Examen de prácticas (C4)

Programación-II - Ingeniería Robótica

2020.07.03

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/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: 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:

25	p1.cc
20	pressuresensor.cc
25	pressuresensor.h
100	rubikscubesolver.cc
60	rubikscubesolver.h

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

Ejercicio 1 (4 puntos)

Dado el tipo:

```
// F representa el tipo: Funcion que recibe un dato de tipo double y
// devuelve un dato de tipo double.
using F = double (double);
```

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

En un fichero llamado **p1.cc** escribe una función con el siguiente prototipo:

```
double root(F f, double a, double b, double epsilon);
```

Esta función `root`, dada una función $f : \mathbf{R} \to \mathbf{R}$, continua en el intervalo $[a..b]$ y $f(a) \times f(b) < 0$, devolverá con una precisión `epsilon` el valor de `x` tal que $f(x) = 0{,}0$.

Suponiendo que haya más de una solución en el intervalo, devuelve el primer valor de `x` encontrado que cumple lo anterior.

Ejemplos:

```
root(sin, 0.5, 4.0, 0.0000000001) = 3.14159 -> approx. PI

// myfn(x) = 2.0*x - 3.0
root(myfn, 7.0, -7.0, 0.0001)     = 1.49995 -> approx. 3.0/2.0

// myfn2(x) = 3.0*x*x - 6.0
root(myfn2, 7.0, -7.0, 0.0001)    = -1.41421 -> approx. -sqrt(2.0)
```

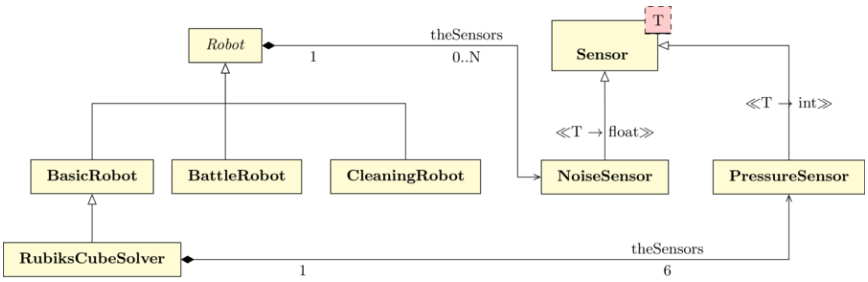
Recuerda:

- La función `root` **no imprime nada**.

- Con los límites del intervalo puede ocurrir que $a > b$ o que $a < b$.
- 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 (6 puntos)

Dada la siguiente estructura de clases basada en la que ya conoces de la convocatoria anterior (C3):



A destacar de este diagrama:

- Las clases que ya conoces del examen de mayo se comportan de la misma manera que lo hacían (por eso los mensajes en UA-Cloud relativos a que terminarais el ejercicio).
- Se introducen dos clases nuevas: `PressureSensor` y `RubiksCubeSolver`.
- `PressureSensor` deriva de una instanciación de `Sensor` en la que `T == int`.
- La relación entre `RubiksCubeSolver` y `PressureSensor` es de *composición*.

Lo que tienes que hacer:

- Escribir el contenido de los archivos `pressuresensor.h`, `pressuresensor.cc`, `rubikscubesolver.h` y `rubikscubesolver.cc` además de entregar el código de los archivos `'.cc'` del resto de clases. 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`.

- Añadir a los archivos `'.h'` la guarda de seguridad para evitar *includes* repetidos.
- Asegurarte que el código que escribas compila y enlaza con el `Makefile` proporcionado y el ejemplo de programa principal que tienes en `mainp2.cc`.
- Debes implementar todos los métodos, aunque estén vacíos, o tu código no compilará/enlazará.
- 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.
- 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 runp2
./p2
Facing face is: FRONT
No existing pressure sensor is: -100
bad Facing face produces: UP
Rubik's Cube Solver data:
 RCS name: First Rubik's Cube Solver
  Sensor [PS_0] = 3
  Sensor [PS_1] = 3
  Sensor [PS_2] = 2
  Sensor [PS_3] = 3
  Sensor [PS_4] = 222
  Sensor [PS_5] = 3
  Facing face: 0 (UP)
  Can Walk: NO
  Can Talk: YES
6. Total robots: 1
```

Aclaraciones:

1. Sólo las dos nuevas clases (`PressureSensor` y `RubiksCubeSolver`) pertenecen al *espacio de nombres* `C4`. Las otras pertenecen al *espacio de nombres* `C3`.

2. Un `RubiksCubeSolver`:

   a) No puede andar pero sí puede hablar.

   b) Sabe en todo momento qué cara del cubo tiene delante y esto se representa por un valor de un tipo *enumerado* llamado `RubiksCubeSolver::Face` (como ves el tipo `Face` se declara dentro de la clase y además en la parte pública), el cual podrá ser uno de estos seis valores: `UP, DOWN, FRONT, BACK, LEFT, RIGHT`. Inicialmente tiene el valor `FRONT`.

   Para leerlos/cambiarlos se emplean los métodos: `set_facing_face` y `get_facing_face`. El método `get_facing_strface` devuelve como cadena el valor del enumerado correspondiente, p.e. si el valor es `RubiksCubeSolver::UP` el método devuelve la cadena `"UP"`, si es `RubiksCubeSolver::FRONT` el método devuelve la cadena `"FRONT"`, etc...

   Los prototipos de estos métodos son:

   ```
   void set_facing_face (RubiksCubeSolver::Face f);
   RubiksCubeSolver::Face get_facing_face ();
   std::string get_facing_strface ();
   ```

   c) El método `set_sensor` cambia el valor del sensor que se le pasa como argumento (`sid`) si este está entre $[0 \ldots 6)$ y si no, no hace nada. El método `get_sensor` devuelve el valor del sensor que se le pasa como argumento si este está entre $[0 \ldots 6)$ y si no devuelve la constante `RubiksCubeSolver::NOVALUE` que se inicializa a `-100`. Esta *constante de instancia* se declara en la parte pública de la clase.

   El valor inicial de los `6` sensores es `3`.

   Los prototipos de estos métodos son:

   ```
   void set_sensor(int sid, int sv);
   int get_sensor(int sid);
   ```

   d) Define el operador de salida `operator` $<<$ como función amiga de la clase y produciendo el resultado que puedes ver en el ejemplo anterior, el cual se obtiene al ejecutar el programa principal de prueba `mainp2.cc` que se te entrega.

3. **IMPORTANTE:** La plantilla de clase `Sensor` actúa como clase base de `PressureSensor` con `T == int`. Recuerda que para poder usar sus métodos p.e. en `PressureSensor`, en el archivo `pressuresensor.cc` deberás incluir `sensor.cc`.

Ayuda:

- Implementa los nuevos archivos `'.h'` y `'.cc'` en este orden, te será más sencillo:
  1. `pressuresensor.{h,cc}`
  2. `rubikscubesolver.{h,cc}`
- Una vez creado un fichero `'.cc'` puedes comprobar si compila bien así:

```
make pressuresensor.o
make rubikscubesolver.o
etc...
```

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-c4.tgz` (todo en minúsculas). En el estarán todos los ficheros `'.cc'` pedidos en una carpeta llamada `irp2-c4`.
- Al descomprimir el archivo `irp2-c4.tgz` se crea un directorio de nombre `irp2-c4` (todo en minúsculas).
- Dentro del directorio `irp2-c4` sólo estarán los archivos pedidos con el nombre apropiado cada uno de ellos. Si entregas los ficheros `'.h'` *no necesarios*, `mainp1/2.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.

- **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.