a) TAD Lista enlazada:

* TAD ListaEnlazada
* Igualdad Observacional: Dos listas enlazadas son observacionalmente iguales si tienen la misma longitud y los mismos elementos en el mismo orden.
* Usa: Bool (para observadores que devuelven verdadero o falso), Nat (para medir longitud), Elemento (tipo genérico que representa los valores almacenados).
* Parámetro formal: a (tipo de los elementos contenidos en la lista)
* Género: Lista<a> (representa una lista de elementos del tipo a)
* Observadores básicos:
* es\_vacía(lista: Lista<a>) -> Bool (para saber si una lista no tiene elementos)
* inicio(lista: Lista<a>) -> a (para acceder al primer elemento de la lista (solo si no está vacía))
* resto(lista: Lista<a>) -> Lista<a> (para obtener una sublista sin el primer elemento)
* Generadores:
* vacía() -> Lista<a> (genera una lista vacía)
* agregar\_al\_inicio(elem: a, lista: Lista<a>) -> Lista<a> (agrega un elemento al inicio de la lista)
* Otras operaciones:
* longitud(lista: Lista<a>) -> Nat (cuenta cuantos elementos hay)
* pertenece(elem: a, lista: Lista<a>) -> Bool (verifica si un número está presente)
* concatenar(lista1: Lista<a>, lista2: Lista<a>) -> Lista<a> (une dos listas)
* Axiomas:
* es\_vacía(vacia()) -> true
* es\_vacía(agregar\_al\_inicio(elemento, lista)) -> false
* primero(agregar\_al\_inicio(elemento, lista)) -> elemento
* resto(agregar\_al\_inicio(elemento, lista)) -> lista
* longitud(vacia()) -> 0 (define la base de una lista)
* longitud(agregar\_al\_inicio(elemento, lista)) -> 1 + longitud(lista) (define el caso recursivo)
* pertenece(elemento, vacia()) -> false (usa recursión y comparación)
* pertenece(elemento, agregar\_al\_inicio(x, lista)) -> (elemento = x) or pertenece(elemento, lista)
* concatenar(vacia(), lista2) -> lista2 (define la combinación de listas paso a paso)
* concatenar(agregar\_al\_inicio(elemento, lista1), lista2) -> agregar\_al\_inicio(elemento, concatenar(lista1, lista2))
* Exporta: Lista<a>, vacía, agregar\_al\_inicio, es\_vacía, primero, resto, longitud, pertenece, concatenar

b) TAD Lista doblemente enlazada:

* TAD ListaDobleEnlazada
* Igualdad Observacional: Dos listas doblemente enlazadas son observacionalmente iguales si tienen la misma longitud y los mismos elementos en el mismo orden
* Usa: Bool, Nat, Elemento (tipo genérico que representa los valores almacenados)
* Parámetro formal: a (tipo de los elementos contenidos en la lista)
* Género: ListaDoble<a> (representa una lista doblemente enlazada de elementos de tipo a)
* Observadores básicos:
* es\_vacía(lista: ListaDoble<a>) -> Bool (para saber si una lista no tiene elementos)
* primero(lista: ListaDoble<a>) -> a (observan desde el inicio)
* ultimo(lista: ListaDoble<a>) -> a (observan desde el final)
* resto(lista: ListaDoble<a>) -> ListaDoble<a> (observan desde el inicio)
* inicio(lista: ListaDoble<a>) -> ListaDoble<a> (observan desde el final)
* Generadores:
* vacía() -> ListaDoble<a> (base de toda lista)
* agregar\_al\_inicio(elemento: a, lista: ListaDoble<a>) -> ListaDoble<a> (genera la lista desde el frente)
* agregar\_al\_final(elemento: a, lista: ListaDoble<a>) -> ListaDoble<a> (permite construir desde atrás (gracias al doble enlace))
* Otras operaciones:
* longitud(lista: ListaDoble<a>) -> Nat (mide el tamaño)
* pertenece(elemento: a, lista: ListaDoble<a>) -> Bool (evalúa si un elemento está presente)
* Axiomas:
* es\_vacia(vacia()) -> true
* es\_vacia(agregar\_al\_inicio(elemento, lista)) -> false
* es\_vacia(agregar\_al\_final(elemento, lista)) -> false
* primero(agregar\_al\_inicio(elemento, lista)) -> elemento (muestran los extremos)
* ultimo(agregar\_al\_final(elemento, lista)) -> elemento (muestran los extremos)
* resto(agregar\_al\_inicio(elemento, lista)) -> lista (eliminan un extremo)
* inicio(agregar\_al\_final(elemento, lista)) -> lista (eliminan un extremo)
* longitud(vacia()) = 0
* longitud(agregar\_al\_inicio(elemento, lista)) -> 1 + longitud(lista) (crece con cada inserción)
* longitud(agregar\_al\_final(elemento, lista)) -> 1 + longitud(lista) (crece con cada inserción)
* pertenece(elemento, vacia()) -> false
* pertenece(elemento, agregar\_al\_inicio(x, lista)) -> (elemento = x) or pertenece(elemento, lista)
* Exporta: ListaDoble<a>, vacia, agregar\_al\_inicio, agregar\_al\_final, es\_vacia, primero, ultimo, resto, inicio, longitud, pertenece

c) TAD Fecha:

* TAD Fecha
* Igualdad Observacional: Dos fechas son observacionalmente iguales si tienen el mismo día, mes, año y horario.
* Usa: Nat (se usa para representar el día, mes y año), Bool (se usa para operaciones lógicas)
* Parámetro formal: No aplica (no necesita ser genérico, porque no almacena elementos de otro tipo como una lista o una pila. Su estructura interna está bien definida: día, mes y año.)
* Género: Fecha (representa una instancia del tipo de dato Fecha)
* Observadores básicos:
* dia(f: Fecha) -> Nat
* mes(f: Fecha) -> Nat
* año(f: Fecha) -> Nat
* Generadores:
* crear\_fecha(d: Nat, m: Nat, a: Nat) -> Fecha (Es la forma de construir cualquier fecha a partir de sus componentes)
* Otras operaciones:
* es\_bisiesto(f: Fecha) -> Bool (permite realizar un cálculo típico con fechas)
* misma\_fecha(f1: Fecha, f2: Fecha) -> Bool (permite comparar dos fechas, útil para condiciones, búsquedas, validaciones, etc)
* Axiomas:
* dia(crear\_fecha(dia, mes, año)) -> dia
* mes(crear\_fecha(dia, mes, año)) -> mes
* año(crear\_fecha(dia, mes, año)) -> año
* es\_bisiesto(fecha) -> (año(fecha) módulo 4 = 0) and ((año(fecha) módulo 100 ≠ 0) or (año(fecha) módulo 400 = 0))
* misma\_fecha(fecha1, fecha2) -> (dia(fecha1) = dia(fecha2)) and (mes(fecha1) = mes(fecha2)) and (año(fecha1) = año(fecha2))
* Exporta: Fecha, crear\_fecha, dia, mes, anio, es\_bisiesto, misma\_fecha