UNIVERSIDADE DA CORUÑA

FACULTADE DE INFORMÁTICA
Departamento de Computación
Programación II

Otros TADs relacionados con listas:
Multilistas y listas multiordenadas

Dadas las siguientes representaciones gráficas se propone como ejercicio hacer la declaración de los tipos de las units implicadas:

1. Este ejemplo se trata de una *multilista* (lista compuesta de listas) que utilizaremos para mantener la lista de los *usuarios* de un sistema informático—siendo los datos de interés login y password encriptada. Para cada uno de ellos, además, guardaremos la lista de los *trabajos* que tiene en ese momento en ejecución. En este caso, para cada trabajo almacenaremos su identificador y la carga máxima de CPU que supone. La estructura de datos se corresponde con la mostrada en la Figura 1.

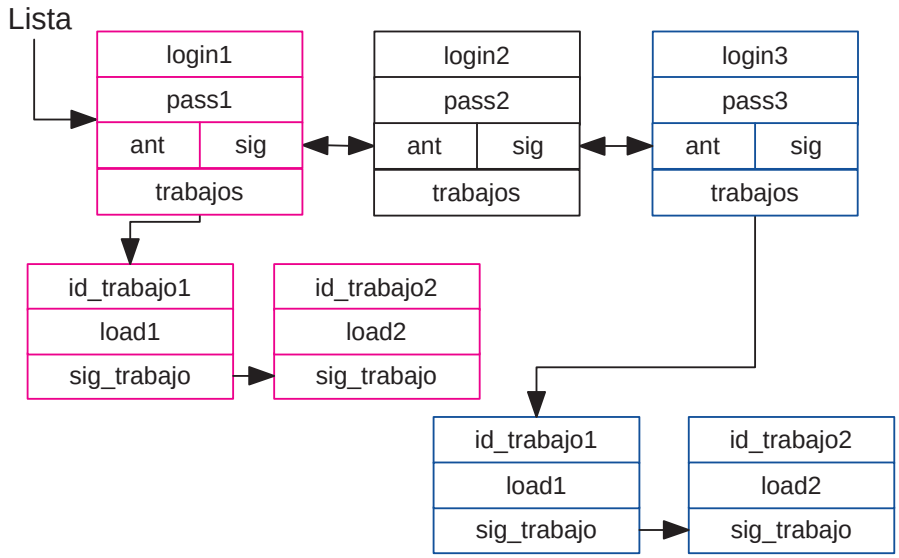

Figura 1: Multilista de usuarios y trabajos

2. En este ejercicio manejaremos *listas multiordenadas*. En estas listas:

   a) Los elementos se encuentran ordenados por distintos criterios

   b) Los nodos tendrán tantos campos de enlace como claves de ordenación.

   c) Cada campo de enlace mantiene la lista ordenada por uno de los criterios.

   d) Existe un nodo de cabecera que será un registro con tantos campos como claves de ordenación existan.

   En este caso, nos interesa mantener una lista de personas ordenadas por apellido y también por DNI, tal como se refleja en la Figura 2. El nodo de cabecera sería un registro con dos punteros, uno al principio de la lista de registros ordenados por nombres y otro a la lista de los mismos registros pero ordenados por edad.

