Apellidos, Nombre: **DNI:**

Instrucciones

Realice los ejercicios tal como se detallan en cada uno de ellos. El profesor puede proporcionar folios vacíos para poder anotar lo que se necesite para la resolución del examen. El contenido de esos folios no se evaluará pero serán entregados al profesor.

Descargue el `zip` proporcionado en Moodle llamado *OrdinariaBloqueII-ApellidosNombre.zip*. **Descomprímalo** y **renombre** la carpeta sustituyendo *ApellidosNombre* por los suyos. Por ejemplo, si mi nombre es Antonio Manuel Durán Rosal, el directorio pasa a llamarse *OrdinariaBloqueII-DuranRosalAntonioManuel* (no incluir **ni espacios ni caracteres especiales, como la ñ o tildes**).

El directorio contiene una carpeta por cada ejercicio de programación que debe resolverse utilizando un editor de texto plano y su correspondiente compilación y ejecución en el sistema operativo Linux. **Al profesor sólo se le subirán los códigos fuente sin incluir el ejecutable**. Aunque a continuación se detallan los ejercicios y el qué hay que realizar, a lo largo de los ficheros aparece la palabra **TODO** que indicará lo mismo.

Reglas y aclaraciones para todos los ejercicios

- Obligatorio el uso del calificador `const`.
- Usar las directivas `#ifndef`, `#define` y `#endif`; o `#pragma once`.
- No modificar nada de los archivos `main.cpp`, sólo descomentar aquellas partes que se vayan completando para que compile.
- Por lo general, podrá incluir las funciones que necesite en los archivos `.h` y `.cpp`, pero se evaluará el funcionamiento de los apartados que se piden.
- Comprobar los **TODO**.
- Ante cualquier duda, contactar con el profesor.

Hay que entregar

- En Moodle el directorio completo en formato `.zip` con el nombre *OrdinariaBloqueII-Apellidos Nombre.zip* (poner los apellidos y nombre del alumno).
- **Antes de darle a enviar** los documentos en Moodle, **llamar al profesor** para comprobar que todo es correcto. **No se aceptarán envíos sin la supervisión del profesor.**

Confirmo que he leído las instrucciones del examen. Firma y hora de entrega:

Ejercicio 1 — Tiempo estimado: 40min — Puntuación: 4

En un torneo de robótica se selecciona un número de robots que siempre es potencia de dos. Cada robot dispone de un tiempo limitado para cargar su batería antes de salir a competir. Tras la carga, los robots deben colocarse en fila. El equipo obtendrá la victoria si la fila cumple la siguiente propiedad:

- La suma de la carga de batería de los robots de la primera mitad de la fila debe ser exactamente la mitad de la suma de la segunda mitad. Además, esta condición debe cumplirse recursivamente para cada mitad.

Por ejemplo:

R1, 2 - R2, 4 - R3, 4 - R4, 8 - R5, 4 - R6, 8 - R7, 8 - R8, 16

Para comprobar si una fila de robots supera el reto, se programará una función denominada `dobleCargaBateria`.

Se debe realizar:

a) Defina la clase **Robot** que constará de dos atributos (nombre y carga_bateria), constructor dados dos parámetros, observadores y modificadores, sobrecarga del operador = y la función de salida show de la forma compacta: (Nombre, carga_bateria). (No hará falta especificar ninguna otra función) **(1 punto)**.

b) Implemente la función `dobleCargaBateria` **(3 puntos)**. Esta función debe devolver la suma total de las cargas si la fila cumple la propiedad, o -1 en caso contrario. Cuando el tamaño del vector es 1, siempre se cumple la condición y directamente se devuelve la carga del robot.

