Laboratorio de PRED – Práctica 1

Se desea implementar en C++ una lista de cuyos nodos cuelgan listas circulares. Es decir los nodos de la lista tienen la siguiente declaración:

```
class nodo1
{friend class nodo2;
public:
string  x1;
nodo2 * cabezalc; /* puntero a la lista circular que cuelga de
este nodo*/
nodo1 * sig1;
} ;

class nodo2
{ public:
 int x2;
nodo2* sig2;
} ;
```

Además, se desea que, tanto la lista general como las listas circulares que cuelgan de cada nodo de la lista general, estén ordenadas en sentido ascendente, aunque los punteros cabezalc no tienen por qué apuntar al menor elemento de cada lista circular.
Las operaciones para trabajar con esta estructura de datos han de ser las siguientes:

- **insertar**. Dados dos valores V1 y V2, si el valor V1 ya existe en un nodo N1 de la lista general entonces añade un nodo con el valor V2 a la lista circular que cuelga de N1 y se deja que N1.cabezalc apunte al nuevo nodo. Si V1 no está en la lista general, entonces se añade un nuevo nodo a la lista general con el valor V1 y se hace que cuelgue de ese nodo una lista circular con un único nodo con el valor V2.

- **suprimir**. Dados los valores V1 y m, se elimina de la lista general el nodo cuyo valor es V1, y la lista circular que cuelga de él. Además, esta operación deberá devolver una lista de nodo2 que contenga los nodos de la lista circular eliminada en el siguiente orden de borrado: Primero se borra el nodo que está en la posición m-ésima, después se van eliminando los nodos que estén en las sucesivas posiciones m-ésimas a partir del siguiente a cada nodo borrado, hasta la eliminación completa de la lista circular.

  Por ejemplo, si originalmente la lista circular a eliminar poseía 4 elementos <1,2,3,4>, el apuntado fuese el nodo 3, y m fuese 2, la operación devolvería la lista <4,2,1,3> como resultado del proceso de borrado que representa la siguiente secuencia de estados de la lista circular con el nodo que hay que eliminar subrayado:

  <1, 2, 3, **4**> → <1,**2**, 3> → <**1**, 3> → <**3**> → < >

  Si V1 no está en la lista general entonces no hay que hacer nada y devolver la lista vacía.
- **estan**. Dados dos valores V1 y V2, la función nos devuelve cierto si hay un nodo con valor V1 en la lista general y de él cuelga una lista circular que incluye el valor V2. En caso contrario, la función devuelve falso.

- **menor**. Dado un valor V1, si V1 está en el nodo N1 de la lista principal entonces nos devuelve el menor valor de la lista circular que cuelga de N1. Si V1 no está en la lista principal entonces nos devuelve 9999.

Específicamente, la estructura de datos pedida se implementará por medio de la clase:

```
class ListaPares
{
public:
      nodo1 * primero;
      void insertar(string V1, int V2)
      {
      //código de insertar

      }
      Nodo2* suprimir(string V1, int m)
      {
      //código de suprimir
      }
       bool estan(string V1, int V2)
      {
      //código de estan
      }

       int menor(string V1)
      {
      // código de menor
      }
      };
```

Las clases mencionadas anteriormente han de estar incluidas en un único fichero de nombre ListaPares.cpp que estará encabezado por un comentario que contenga los nombres y números de DNI del autor o los autores de la práctica (las prácticas se hacen individualmente o en grupos de dos personas) y contendrá también los comentarios que se considere oportuno incluir para documentar la práctica (como mínimo pre y postcondiciones para las funciones e invariantes para los bucles). Además, tanto dichas clases, como las operaciones y los campos mencionados más arriba han de tener exactamente los nombres que tienen en este enunciado. En la misma línea, las operaciones descritas no han de realizar ninguna escritura. Por último, desde este fichero no se debe de importar ninguna clase que no sea standard de C++. El incumplimiento de estas especificaciones será penalizado.