Implementación del TAD árbol n-ario

La implementación de un árbol n-ario se hará de forma dinámica mediante punteros, usando una representación denominada *"primogénito – siguiente hermano"* o *"hijo izquierdo - hermano derecho"*.

Consiste en crear, para cada nodo, una lista dinámica con sus hijos. Así, desde un nodo se accede al nodo que contiene el primer hijo (el que está situado más a la izquierda), y al nodo que contiene al hermano derecho.

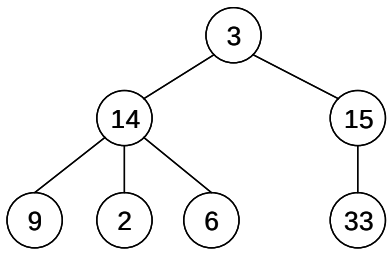
Por tanto, el tipo *árbol* es un puntero a un registro formado por un campo con la información correspondiente al elemento raíz, un campo puntero apuntando al registro correspondiente al primer hijo (con valor nulo en caso de ser una hoja), y otro campo puntero apuntando al registro correspondiente al siguiente hermano (con valor nulo en caso de no tener más hermanos).

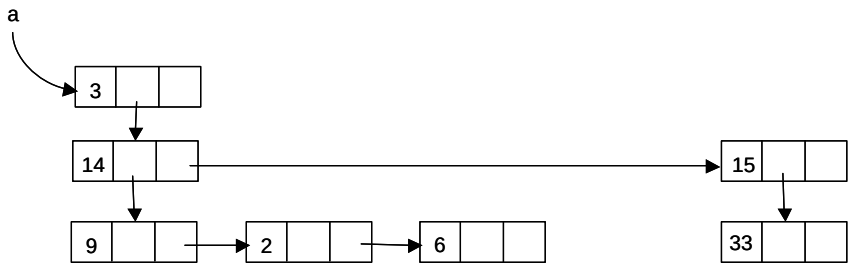
Los tipos que vamos a utilizar se definen de la siguiente forma:

```
tipos   árbol = puntero a nodo

        nodo = registro
                 dato: elemento
                 primogénito, sigHermano: árbol
               fregistro

        bosque = árbol
```

Ejemplo

Veamos cómo se representa en memoria el siguiente árbol n-ario:





La lista enlazada que se forma con los punteros que apuntan al siguiente hermano representa un bosque ordenado.

El módulo para el TAD *árboles ordenados* queda de la siguiente forma:

```
módulo árbolesOrdenados
importa defTipoElemento
exporta
    tipos árbol, bosque

    acción creaVacío (var b: bosque)
    acción añadeDch (var b: bosque; a: árbol)
    función long (b: bosque): 0..maxEntero
    acción observa (b: bosque; i: 1..maxEntero; var a: árbol)
    función altBosque (b: bosque): 0..maxEntero
    acción resto (b: bosque; var rb: bosque)
    acción plantar (e: elemento; b: bosque; var a: árbol)
    función raíz (a: árbol): elemento
    acción bosq (a: árbol; var b: bosque)
    acción subárbol (a: árbol; i: 1..maxEntero; var sa: árbol)
    función numHijos (a: árbol): 0..maxEntero
    función esHoja (a: árbol): booleano
    función altArbol (a: árbol): 0..maxEntero
    acción asignaBosque (var nuevo: bosque; viejo: bosque)
    acción liberaBosque (var b: bosque)
    acción asignaArbol (var nuevo: árbol; viejo: árbol)
    acción liberaArbol (var b: árbol)


implementación

        tipos   árbol = puntero a nodo

                nodo = registro
                          dato: elemento
                          primogénito, sigHermano: árbol
                       fregistro

                bosque = árbol


    acción creaVacío (var b: bosque)
        b:= nulo
    facción
```

```
acción añadeDch (var b: bosque; a: árbol)
 var
    aux: bosque;
 fvar

    si b = nulo entonces
       b:= a;
    sino
       aux:= b;
       mientras aux^.sigHermano ≠ nulo hacer
          aux:= aux^.sigHermano
       fmientras
       aux^.sigHermano:= a
    fsi
 facción


 función long (b: bosque): 0..maxEntero
    si b = nulo entonces
       retorna (0)
    sino
       retorna (1 + long (b^.sigHermano))
    fsi
 ffunción


 acción observa (b: bosque; i: 1..maxEntero; var a: árbol)
    si i = 1 entonces
       a:= b
    sino
       observa (b^.sigHermano, i-1, a)
    fsi
 facción


 función altBosque (b: bosque): 0..maxEntero
    si b = nulo entonces
       retorna (0)
    sino
       retorna (max (altArbol (b), altBosque (b^.sigHermano)))
    fsi
 ffunción


 acción resto (b: bosque; var rb: bosque)
    rb:= b^.sigHermano
 facción


 acción plantar (e: elemento; b: bosque; var a: árbol)
    reservar (a);
    a^.dato:= e;
    a^.primogénito:= b;
    a^.sigHermano:= nulo;
 facción
```

```
función raíz (a: árbol): elemento
    retorna (a^.dato)
ffunción


acción bosq (a: árbol; var b: bosque)
    b:= a^.primogénito
facción


acción subárbol (a: árbol; i: 1..maxEntero; var sa: árbol)
var
    b: bosque
fvar

    bosq (a, b);
    observa (b, i, sa);
facción


función numHijos (a: árbol): 0..maxEntero
var
    b: bosque
fvar

    bosq (a, b);
    retorna (long (b));
ffunción


función esHoja (a: árbol): booleano
    retorna (a^.primogénito = nulo)
ffunción


función altArbol (a: árbol): 0..maxEntero
var
    b: bosque
fvar

    si esHoja (a) entonces
        retorna (1)
    sino
        bosq (a, b);
        retorna (1 + altBosque (b))
    fsi
ffunción
```

```
acción asignaBosque (var nuevo: bosque; viejo: bosque)
var
    primerArbol: árbol
fvar

    si viejo = nulo entonces
        nuevo:= nulo
    sino
        observa (viejo, 1, primerArbol);
        asignaArbol (nuevo, primerArbol);
        asignaBosque (nuevo^.sigHermano, viejo^.sigHermano)
    fsi
facción


acción liberaBosque (var b: bosque)
    si b ≠ nulo entonces
        liberaBosque (b^.sigHermano);
        liberaArbol (b)
    fsi
facción


acción asignaArbol (var nuevo: árbol; viejo: árbol)
var
    viejoBosque, nuevoBosque: bosque;
fvar

    bosq (viejo, viejoBosque);
    asignaBosque (nuevoBosque, viejoBosque);
    plantar (raíz (viejo), nuevoBosque, nuevo);
facción


acción liberaArbol (var b: árbol)
    liberaBosque (a^.primogénito);
    liberar (a)
facción

fmódulo
```