Especificación TADs

Esta sección contiene la mayoría de los TADs que vamos a ver en la materia

- Conjunto
- Diccionario
- Cola
- Pila
- Cola de prioridad

```
TAD Conjunto<T> {
    obs elems: conj<T>

    proc conjVacio(): Conjunto<T>
        asegura res.elems == {}

    proc pertenece(in c: Conjunto<T>, in T e): bool
        asegura res == true <==> e in c.elems

    proc agregar(input c: Conjunto<T>, in e: T)
        asegura c.elems == old(c).elems + {e}

    proc sacar(inout c: Conjunto<T>, in e: T)
        asegura c.elems == old(c).elems - {e}

    proc unir(inout c: Conjunto<T>, in c': Conjunto<T>)
        asegura c.elems == old(c).elems + c'.elems

    proc restar(inout c: Conjunto<T>, in c': Conjunto<T>)
        asegura c.elems == old(c).elems - c'.elems
}
```

```
TAD Diccionario<K, V> {
    obs data: dict<K, V>

    proc diccionarioVacio(): Diccionario<K, V>
        asegura res.data == {}

    proc esta(in d: Diccionario<K, V>, in k: K): bool
        asegura res == true <==> k in d.data

    proc definir(inout d: Diccionario<K, V>, in k: K k, in v: V)
        asegura d.data == setKey(old(d).data, k, v)

    proc obtener(in d: Diccionario<K, V>, in k: K): V
        requiere k in d.data
        asegura res == d.data[k]

    proc borrar(inout d: Diccionario<K, V>, in k: K)
        requiere k in d.data
        asegura d.data == delKey(old(d).data, k)
}
```

```
TAD Cola<T> {
    obs s: seq<T>

    proc colaVacía(): Cola<T>
        asegura res.s == []

    proc vacia(in c: Cola<T>): bool
        asegura res == true <==> c.s == []

    proc encolar(inout c: Cola<T>, in e: T)
        asegura c.s == old(c).s + [e]

    proc desencolar(inout c: Cola<T>): T
        requiere c.s != []
        asegura c.s == old(c).s[1..|old(c).s|]
        asegura res == old(c)[0]
}
```

```
TAD Pila<T> {
    obs s: seq<T>

    proc pilaVacia(): Pila<T>
        asegura res.s == []

    proc vacia(in p: Pila<T>): bool
        asegura res == true <==> p.s == []

    proc apilar(inout p: Pila<T>, in e: T)
        asegura p.s == old(p).s + [e]

    proc desapilar(inout p: Pila<T>): T
        requiere p.s != []
        asegura p.s == old(p).s[0..|old(p).s|-1]
        asegura res == old(p).s[|old(p).s|-1]
}
```

```
TAD ColaPrioridad<T> {
    obs s: seq<T>

    proc ColaPrioridadVacía(): ColaPrioridad<T>
        asegura res.s == []

    proc vacia(in c: ColaPrioridad<T>): bool
        asegura res == true <==> c.s == []

    proc apilar(inout c: ColaPrioridad<T>, e: T)
        asegura c.s == old(c).s + [e]

    proc desapilarMax(inout c: ColaPrioridad<T>): T
        requiere c.s != []
        asegura esMax(old(c).s, res)
```

```
        asegura exists i: int :: 0 <= i < |c.s| && c.s[i] == res &&
            c.s == old(c).s[0..i] + old(c).s[i+1..|old(c).s|]

    pred esMax(s: seq<T>, res: T) {
        res in s && forall e: T :: e in s ==> e <= res
    }
}
```