Ejercicio 1

Para el día del tercer parcial de algoritmos se decidió agrupar a los alumnos en 6 aulas. Cada aula tendrá un profesor a cargo.

Se dispone de un lista simplemente enlazada de alumnos con los siguientes datos:

- legajo
- apellido_nombre
- comision
- notas: Arreglo [1..5] enteros

Además se cuenta con dos funciones:

1) EsAlumnoLibre(), que recibe como parámetro el legajo del alumno, y devuelve Verdadero o Falso de acuerdo a si el alumno se encuentra libre o no.
2) AsignarAula(), devuelve un número aleatorio entre 1 y 6.

Se solicita:
- Generar un lista doblemente enlazada, de los alumnos en condiciones de rendir (es decir, que no se encuentren libres) con la siguiente información:
  * cursoNro
  * cantidad
  * legajo

Donde se deberá cargar un nodo por cada curso, y de forma intermedia, un nodo por cada alumno asignado a dicho curso.
Los nodos de los cursos serán los nodos cabeceras, los nodos intermedios contendrán información del alumno, de tal forma que:

Si es un nodo cabecera,
el campo cursoNro será el número del curso (1..6),
el campo cantidad indicará la cantidad de alumnos asignados a dicho curso (SE DEBERÁ ACTUALIZAR CADA VEZ QUE SE ASIGNE UN NUEVO ALUMNO).
el campo legajo será 0.

Si es un nodo intermedio:
el campo cursoNro será el número del curso (1..6),
el campo cantidad será 1.
el campo legajo será el legajo del alumno correspondiente.

En el caso de que el alumno se encuentre en condiciones de rendir, eliminarlo de la lista original.

Informar además:
- Promedio de alumnos por aula
- Aula con mayor cantidad de alumnos

Ejercicio 2

Se dispone de un lista simplemente enlazada de estudiantes con los siguientes datos:

- legajo
- apellido_nombre
- comision
- notas: Arreglo [1..5] enteros

Se desea generar una lista simplemente enlazada ordenada por nro de legajo, considerando solo aquellos que hayan obtenido una calificación mayor a 6 en al menos 3 instancias de exámenes. Para indicar si han aprobado al menos 3 exámenes, utilizar una función recursiva. Finalmente, mostrar los estudiantes que se cargaron en la nueva lista.

ACCION ej2(PRIM: puntero a nodo) es
AMBIENTE

```
función ContarNotasMayoresSeis(notas: arreglo [1..5] de enteros, indice: entero, conteo: entero): booleano es
//inicializar la función con índice = 0 y conteo = 0
   SI indice = 1 entonces
      si conteo > 2 entonces
                ContarNotasMayoresSeis := VERDADERO
      sino
           ContarNotasMayoresSeis := FALSO
   SINO
       si notas[indice] > 6 entonces
                  ContarNotasMayoresSeis := ContarNotasMayoresSeis(notas, indice - 1, conteo + 1)
         sino
                  ContarNotasMayoresSeis := ContarNotasMayoresSeis(notas, indice - 1, conteo)
          fin si
fin función

//IMPORTANTE (!) No sirve en recursividad utilizar variables como contadores, ya que en cada llamada se hace una copia de la variable. deberá ser un parámetro

nodo = REGISTRO
      legajo: N(5)
      apellido_nombre: AN(100)
      comision: AN(1)
      notas: Arreglo [1..5] enteros
      prox: puntero a nodo
finreg

p, prim2, q, t, a: puntero a nodo
```

PROCESO
p:= prim

MIENTRAS p <> nill HACER *//recorrer y tratar cada nodo de la lista de entrada*

SI ContarNotasMayoresSeis(*p.notas, 0, 0) ENTONCES

//generar un nodo para la nueva lista que usa prim2

```
            a:= nil
            NUEVO (t)
            //cargo datos
            *t.legajo:= *p.legajo
            *t.apellido_nombre:= *p.apellido_nombre
            *t.notas := *p.notas
            //carga ordenada por legajo:
            q:= prim2
            MIENTRAS (q <> nil) y (*q.legajo < *t.legajo) HACER
                a:= q
                q:= *q.prox
            FIN MIENTRAS

            SI (a = nil) ENTONCES
                *t.prox:= nil
                Prim2:= t
            SINO
                *t.prox:= q
                *a.prox:= t
            FIN SI

      finsi

      p := *p.prox

finmientras
```