REGLAS DE CODIFICACIÓN Y BUENAS PRÁCTICAS DE PROGRAMACIÓN

Marcos Rodríguez Sanz

Las reglas y prácticas de programación llevadas a cabo en la entrega se pueden resumir en ocho grandes grupos:

1. Optimización en el uso de la memoria: Esto quiere decir que se tratará de tener ocupado en todo momento el mínimo espacio de la memoria disponible para que el programa no requiera de la asignación de muchos recursos por parte de la CPU.

   La forma de implementar este principio en el código ha sido la siguiente

   - No se permite y se trata de evitar a toda costa el paso de parámetros que sean de gran tamaño (como instancias de clases, vectores o deques) por valor, para no duplicar estos elementos en la memoria. Este paso de parámetros se hará siempre por referencia.
   - Las instancias que sean de carácter temporal contarán en su clase con un método destructor para su eliminación cuando ya no sea necesario mantenerlas en la memoria. En este caso los únicos objetos que son temporales son los mensajes que se almacenan en la cola, que serán procesados y después eliminados con el destructor.
   - Los elementos repetitivos del programa, como los drones y los hilos creados durante el mismo, se almacenarán en vectores que serán eliminados una vez no sea necesario su existencia posteriormente en la ejecución del programa.
   - Los elementos que sean creamos de forma dinámica se eliminarán mediante el comando delete cuando ya no sean necesarios en el programa para garantizar el futuro acceso a esas posiciones de memoria posteriormente durante la ejecución del programa.

2. Evitar el uso repetitivo de la ALU en sentencias idénticas. La forma en la que hemos implementado este principio es mediante el uso de variables centinela (o flags) de carácter entero o booleano cuando, especialmente en bucles iterativos, se pudiera prever que se le va a pedir a la ALU que repita operaciones idénticas.

3. Legibilidad y organización de las clases. Este principio se ha implementado en el código de la siguiente manera:

- Las clases siempre se nombrarán empezando por una letra mayúscula, y sus métodos empezando por una letra minúscula
- Los métodos de las clases que sean muy genéricos y poco específicos (como getters, setters u otros) se nombrarán con letras mayúsculas y minúsculas, mientras que los que sean más específicos se nombrarán en minúsculas y separados por guiones bajos.

4. Simplicidad y fácil manejo y utilización del programa por parte del usuario. Este principio es fundamental en el diseño de un software, en nuestro programa se ha implementado de la siguiente forma.

    - Todas las acciones e instrucciones que introduzca el usuario tendrán implementados mensajes por pantalla que le indiquen el rumbo que ha tomado el programa.
    - Ante posibles entradas peligrosas del usuario se prevén excepciones, peticiones de entrada de la instrucción o si fueran muy graves la finalización del programa.
    - El algoritmo y la versión del programa se han simplificado para que no requiera que el usuario tenga que introducir grandes cantidades de información para enviar un mensaje. Hemos reducido el número de velocidades y de direcciones a introducir por el usuario para ello.

5. Correcta utilización de los ficheros. A la hora de abrir un fichero se preverá la posibilidad de que este no pueda ser abierto y se finalizará el programa. A la hora de finalizar el programa se cerrarán todos los ficheros que estén abiertos, o antes siempre y cuando no sean necesarios para el desarrollo posterior del programa.
6. La gestión de los recursos compartidos, como son la pantalla, la cola limitada o la salida a fichero, se hará mediante un mutex que se identifica como miermbro de la clase cola limitada; de tal forma que podamos asegurar que dos hilos no están accediendo a estos recursos simultáneamente y se pueda altererar la ejecución del programa.
7. La mayor parte de las salidas por pantalla y a fichero que se hagan en el programa se harán mediante el functor genérico Imprimir para reducir la redundancia de estas sentencias.
8. Toda la aleatoriedad del programa se hará mediante functores y el uso de la librería random generando números pseudoaleatorios. Las instancias de estas clases se crearán una sola vez para evitar el reinicio de la semilla y que se repitan valores,