Algoritmos sobre lostas simplemente enlazadas Nodo<T> es Struct<> T valor;
Nodo siguiente;

```
modulo ListaEnlazadaSimple<T> implementa Secuencia<T>{
      var primero: NodoLista<T>
      var ultimo: NodoLista<T>
      var longitud: int
      pred InvRep (args){
          cuerpo
      }
      pred Abs (args){
          cuerpo
      }
      impl pertenece(in lista: ListaEnlazadaSimple<T>,in buscado: T):Bool{
            Nodo actual=primero;
            int indice=0;
            if longitud==0 then return false
            while longitud! = indice + 1&& actual.valor ! = buscado do
               actual= actual.siguiente()
               i++
            end while
            return actual.value==T;
            acá medio que simplemente miraron que el nodo no sea null en vez de usar el indice, que bueno,
            no me convencio mucho asi que prefiero mi implementacion
        }
      impl obtener(in lista: ListaEnlazadaSimple<T>,in buscado: T):T{
            Nodo actual=primero
            while actual.valor! = T do
               actual.siguiente
            end while
            return actual.valor
        }
      impl concat(inout s1: ListaEnlazadaSimple<T>,in s2: ListaEnlazadaSimple<T>):{
            s1.ultimo.siguiente=s2.primero; s1.ultimo=s2.ultimo; s1.longitud+=s2.lingitud;
            acá hay que tener cuidado con el aliasing, en realidad lo idea es copiar toda l2 y concatenarle
            eso, para no correr peligro de que alguien despues modifique l2 y me cambie cosas que no quiere.
            Acá, como estamos en una materia de la facu, podemos decir que alguien imaginario se encargo
            de que lo que nos estan pasando ya es una copia.
        }
}
```

Recursion Vamos a hacer cosas simples sobre recorrer estrucutras, nada loco (como si se hacia en otras insta
de esta materia).
Estructura basica:

- Caso base
- Paso recursivo

Ejercicio 2. Calculo del factorial de un entero positivo de forma recursiva.

```
impl factorial(in n: N):N{
      if n!=1 then
         return n*factorial(n-1)
      else
         return 1
      end if
   }
```

Arboles binarios **modulo** NodoAB<T> **implementa** Nada{

```
      var val: T
      var izquierda: NodoAB<T>
      var derecha: NodoAB<T>
}
modulo ArbolBinario<T> implementa ArbolBinario<T>{
      var raiz: NodoAB<T>
      pred InvRep (args){

      }
      pred Abs (args){
          no lo hacemos :p
      }
      impl esVacio(in ab: ArbolBinario<T>):Bool{
            return ab.raiz==null;
         }
}
```

Como recorremos un arbol binario? Recursivamente!

```
impl cantidadDeNodos(in ab):nat{
      if esVacio(ab) then
         return 0
      else
         return 1+cantidadDeNodos(rama izquierda)+cantidadDeNodos(rama derecha)
         Acá no matchean bien los tipos, porque la funcion recibe un arbol binario pero le estamos pasando
         un nodo y bla bla, pero se entiende el concepto
      end if
   }
```

Ejercicio 3.

- altura(in ab: ArbolBinario<T>):int
- está(in ab: ArbolBinario<T>):bool

```
impl altura(in ab: ArbolBinario<T>):int{
      if esVacio(ab) then
         return 1
      else
         if cantidadDeNodos(ab.izquierda)>=cantidadDeNodos(ab.derecha) then
            return 1+altura(ab.izquierda)
         else
            return 1+altura(ab.derecha)
         end if
      end if
   }
impl altura(in ab: ArbolBinario<T>):int{
      if esVacio(ab) then
         return 1
      else
         int cd = altura(ab.derecha)
         int ci = altura(ab.izquierda)
         if cd>=ci then
            return 1+altura(ab.derecha)
         else
            return 1+altura(ab.derecha)
         end if
      end if
   }
```

```
impl esta(in ab: ArbolBinario<T>, int: T):bool{
      if esVacio(ab) then
         return false
      else
         if ab.valor=T then
            return true
         else
            esta(ab.izquierda)||esta(ab.derecha)
         end if
      end if
   }
```

Resoluciones comunitarias (?)

```
impl altura(in ab: ArbolBinario<T>):int{
      int res;
      if estaVacio(ab) then
         return 0
      else
         res=1+max(ab.izquierda,ab.derecha)
      end if
   }
```

esto es $O(n)$ (no entiendo bien por que)

```
impl pertenece(arbol y elemento):bool{
      if esVacio(arbol) then
         return false
      else
         if arbol.raiz.valor==elemento then
            return true
         else
            return esta(arbol.raiz.izquierda)||esta(arbol.raiz.derecha)
         end if
      end if
   }
```

Acá si estan usando bien la forma de acceder a los lugares, yo lo hice un poco a lo yolo, pero se entiende el punto me parece

Por si nos quedamos con ganas de mas...

Rosetree

Árbol con cantidad arbitraria de hojas por nodo.
Lista simplemente enlazada
NodoRosetree<T> es struct<val: T, hijos: Array<NodoRosetree> >

```
modulo Rosetree implementa Nada{
      var raiz: NodoRosetree<T>
      ...
      pred InvRep (){
          cuerpo
      }
      pred Abs (args){
          cuerpo
      }
}
```

Implementar

- altura(in arbol: Rosetree<T>):int
- está(in arbol: Rosetree<T>, in t: T): bool

NodoRosetree<T> es struct<val: T, hijos: Array<NodoRosetree> >

```
modulo Rosetree implementa Nada{
      var raiz: NodoRosetree<T>
      ...
      pred InvRep (){
```

```
        todos los nodos tienen un unico padre
    }
    impl altura(arbol):int{

       }
}
```