El programa principal probará la función con las siguientes 5 filas de robots. Las dos primeras devolverán la suma total, mientras que las tres restantes devolverán -1:

✓ R1, 1 - R2, 2 - R3, 2 - R4, 4 - R5, 2 - R6, 4 - R7, 4 - R8, 8

✓ R1, 3 - R2, 6 - R3, 6 - R4, 12 - R5, 6 - R6, 12 - R7, 12 - R8, 24

× R1, 3 - R2, 6 - R3, 6 - R4, 12 - R5, 3 - R6, 6 - R7, 6 - R8, 12

× R1, 3 - R2, 6 - R3, 3 - R4, 6 - R5, 4 - R6, 8 - R7, 4 - R8, 8

× R1, 3 - R2, 7 - R3, 2 - R4, 1 - R5, 5 - R6, 2 - R7, 1 - R8, 24

La salida por pantalla sería:

```
Primera fila:
(R1, 1) (R2, 2) (R3, 2) (R4, 4) (R5, 2) (R6, 4) (R7, 4) (R8, 8)

Segunda fila:
(R1, 3) (R2, 6) (R3, 6) (R4, 12) (R5, 6) (R6, 12) (R7, 12) (R8, 24)

Tercera fila:
(R1, 3) (R2, 6) (R3, 6) (R4, 12) (R5, 3) (R6, 6) (R7, 6) (R8, 12)

Cuarta fila:
(R1, 3) (R2, 6) (R3, 3) (R4, 6) (R5, 4) (R6, 8) (R7, 4) (R8, 8)

Quinta fila:
(R1, 3) (R2, 7) (R3, 2) (R4, 1) (R5, 5) (R6, 2) (R7, 1) (R8, 24)

El resultado de la primera fila es: 27
El resultado de la segunda fila es: 81
El resultado de la tercera fila es: -1
El resultado de la cuarta fila es: -1
El resultado de la quinta fila es: -1
```

Ejercicio 2 — Tiempo estimado: 60min — Puntuación: 6

Desde hace unos años, se está impulsando el desarrollo de **ciudades inteligentes (Smart Cities)**, donde se emplean sensores para monitorizar variables como el tráfico, la contaminación o el consumo energético. Una empresa tecnológica llamada **LoyorTech** está diseñando un sistema para instalar **sensores ambientales** en distintas zonas urbanas.

Cada sensor dispone de una **precisión distinta**, y cada zona requiere un sensor adecuado ya que dependiendo de la zona el sensor tiene mayor o menor nivel de precisión. La empresa desea instalar n **sensores en** n **zonas**, de forma que se **maximice la precision total de monitorización** obtenida.

Para resolver este problema se proporciona el siguiente esqueleto de clases:

- **sensor.h y sensor.cpp**: representa el sensor y su nivel de precisión.
- **zona.h y zona.cpp**: representa una zona urbana mediante su código, su nombre y el sensor instalado en ella.
- **solucion.h y solucion.cpp**: almacena la lista de zonas utilizadas con sus sensores asignados y la calidad total. Incluye estructuras auxiliares como los conjuntos de zonas y sensores utilizados.
- **estado.h y estado.cpp**: representa un estado dentro del algoritmo de backtracking.
- **problema.h y problema.cpp**: contiene la lógica del problema y la ejecución del algoritmo de backtracking.

Se pide:

a) Sobrecarga el operador $>$ de la clase **Solucion (0.5 puntos)**

b) Implemente la función **addZona (0.5 puntos)**

c) Implemente la función **deleteZona (0.5 puntos)**

d) Implemente la función **avanza (0.5 puntos)**

e) Implemente la función **retrocede (0.5 puntos)**

f) Implemente la función **getAlternativas (1.5 puntos)**

g) Implemente la función **esFinal (0.5 puntos)**

h) Implemente la función **ejecutaBacktracking (0.25 puntos)**

i) Implemente la función **bt (0.75 puntos)**

j) Implemente la función **actualizaMejorSolucion (0.5 puntos)**

La salida por pantalla sería:

```
Backtracking:
Precision de la solucion: 21
Zona: Dos Hermanas(cod: 0) instala Sensor (Sensor: 1, Precision: 9)
Zona: Entrenucleos(cod: 1) instala Sensor (Sensor: 0, Precision: 7)
Zona: Montequinto(cod: 2) instala Sensor (Sensor: 2, Precision: 5)
```