PruebaAlien
www.wuolah.com/student/PruebaAlien



# ExamenMPSept2017Practica.pdf

*Examenes Teoria*

**1° Metodología de la Programación**

**Grado en Ingeniería Informática**

**Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación**
**Universidad de Granada**

**Normas para la realización del examen:** **Duración: 1 hora**

- **El único material permitido durante la realización del examen es un bolígrafo azul o negro.**
- **Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.**
- **No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.**

Se pretende informatizar la gestión de reservas para las entradas de grupos de turistas a la Alhambra, que se producen a lo largo del día. Para ello, se construirá la clase **ReservasGrupos**. Para cada grupo, esta clase almacenará los nombres y apellidos de sus componentes (se usará un cstring para representarlos) y la hora de reserva. Los grupos pueden tener un número diferente de turistas y tendrán distintas horas de reserva asignadas.

Así, se propone la siguiente representación para esta clase:

```
struct Hora{
    int hh;
    int mm;
 };

class Turista{
  private:
    char* nombre;      // Cadena-C reservada dinámicamente para nombre y apellido
    char *id;          // Cadena-C reservada dinámicamente para DNI o pasaporte
    int nacionalidad;  // Código de nacionalidad (por ej. código telefónico)
  public:
  // ... interfaz pública de la clase
};

class GrupoTuristas{
  private:
    Turista *turistas;   // Vector dinámico con los turistas del grupo
    int nturistas;       // Número de turistas en el grupo
    Hora hora;           // Hora de reserva del grupo
  public:
  // ... interfaz pública de la clase
};

class ReservasGrupos {
  private:
     GrupoTuristas *grupos;  // Vector dinámico con los grupos de turistas
     int ngrupos;            // Número de grupos con reserva
  public:
  // ... interfaz pública de la clase
};
```

## NOTAS:

1. Los métodos que modifiquen el estado de los objetos de esta clase deben asegurarse que utilizan la memoria dinámica **exacta** necesaria para almacenar cada elemento de información.

2. Para la realización de los ejercicios del examen hay que implementar, de forma explícita, determinados métodos y, también, se pueden usar otros que se asume que ya están implementados (se especifica con claridad cada caso). En cada pregunta, se pueden emplear directamente los métodos considerados en preguntas anteriores, tanto de un tipo como de otro.

3. Por último, el alumno puede implementar los métodos privados auxiliares y métodos adicionales que vea convenientes.

---

## ◁ Ejercicio 1 ▷ Métodos básicos de la clase [ puntos]

Defina los siguientes métodos para la clase **ReservasGrupos**:

1. **Constructor por defecto** y **destructor**.
2. **Constructor de copia** y **operador de asignación**.

Para la realización de estos ejercicios, se puede asumir la existencia (e implementación) de determinados métodos para las clases **Turista** y **GrupoTuristas**. Debe especificar sus cabeceras y justificar brevemente su necesidad.

### ◁ Ejercicio 2 ▷ Sobrecarga de operadores [ puntos]

Implemente la siguiente funcionalidad para la clase **ReservasGrupos** sobrecargando los operadores +, << y >>.

1. **Operador de suma** para poder obtener la combinación de dos objetos de la clase **ReservasGrupos**. El conjunto de reservas resultado de este operador se obtendrá uniendo las reservas representadas en los dos argumentos. Cuando existan dos grupos pertenecientes a cada argumento con la misma hora de reserva, se combinarán en un único grupo. Este operador será adecuado cuando se realicen reservas desde distintos puntos de venta y, al final, se debe confeccionar un conjunto de reservas global.
2. **Operador de salida** << para poder insertar en un flujo de salida el contenido de las reservas de grupos en formato texto. En concreto, deberá aparecer:
   - Un entero que indica el número de grupos de turistas que tienen reserva.
   - Para cada grupo de turistas, se insertará la siguiente información:
     - Un entero que indica el número de turistas en el grupo.
     - Para cada turista del grupo, se insertará, en líneas separadas, la siguiente información: su identificador, su nombre y apellidos y su código de nacionalidad.
     - Un entero que indica la hora de reserva.
     - Un entero que indica los minutos de la reserva.
3. **Operador de entrada** >> para poder obtener desde un flujo de entrada el contenido de un objeto. Se asume el mismo formato indicado anteriormente.