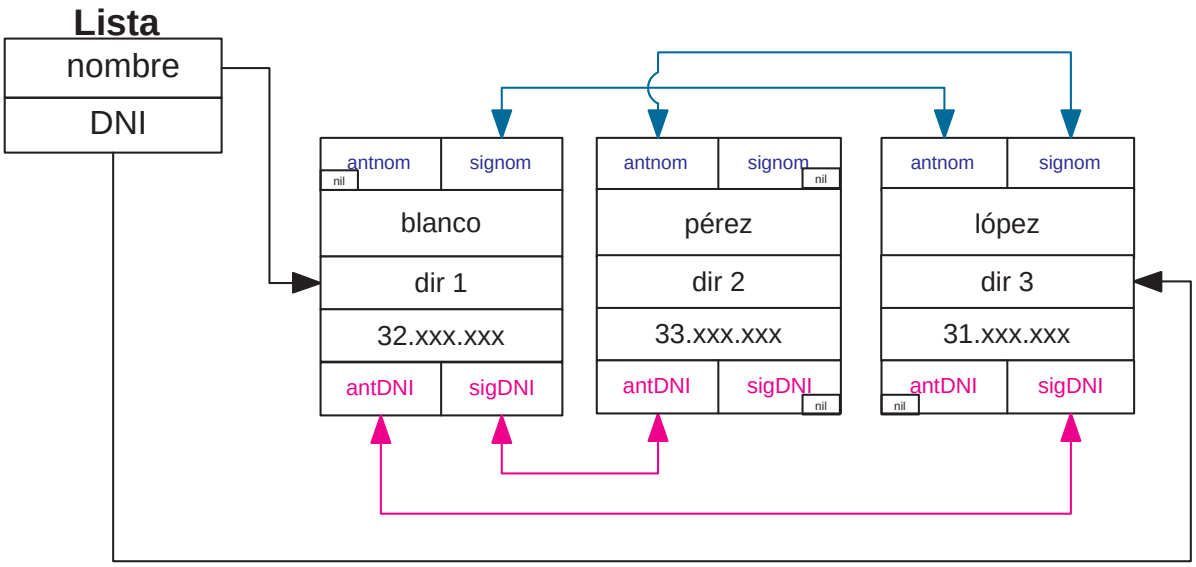

Figura 2: Lista de personas ordenada por dos criterios: apellido y DNI

3. En este ejemplo manejaremos una estructura más compleja. Se trata de un ambulatorio en el que se desea asociar asegurados con médicos, de manera que podamos acceder a la lista de médicos del seguro, a la lista de pacientes del seguro y, además, a la lista de pacientes que son atendidos por un determinado médico. La estructura se refleja en la Figura 3.

Como se puede observar, se mantienen dos listas doblemente enlazadas: una para los asegurados y otra para los médicos. Para facilitar la búsqueda de médicos y pacientes ambas listas están ordenadas: la lista de médicos está ordenada alfabéticamente por apellido y la de pacientes por DNI. Además, para poder saber qué pacientes atiende qué médico, cada nodo médico tiene un puntero al nodo del primer asegurado asociado, y cada nodo de asegurado tiene un enlace al siguiente asegurado al que le atiende el mismo médico.

