# Examen de prácticas (C3)

## 1. Información

- El examen tendrá una duración máxima de 2:00 horas, desde las 10:00 a las 12:00.
- Todo el código se escribirá en lenguaje **C++** y se corregirá con la versión del compilador instalado en los laboratorios de la EPS.
- Puedes descargar los archivos de partida desde este enlace.
- El `Makefile` que tienes disponible te permite crear el archivo para la entrega así: `make tgz`, como has hecho en las prácticas. También te permite:
  1. compilar el código así: `make p1` o `make p2` o símplemente `make` para compilar las dos preguntas.
  2. ejecutar el código así: `make runp1` o `make runp2`.
  3. ejecutar el código bajo valgrind así: `make valgrindp1` o `make valgrindp2`.

- Dispones de un programa principal de prueba para cada pregunta: `mainp1.cc` y `mainp2.cc` así como la salida que produce cada uno de ellos.
- Un error de compilación/enlace implicará un cero en la pregunta donde se produzca, por tanto **asegúrate de que tu código compila correctamente aunque determinadas funciones no hagan nada o no lo hagan bien**.
- Puedes hacer tantas entregas como quieras, sólo se corregirá la última. Las entregas son similares a las que has hecho durante el curso con las prácticas: https://pracdlsi.dlsi.ua.es. Recuerda ir haciendo entregas parciales mientras esté abierto el plazo de entrega, no se admiten entregas por ningún otro cauce ni fuera de plazo. **A las 12:00 ya no se podrá entregar**.
- Para que te hagas una idea de la cantidad de código a escribir, cada archivo `.cc` pedido ocupa, más o menos, esta cantidad de líneas:

| | |
|---|---|
| 45 | msort.cc |
| 34 | vehicle.h |
| 26 | vehicle.cc |

| | |
|---|---|
| 29 | gaspowered.h |
| 19 | gaspowered.cc |
| 26 | car.h |
| 20 | car.cc |

Si a ti te ocupan un número distinto de líneas, es normal, no pasa nada.

- Todas las clases y funciones de ambos ejercicios pertenecen al *espacio de nombres* `C3`. Cualquier función auxiliar que definas deberá estar dentro de este espacio de nombres para evitar colisiones.
- No se permiten apuntes en el examen, ni teléfono móvil, smartwatch, memorias usb o similar. Se expulsará del examen a cualquiera que incumpla esta norma.
- Puedes usar bolígrafo y papel en blanco para ayudarte en el desarrollo de los problemas.
- Ten tu DNI disponible para poder identificarte.

# 2. Ejercicio 1 (4 pts.)

1. Dados los siguientes prototipos de plantillas de funciones (disponibles en el fichero `msort.h`):

```
namespace C3 {
  template <typename T>
  std::vector<T> mergesort(const std::vector<T>& v);

  template <typename T>
  std::vector<T> merge(const std::vector<T>& v1, const std::vector<T>&
}
```

2. **Se pide:**
   - Escribir el código de ambas plantillas de manera que el ejemplo disponible en el fichero `mainp1.cc` compile y funcione correctamente.
   - Ambas funciones nos sirven para implementar el algoritmo de

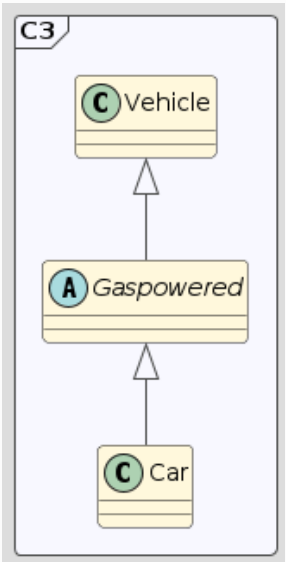
ordenación de vectores conocido como `mergesort` u *ordenación por mezcla*. La ordenación que aplicaremos es de menor a mayor.

- El funcionamiento de `mergesort` es muy sencillo, para ordenar un vector de `N` componentes:
  - Divide el vector original en dos vectores de `N/2` componentes cada uno. Si al primero de estos vectores lo llamamos `v1` y al segundo `v2` entonces `v1` contendrá las componentes `v[0]..v[(N/2)-1]` y `v2` contendrá las componentes `v[N/2]..v[N-1]`.
  - A continuación se ordenan `v1` y `v2` llamando a la función `mergesort` de forma recursiva para cada uno de ellos.
  - La función `mergesort` asume que un vector de `0` o `1` componentes y está ordenado.
  - Una vez `v1` y `v2` están ordenados, los contenidos de ambos se mezclan de forma ordenada en un solo vector. Esto lo hace la función `merge`. Por ejemplo dados `v1=[4, 6, 7]` y `v2=[1, 3, 5, 9]` el resultado de llamar a `merge(v1, v2)` es `[1, 3, 4, 5, 6, 7, 9]`

