

# Listas doblemente enlazadas, listas circulares, multilistas

## Introducción

A veces es conveniente efectuar algunas modificaciones en la implementación de listas para facilitar ciertas operaciones sobre ellas. Además, podemos tener algunas cuyos elementos posean, a su vez, otras listas enlazadas. En este módulo se verán algunas de esas variantes.

### *Listas doblemente enlazadas*

Si es necesario recorrer una lista en sentido directo y, luego, en sentido inverso es conveniente que todo nodo tenga un puntero al anterior, además de tener un puntero al siguiente. Dicha implementación de listas se conoce como **listas doblemente enlazadas**.

```
/* Estructura de los nodos*/
typedef struct nodo{
    int clave;
    struct nodo *sig;
    struct nodo *ant;
} tNodo;
/* Definición del tipo lista
 * como un puntero a un nodo
 */
typedef tNodo * tLista;
```

### *Listas circulares simplemente enlazadas*

Se caracterizan porque el último elemento apunta al primero. Dicha implementación de listas se conoce como **listas doblemente enlazadas**.

### *Listas circulares doblemente enlazadas*

Permiten un recorrido en sentido horario y antihorario de toda la lista. La estructura es la de una lista doblemente enlazada, pero implementada de tal manera que, al llegar al

último nodo, se continua con el primero y, si se llega al primer nodo, se puede pasar al último.

## Multilistas

Una lista puede tener, como elemento, otra lista. Por ejemplo: se lee un archivo de texto (solo palabras) y se quiere clasificar las palabras por su longitud, para luego efectuar ciertas estadísticas. Como máximo, una palabra puede tener 30 letras. Podríamos hacer un arreglo de 30 listas de palabras, pero desperdiciaríamos memoria. Entonces, es conveniente hacer una lista ordenada por tamaño de las palabras y que cada elemento de esa lista tenga las palabras de ese tamaño.


Al liberar memoria de la lista habrá que recorrer cada nodo de la lista principal y liberar cada lista que se encuentra ahí. En el ejemplo anterior, por cada tamaño libero su lista de palabras y luego libero el nodo de ese tamaño.

### *Estructura de una multilista*

Supongamos que queremos guardar los vuelos de distintas aerolíneas. En la lista principal guardaremos el nombre de la aerolínea y, para cada aerolínea, se almacenará el destino del vuelo, la hora de partida y la hora de llegada. Definimos dos estructuras de la siguiente manera:

```
typedef struct aerolínea
{
    char nombreA[40];
    struct aerolínea *sig;
    struct vuelos *v;
}nodoA;

typedef struct vuelos
{
    char destino[20];
    char horA[8];
    char horP[8];
    struct vuelos *sigV;
}nodoV;
```



Se define un nuevo puntero a la estructura vuelos.