Para la realización de estos ejercicios, se puede asumir la existencia (e implementación) de determinados métodos para las clases **Turista** y **GrupoTuristas**. Debe especificar sus cabeceras y justificar brevemente su necesidad.

### ◁ Ejercicio 3 ▷ Entradas y Salidas [ puntos]

Se desea añadir la posibilidad de cargar y salvar las reservas de grupos para un día desde fichero. Para ello, se establece un formato de texto que consiste en:

- Una cadena mágica `"RESERVASGRUPOS-ALHAMBRA"` y un salto de línea (para poder distinguir que es un fichero usado para almacenar información de objetos de la clase de interés).
- Las reservas en formato de texto, tal como se indica en el ejercicio 2.

Considerando que tiene disponibles los métodos de los ejercicios previos implemente las siguientes funciones externas:

1. Función **Cargar**, que recibe un nombre de archivo y un objeto de tipo **ReservasGrupos** y carga en éste el contenido del archivo. Devuelve si la operación ha tenido éxito.
2. Función **Salvar**, que recibe un nombre de archivo y un objeto de tipo **ReservasGrupos** y lo salva en el archivo. Devuelve si la operación ha tenido éxito.

### ◁ Ejercicio 4 ▷ Método que modifica objetos [ puntos]

Implemente el método **EliminarReserva** que permita eliminar la reserva realizada a un grupo en una hora determinada. Así, su argumento será un objeto de tipo **Hora**.

### ◁ Ejercicio 5 ▷ Aplicación [ puntos]

Escriba un programa completo que reciba desde la línea de órdenes el nombre de dos ficheros de texto que contienen reservas para grupos en un día particular (como se indica en el ejercicio 3). Deberá combinar tales reservas, eliminar las comprendidas entre las 12 horas y las 13.30 horas, ordenarlas en función de la hora de reserva y, finalmente, salvarlas en un fichero de salida cuyo nombre también se indica en la línea de órdenes.

Para la realización de este ejercicio, puede suponer que ya está implementado el método **OrdenarReservas()** para la clase **ReservasGrupos**.

## ◁ Ejercicio 6 ▷ Método redistribuir [1,5 puntos]

Implemente el método **RedistribuirGrupo** que permita eliminar la reserva realizada a un grupo en una hora determinada, pero, además, debe redistribuir los turistas de ese grupo entre los restantes, aplicando la siguiente estrategia: *iterativamente, cada turista se asignará al grupo actual con menos turistas (en caso de empate, se asigna a cualquiera de ellos)*.

## ◁ Ejercicio 7 ▷ Método ordenar grupos [1,5 puntos]

Implemente el método **OrdenarGrupos** que permita ordenar todos los grupos de turistas almacenados en un objeto de la clase **ReservasGrupos** alfabéticamente por apellidos y nombre (Recuerde que los objetos de esta clase almacenan nombre y apellidos).

Para la realización de este ejercicio, se puede asumir la existencia (e implementación) de determinados métodos para las clases **Turista** y **GrupoTuristas**. Debe especificar sus cabeceras y justificar brevemente su necesidad.

## ◁ Ejercicio 8 ▷ Organización en módulos y Makefile [1 punto]

1. (0,5 puntos) Escriba el contenido de los archivos de cabecera turistas.h, grupoTuristas.h y reservasGrupos.h, donde debe aparecer todo lo necesario para usar las clases **Turista**, **GrupoTuristas** y **ReservasGrupos**, sin incluir ninguna definición de función (suponga que estarán en turistas.cpp, grupoturistas.cpp y reservasGrupos.cpp).
2. (0,5 puntos) Considere los archivos que implementan las tres clases así como el programa principal.cpp indicado en el ejercicio 5. Escriba el archivo `Makefile` que permita la compilación del programa. Incluya las dependencias necesarias y un objetivo clean que permita eliminar los archivos intermedios generados en el proceso de compilación.