3. Dispones de la salida que proporciona el programa principal de ejemplo `mainp1.cc` en el fichero `salida-p1.txt`.

# 3. Ejercicio 2 (6 pts.)

1. Dada la siguiente estructura de clases representada mediante un diagrama **UML**:



2. **Se pide:**
   - Escribir el contenido de los archivos `vehicle.{h,cc}`, `gaspowered.{h,cc}` y `car.{h,cc}` como en las prácticas hechas durante el curso. No olvides la guarda de seguridad en los archivos `.h`. El contenido de estos ficheros será el apropiado para que el programa principal proporcionado (`mainp2.cc`) compile y ejecute sin problemas.
   - Para ello debes tener en cuenta que:
     - Las características de peso (*weight*, `float`), número de ruedas (*nwheels*, `uint8_t`), autonomía (*autonomy*, `uint16_t`) y velocidad (*speed*, `float`) pertenecen a la clase `Vehicle` y son privadas de ella.

       Para cada una de ellas tendrás que definir los métodos `set_xxx` y `get_xxx` apropiados.

       Cuando se crea un objeto de clase `Vehicle` su velocidad inicial es `0.0`, el resto de valores se especifican en un único constructor.
     - La característica de capacidad (*capacity*, `float`) del depósito de gasolina pertenece a la clase `Gaspowered`. También tendrás que definir los métodos `set_xxx` y `get_xxx` apropiados.

       Cuando se inicializa un objeto de clase `Gaspowered` la *capacidad* de su depósito se indica junto al resto de valores en un único constructor.

       La clase `Gaspowered` **es abstracta** por que los métodos `set_brand` y `get_brand` se declaran en ella como abstractos.
     - La característica de marca (*brand*, `std::string`) de un coche pertenece a la clase `Car`. Esta clase **no es abstracta**.

       Cuando se inicializa un objeto de clase `Car` la *marca* del mismo se indica junto al resto de valores en un único constructor.
     - Cada clase proporcionará en su `.h` y también dentro del espacio de nombres `C3` el *alias* `TipoPtr` tal y como hemos usado durante el curso, p.e. para la clase `C3::Car` dispondremos del *alias* `C3::CarPtr`

y así con el resto de clases.

3. Todas estas explicaciones te quedarán más claras entendiendo el diagrama de clases anterior y viendo el programa principal de ejemplo (`mainp2.cc`) proporcionado así como su salida (`salida-p2.txt`), la cual tienes entre los archivos de partida.
4. Es recomendable que implementes las diferentes clases empezando por la clase base y siguiendo por sus derivadas: `Vehicle`, `Gaspowered` y `Car`.

# 4. Ayuda

Aquí tienes el enlace a la versión PDF del libro C++ Annotations.

# 5. Requisitos técnicos

- Requisitos que tiene que cumplir este trabajo práctico para considerarse válido y ser evaluado (si no se cumple alguno de los requisitos la calificación será **cero**):
- Al principio de todos los ficheros fuente (`.cc`) entregados se debe incluir un comentario con el nombre y el NIF (o equivalente) de la persona que entrega el examen, como en el siguiente ejemplo:

```
// NIF: 12345678X
// NOMBRE: PEREZ GARCIA, ALEJANDRO
```

- El archivo entregado se llama `irp2-c3.tgz` (todo en minúsculas). En el estarán todos los ficheros `.h` y `.cc` pedidos en una carpeta llamada `irp2-c3`.

  Si por algún motivo entregas los ficheros `mainp1.cc`, `Makefile`, etc... no pasa nada.

- Las clases, métodos y funciones implementados se llaman como se indica en el enunciado (respetando en todo caso el uso de mayúsculas y minúsculas). También es imprescindible respetar estrictamente los textos y los formatos de salida que se indican en este enunciado.

## 6. Lugar y plazo de entrega

La entrega se realiza en https://pracdlsi.dlsi.ua.es. Allí puedes ver el plazo de entrega.

Puedes entregar el examen tantas veces como quieras mientras el plazo de entrega esté abierto, sólo se corregirá la última entrega (las anteriores no se borran).

El usuario y contraseña para entregar prácticas es el mismo que se utiliza en UACloud.

## 7. Detección de plagios/copias

- El examen debe ser un trabajo original de la persona que lo entrega.
- En caso de detectarse indicios de copia en el examen entregado, se tomarán las medidas disciplinarias correspondientes, informando a la dirección del DLSI por si hubiera lugar a otras medidas disciplinarias adicionales.