Examen de prácticas (C3)

Programación-II - Ingeniería Robótica

2020.05.29

Información

- El examen tendrá una duración máxima de 2 horas, desde las 10:00 a las 12:00.
- Todos los ejercicios se escribirán en lenguaje **C++**.

  Se corregirán con la versión del compilador instalado en los laboratorios de la EPS.

  Puedes ayudarte de estas referencias sobre C++.
- 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 compilar cada una de las dos *preguntas* por separado así: `make p1` y `make p2` o una tras otra así: `make`.
- Dispones de un programa principal de prueba para cada pregunta: `mainp1.cc` y `mainp2.cc`.
- Un error de compilación 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: http://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.
- 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:

20	p1.cc
20	basicrobot.cc
80	battlerobot.cc
30	cleaningrobot.cc
45	robot.cc
20	noisesensor.cc
25	sensor.cc

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

Ejercicio 1 (3 puntos)

Dado el tipo:

```
using TMatrix = std::vector< std::vector<int> >;
```

recuerda que esta forma de emplear `using` *es similar a un* `typedef`,

En un fichero llamado **p1.cc** escribe una función

```
std::vector<int> diagonal(const TMatrix& a) { ... }
```

para que dada una matriz bidimensional de enteros `a` de dimensiones arbitrarias $m \times n$ devuelva en un `std::vector<int>` los valores que se obtienen al sumar todos los elementos que se encuentran en la misma diagonal descendente de a, esto es, elementos a_{ij} para los que $d = i - j$ es constante. Por ejemplo, si

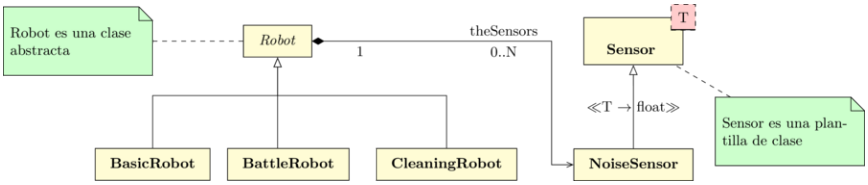
$$a = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \tag{1}$$

la función devolverá el vector `[9, 15, 18, 21, 11, 4]`.
Recuerda:

- La función `diagonal` **no imprime nada**.
- Dispones de un programa principal de prueba (`mainp1.cc`) y puedes compilarlo con ayuda del `Makefile` así: `make p1`, y ejecutarlo así: `make runp1` o bajo valgrind así: `make valgrindp1`, de este modo comprobarás que no deja memoria sin liberar.

Ejercicio 2 (7 puntos)

Dada la siguiente estructura de clases:



A destacar de este diagrama:

- La clase *Robot* es una clase abstracta.
- Las relaciones de herencia representan relaciones " *Es Un* ".
- `Sensor` es una plantilla de clase.
- `NoiseSensor` deriva de una instanciación de `Sensor` en la que `T == float`.
- La relación entre `Robot` y `NoiseSensor` es de *composición*.

Lo que tienes que hacer:

1. Escribir el contenido de los archivos `'.cc'` para cada uno de los archivos `'.h'` entregados. Al igual que en las prácticas de la asignatura, estos archivos se llamarán igual y sólo cambiará la extensión, p.e. `robot.h` → `robot.cc`.
2. Asegurarte que el código que escribas compila y enlaza con el `Makefile` proporcionado y el ejemplo de programa principal que tienes en `mainp2.cc`.
3. Comprobar que la ejecución de `p2` no falla ni deja *memoria dinámica* sin liberar (puedes ayudarte de `make valgrindp2` para ejecutarla bajo valgrind). La salida que tiene que producir al ejecutar `make runp2` es esta:

```
$ make run
./main
0. Total robots: 0
1. Total robots: 1
Could not add Robot (BasicRobot_10 to R2D2 @(10, 0).
```

```
Could not add Robot (BasicRobot_20 to R2D2 @(10, 1).
Could not add Robot (BasicRobot_30 to R2D2 @(10, 2).
Could not add Robot (BasicRobot_40 to R2D2 @(10, 3).
Could not add Robot (BasicRobot_50 to R2D2 @(10, 4).
Could not add Robot (BasicRobot_60 to R2D2 @(10, 5).
Could not add Robot (BasicRobot_70 to R2D2 @(10, 6).
Could not add Robot (BasicRobot_80 to R2D2 @(10, 7).
Could not add Robot (BasicRobot_90 to R2D2 @(10, 8).
Could not add Robot (BasicRobot_100 to R2D2 @(10, 9).
2. BattleRobot ar2 num. sensors: 800
3. Total robots: 111
4. BattleRobot br num. sensors: 73
5. Total robots: 121
```

