$ 
  requiere  $\{0 \leq i < |s.s|\}$ 
  asegura  $\{s.s = concat(subseq(S_0.s, 0, i - 1), \langle e \rangle, subseq(S_0.s, i + 1, |S_0.s|))\}$ 
}

proc concatenar (inout  $s$  : Secuencia( $T$ ), in  $s'$  : Secuencia( $T$ )) {
  requiere  $\{s = S_0\}$ 
  asegura  $\{s.s = concat(S_0.s, s'.s)\}$ 
}

```