Facultad de Estudios Estadísticos. Grado en Estadística Aplicada. Programación II
Curso 2012-2013. Examen parcial. 27 de mayo de 2013.

Puedes definir los subprogramas adicionales que consideres necesarios.
Todos los subprogramas que se utilicen se deben definir.
NO se debe escribir la función `main()` si no se pide explícitamente.

Ejercicio 1 (1,5 puntos) Escribe un subprograma que simule el lanzamiento de un dado cargado de 6 caras, en el que la cara que contiene el "6" tiene el doble de probabilidad de salir que las demás. El subprograma debe devolver un número entero con el valor de la cara que salga (de 1 a 6).

Escribe un programa (incluyendo la función `main()`) que utilice el subprograma anterior para mostrar en la pantalla la probabilidad aproximada de que salga cada una de las caras del dado realizando N experimentos.

Ejercicio 2 (3 puntos) La red de concesionarios de una marca de automóviles va a realizar un programa de gestión de la red de concesionarios. Para ello debe tratar los siguientes datos:

- De cada modelo de automóvil se debe considerar la siguiente información (para esta información se utilizará un tipo de datos con nombre **`tModelo`**):
  - Código de modelo de automóvil, un número entero positivo.
  - Denominación del modelo (por ejemplo, "Volvo V40 Cross Country").
  - Número de automóviles en *stock* de ese modelo **en cada uno de los concesionarios**.
  - Número de automóviles vendidos el último mes **en cada uno de los concesionarios**.
- De la red de concesionarios se debe considerar la siguiente información (para esta información se utilizará un tipo de datos con nombre **`tRedConcesionarios`**):
  - El número de concesionarios de la red (como máximo K).
  - Una lista con los nombres de los concesionarios de la red (por ejemplo, "Automóviles Reto Car, S.L.").
  - El número de modelos de automóvil (como máximo N).
  - Una lista con los datos de cada modelo de automóvil, cada uno de tipo **`tModelo`**.

El programa va a mantener en memoria toda esta información de la red de concesionarios. Contesta a las siguientes preguntas:

a) Diseña los tipos de datos necesarios.

b) Escribe un subprograma `void guardarFic(tRedConcesionarios red)` que guarde todos los datos de la red de concesionarios en un fichero de texto.

c) Escribe un subprograma `void leerFic(tRedConcesionarios& red)` que lea los datos de la red de concesionarios de un fichero de texto escrito con el subprograma del apartado anterior.

Ejercicio 3 (5,5 puntos) Se quiere modificar la práctica de la agenda de tareas y reuniones con algunos cambios.

a) Por una parte, se quieren saber las horas de la semana en las que no hay ninguna tarea planificada. Escribe un subprograma con el siguiente prototipo:

```
void horasDisponibles(tAgenda ag1);
```

que muestre la lista de todas las horas de la semana que están libres de tareas. Por ejemplo, Si la agenda tiene la siguiente información:

```
LISTA DE TAREAS:
-------------------------------------------------------
DP1  : *** 09:00-14:00 - Direccion de proyectos.
SGWEB: Lun 16:00-18:00 - Seguimiento servicio Web.
ADM1 : Mie 16:00-18:00 - Administracion.
PERS1: Jue 08:00-09:00 - Reconocimiento medico.
MKTNG: Jue 16:00-19:00 - Planificacion Marketing.
BBDD : Jue 17:00-18:00 - Seguimiento BBDD.
ADM2 : Vie 17:00-19:00 - Administracion.
```

el subprograma `horasDisponibles()` debe mostrar el siguiente resultado:

```
LISTA DE HORAS DISPONIBLES:
-------------------------------------------------------
LUNES    : 08:00  14:00  15:00  18:00
MARTES   : 08:00  14:00  15:00  16:00  17:00  18:00
MIERCOLES: 08:00  14:00  15:00  18:00
JUEVES   : 14:00  15:00
VIERNES  : 08:00  14:00  15:00  16:00
```

(Se recomienda utilizar una matriz en memoria similar a la usada en el punto 5 de la práctica. Las tareas pueden no estar ordenadas).

b) También se quiere añadir a la agenda la información siguiente: dada una tarea T, se desea saber **las tareas de las que depende.** Es decir, las tareas que deben haberse terminado para que T pueda realizarse.

Por ejemplo, en la lista de tareas del punto anterior la tarea `BBDD` debe realizarse después de que terminen `SGWEB` y `ADM1`. En este caso, la tarea `BBDD` **depende de** `SGWEB` y `ADM1`.

Para ello, en la estructura de datos **`tTarea`** que contiene los datos de una tarea se debe añadir la lista de identificadores de tareas que deben haber terminado. Se deben poder guardar como máximo D identificadores por tarea, pero cada tarea puede tener un número distinto de tareas de las que depende.

b.1) Escribe las estructuras de datos que has utilizado para representar la agenda en tu práctica, con la nueva información que necesitas para esta modificación.

b.2) Escribe un subprograma con el siguiente prototipo:

```
void mostrarDependenciasIncorrectas(tAgenda ag);
```

que muestre en la pantalla las tareas de la agenda `ag` que empiezan antes de que terminen las tareas de las que dependen. Deben indicarse los datos que se muestran en el siguiente ejemplo:

Si la tarea `MKTNG` depende de `BBDD`, `ADM1`, `ADM2` y `SGWEB`, debe mostrar una lista con la siguiente información sobre estas tareas:

```
LISTA DE DEPENDENCIAS INCORRECTAS:
-------------------------------------------------------
MKTNG: Jue 16:00-19:00 - Planificacion Marketing.
       Empieza antes de: BBDD : Jue 17:00-18:00 - Seguimiento BBDD.
                         ADM2 : Vie 17:00-19:00 - Administracion.
...
```