# Programación 2

# Examen de teoría (julio 2017)

# 12 de julio de 2017

## Instrucciones

- **Duración: 3 horas**
- El fichero del primer problema debe llamarse `solapes.cc`. Para el segundo problema es necesario entregar cuatro ficheros, llamados `Startup.cc, Startup.h, Project.cc, Project.h`. Pon tu DNI y tu nombre en un comentario al principio de los ficheros fuente.
- La entrega se realizará como en las prácticas, a través del servidor del DLSI (`http://pracdlsi.dlsi.ua.es`), en el enlace `Programación 2`. Puedes realizar varias entregas, aunque sólo se corregirá la última.
- En la página web de la asignatura `http://www.dlsi.ua.es/asignaturas/p2` tienes disponibles algunos ficheros que te pueden servir de ayuda para resolver los problemas del examen, y el apartado `Reference` de la web `www.cplusplus.com`.

## Problemas

1. **(5.5 puntos)**

   En este ejercicio deberás hacer un programa llamado `solapes` que permita detectar solapamientos en los horarios de los alumnos de una universidad. Un solapamiento se produce cuando un alumno tiene asignadas dos o más clases en el mismo horario.

   Para realizar esta tarea disponemos de un fichero binario "`asignaturas.bin`" en el que hay registros con la información de los horarios de los grupos de teoría y prácticas de las asignaturas. Para cada grupo se almacenan los siguientes datos:

   ```
   int cuatrimestre;  // 1 o 2
   char codAsignatura[6]; // p.ej. "34008"
   int franjaInicio;
   int franjaFin;
   ```

   Las franjas indican periodos de 30 minutos. Cada día hay 24 franjas (de 9 a 9:30 hasta de 20:30 a 21:00) y hay un total de 120 franjas a la semana, es decir, serán siempre un número entero entre 1 (lunes 9:00-9:30) y 120 (viernes 20:30-21:00). Se debe asumir que los valores almacenados en el fichero binario serán siempre correctos y que siempre `franjaInicio` será menor que `franjaFin`.

   Por otra parte, en un fichero de texto llamado "`horario.txt`" se indicará para cada alumno a qué grupos asiste, con el siguiente formato:

   ```
   Jesus Tomellevao:27|2|15|48|57|3|21
   Armando Jaleo:1|18|2|32|9
   Aitor Tilla:14|23|43|38|7
   Maria Unaspatatas:17|16|20|0|34
   ```

   En cada línea aparece primero el nombre del alumno, seguido de "`:`" y una secuencia de números de grupo que se corresponden con registros del fichero binario (el primer registro es el registro 0 y en él estarán alamacenados los datos del grupo 0).

   El programa debe leer línea por línea el fichero de texto y comprobar si para cada alumno hay algún solapamiento entre sus grupos; hay solapamiento entre dos grupos cuando están en el mismo cuatrimestre y hay al menos una franja horaria común en ambos grupos. Por ejemplo, si el grupo 48 es los lunes de 10 a 11 (`franjaInicio=3, franjaFin=4`), y el 21 los lunes de 10:30 a 12:00 (`franjaInicio=4, franjaFin=6`), y ambos grupos son del mismo cuatrimestre (p.ej. el 2), hay un solapamiento. En ese caso, el programa debe imprimir el nombre del alumno, el número y datos de cada grupo:

```
Solapamiento: Jesus Tomellevao
48 2 34014 1-4
21 2 34004 4-6
```