Aclaraciones:

1. Todas las clases pertenecen al *espacio de nombres* `C3`. Para entender mejor el contenido de los archivos `basicrobot.h`, `battlerobot.h` y `cleaningrobot.h` echa un vistazo primero a `robot.h`

2. Los métodos *getters* y *setters* que debes implementar hacen referencia inequívoca a la *variable* o *dato* con el que deben trabajar.

3. El método `Robot::get_sensor(uint s)` devuelve el puntero que hay en la posición `s` del vector de sensores de ruido de un `robot`.

4. En aquellas clases donde el destructor no tenga que hacer nada *debes proporcionar una implementación vacía del mismo*, de lo contrario tendrás errores de enlace.

5. El método `get_num_sensors` de cualquier *robot* devuelve el número de sensores que se le han añadido, salvo en el caso de un `BattleRobot` en cuyo caso devuelve los suyos propios *más* los de todos los robots que tenga en su mapa.

6. Un `BasicRobot` no puede andar ni puede hablar.

7. Un `BattleRobot`:

   a) Dispone de un mapa de *robots atacantes* y al ser creado dimensiona e inicializa con punteros nulos este mapa.

   b) También, al ser creado, inicializa su *nivel de ataque* y de *daño* sufrido a `0.0`.

c) Puede andar pero no puede hablar.

d) En su método `add_robot_at`:

- Solo añade el robot `r` a su mapa (`map`) si las coordenadas son validas (`0 <= x,y < theMapSize`).
- Si en esas coordenadas ya había un robot, libera la memoria del robot existente y luego guarda el puntero `r`.
- Si puede añadir el robot `r` devuelve `true`.
- Si no puede añadir el robot `r` emite la señal `robotNotAddedSignal` (en la que el parámetro `Robot*` representa al robot que no se ha podido añadir, `BattleRobot*` es éste `BattleRobot` y los parámetros `x` e `y` representan la posición en la que no se ha podido añadir el robot) y luego devuelve `false`.

8. Un `CleaningRobot` puede andar y puede hablar.

9. La variable de clase `Robot::totalRobotsCreated` lleva la cuenta de todos los robots creados, sean de la clase que sean. Sólo se incrementa, no se decrementa nunca.

10. **IMPORTANTE:** La plantilla de clase `Sensor` actúa como clase base de `NoiseSensor` con `T == float`. Recuerda que para poder usar sus métodos en `NoiseSensor`, en el archivo `noisesensor.cc` deberás incluir `sensor.cc`.

Ayuda:

- Implementa los archivos `'.cc'` en este orden, te será más sencillo:

  1. `sensor.cc`
  2. `noisesensor.cc`
  3. `robot.cc`
  4. `basicrobot.cc`
  5. `cleaningrobot.cc` y por último
  6. `battlerobot.cc`

- Una vez creado un fichero `'.cc'` puedes comprobar si compila bien así:

```
make sensor.o
make robot.o
etc...
```

- El archivo `macros.h` hace uso del preprocesador para definir un tipo puntero al dato que sea añadiéndole el sufijo `Ptr` de manera similar a como hemos hecho durante el curso, no tiene mayor importancia, no pierdas tiempo con él ahora.

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: GARCIA GARCIA, CARMEN
```

- El archivo entregado se llama `irp2-c3.tgz` (todo en minúsculas). En el estarán todos los ficheros `'.cc'` pedidos en una carpeta llamada `irp2-c3`.
- Al descomprimir el archivo `irp2-c3.tgz` se crea un directorio de nombre `irp2-c3` (todo en minúsculas).
- Dentro del directorio `irp2-c3` sólo estarán los archivos pedidos con el nombre apropiado cada uno de ellos. Si entregas los ficheros `'.h'`, `mainp1/2.cc`, `Makefile` 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.
- **Lugar y fecha de entrega :** https://pracdlsi.dlsi.ua.es. Puedes entregar el examen tantas veces como quieras, 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.

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.