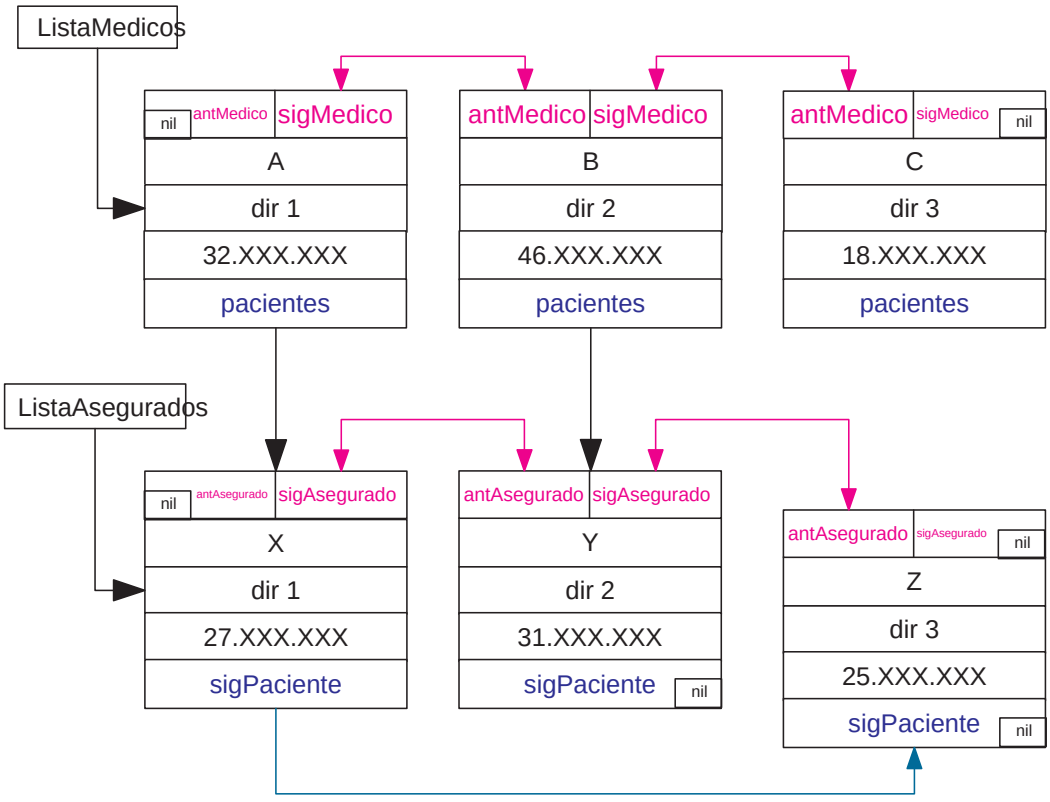

Figura 3: En esta figura el médico A atiende a los pacientes X y Z, el médico B atiende al paciente Y, y el médico C no tiene ningún paciente.

Soluciones

1. Ejercicio 1: lista de usuarios y sus trabajos

La definición de tipos sería la siguiente. En primer lugar, crearíamos el TAD necesario para mantener la lista de trabajos:

```
Unit ListaTrabajos;
interface
    const
        nulo=nil;
    type
        tId=integer;
        tLoad=real;
        tPosTrabajo=^tNodoTrabajo;
        tDatoTrabajo=record
            id: tId;
            load: tLoad;
        end;
        tNodoTrabajo=record
            dato: tDatoTrabajo;
            sig: tPosTrabajo;
        end;
        tListaTrabajos=tPosTrabajo;
    {operaciones}
    procedure ListaTrabajosVacia(var L: tListaTrabajos);
    ...
implementation
    ...
end.
```

```
Unit ListaUsuarios;
uses ListaTrabajos;
interface
    const
        nulo=nil;
    type
        tLogin=string [10];
        tPass=string[10];
        tDatoUsuario=record
```

```
            login: tLogin;
            pass: tPass;
            trabajos: tListaTrabajos;
        end;
        tPosUsuario=^tNodoUsuario;
        tNodoUsuario=record
            dato: tDatoUsuario;
            ant, sig: tPosUsuario;
        end;
        tListaUsuarios=tPosUsuario;
    {operaciones}
    procedure ListaUsuariosVacia(var L: tListaUsurios);
    ...
implementation
    ...
end.
```

2. Ejercicio 2: Lista Multiordenada

```
Unit ListaMultiordenada;
interface
const
   nulo=nil;
type
   tNombre=string [25];
   tDir=string[40];
   tDNI=integer;
   tDato=record
       Nombre: tNombre;
       direccion: tDir;
       DNI: tDNI;
   end;
   tPos=^tNodo;
   tNodo=record
       dato: tDato;
       antnombre, signombre: tPos;
       antDNI, sigDNI: tPos;
   end;
   tLista=record
       nombre: tPos;
       DNI: tPos;
   end;
   ...
   implementation
   ...
```

3. Ejercicio 3: información del ambulatorio.

Este ejemplo está tomado del Joyanes. En el libro aparecen tanto las dos unit desarrolladas como las funciones para hacer consultas a las listas y dar altas. La definición de tipos sería:

```
Unit ListaAsegurados;
interface
    const
        nulo=nil;
    type
        tNombre=string [25];
        tDir=string[40];
        tDNI=integer;
        tDatoAsegurado=record
            Nombre: tNombre;
            direccion: tDir;
            DNI: tDNI;
        end;
        tPosAsegurado=^tNodoAsegurado;
        tNodoAsegurado=record
            dato: tDatoAsegurado;
            antAsegurado, sigAsegurado: tPosAsegurado;
            sigPaciente: tPosAsegurado;
        end;
        tListaAsegurados=tPosAsegurado;
        ...
    end.

Unit ListaMedicos;
interface
    uses ListaAsegurados;
    type
        tDatoMedico=tDatoAsegurado;
        tPosMedico=^tNodoMedico;
        tNodoMedico=record
            dato: tDatoMedico;
            antMedico, sigMedico: tPosMedico;
            pacientes: tPosAsegurado;
        end;
        ...
    end.
```