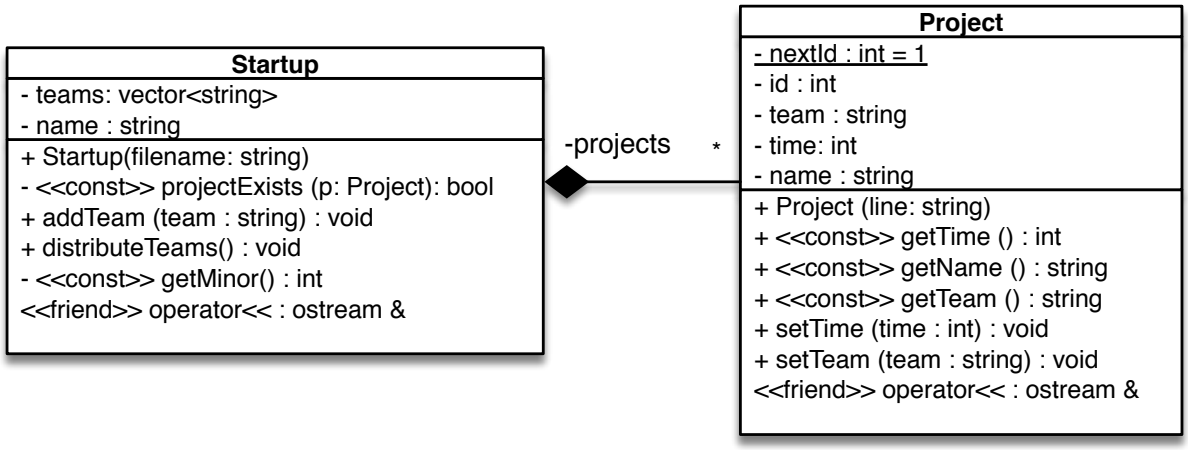
Para cada solapamiento entre dos grupos se emitirá un mensaje como el anterior. Si un alumno no tiene ningún solapamiento entre sus grupos (lo normal, vamos), se debe emitir este mensaje:

```
Armando Jaleo: ok
```

Puedes (y debes) utilizar un vector para almacenar los datos de los grupos de un alumno, pero no debes leer ninguno de los dos ficheros completamente en memoria. Pueden ser más grandes que la memoria disponible para el programa. Tampoco debes leer los ficheros más de una vez.

2. **(4.5 puntos)**

Queremos hacer un programa para gestionar los proyectos de una startup.




Un proyecto (`Project`) tiene un nombre (`name`), un identificador único (`id`), el nombre del equipo de trabajo (`team`) asignado al proyecto, y el tiempo (`time`) en horas que se tarda en terminarlo. El constructor crea un proyecto a partir de un string con el formato `name=time`, por ejemplo: `MusicApp=14`, asignando al atributo `id` el valor de `nextId` (inicialmente 1), e incrementándolo para el siguiente proyecto que se cree. En el constructor se dejará el equipo inicialmente vacío. El operador salida debe imprimir el proyecto como `name (id)= time`. Por ejemplo: `MusicApp (1)=14`.

La clase `Startup` debe asignar los equipos de trabajo (teams) a los distintos proyectos. Para ello, el constructor carga la información de los proyectos a partir de un fichero[1] cuyo nombre recibe por parámetro. Este fichero contendrá el nombre de la startup, y a continuación un proyecto por línea. Si el proyecto ya existe en el vector (debe comprobarse con el método `projectExists`) no hay que añadirlo. El método `addTeam` simplemente añade un equipo al vector de equipos[2]. El método `distributeTeams` se encarga de asignar todos los proyectos a los equipos de trabajo, asignando el proyecto de menor tiempo al primer equipo, después haciendo lo mismo para el segundo equipo, y así para el resto de proyectos. Para ello, debe usar la función `getMinor`, que devuelve la posición (en el vector) del proyecto sin equipo asignado que tenga un tiempo menor. Esta función devolverá `-1` si todos los proyectos tienen ya equipo asignado. Finalmente, el operador salida debe imprimir la información de los proyectos agrupados por equipos como se muestra en el siguiente ejemplo[3]. Dado el siguiente `main.cc`, que puedes compilar con `g++ Project.cc Startup.cc main.cc -o ej2`:

```
#include "Startup.h"
int main() {
  Startup s("data.txt");
  s.addTeam("A");
  s.addTeam("B");
  s.addTeam("C");

  s.distributeTeams();

  cout << s << endl;
}
```

Con `data.txt`, el programa debe mostrar:

```
TheBestCompanyEver
------
A: Music App (1)=14 GTA IX (5)=500
B: Space Shuttle Simulator (2)=522 Eyes App (3)=41
C: Happy Birds (4)=120
```

[1] Hay que dar un error si el fichero no se puede abrir, pero se asume que el formato del fichero será siempre correcto.
[2] No hace falta comprobar que no haya equipos duplicados.
[3] Si un equipo no tiene proyecto asignado se mostrará solamente el nombre del equipo y el caracter `':'`.