VNIVERSITAT DE VALÈNCIA

ESTRUCTURAS DE DATOS Y ALGORITMOS
***Grado en Ingeniería Multimedia (2º).* Curso 2022-23**

Práctica Nº 1: Programación con asertos
Periodo de realización: Semana del **03/10/2022 a 07/10/2022**

Objetivos

- Familiarizarse con el entorno de desarrollo de programas en *VSCode*.
- Generación de proyectos con *make*.
- Documentación de programas con *Doxygen*.
- Utilización de asertos en *C++*.

Introducción

En aplicaciones para la visualización de gráficos es necesario poder manejar el concepto de punto sobre el espacio de representación (escenario, lienzo, etc.). Este espacio puede ser bidimensional (2D) o tridimensional (3D). Suponiendo el caso más simple, 2D, un punto en el espacio es una entidad formada por un par de números que representan su localización en cada una de las dos dimensiones del espacio (x,y). En la imagen 1 se muestra la localización del punto A, cuyas coordenadas son (4,5).

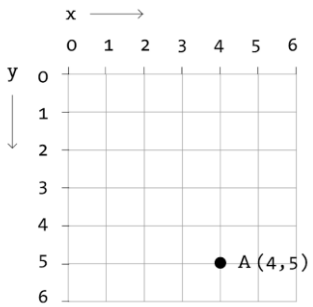

Imagen 1. Representación de un punto en el espacio 2D.

A partir del concepto de punto es posible construir conceptos más complejos como, por ejemplo, el de figuras geométricas. En particular, las figuras tipo triángulo son especialmente útiles para modelar la superficie de otros objetos más complejos. En este caso, un triángulo se puede definir a partir de 3 puntos, que representan sus vértices y que permiten establecer su localización y sus dimensiones.

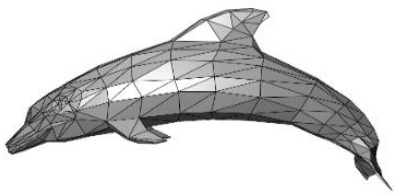

Ilustración 1. Ejemplo de objeto construido con triángulos

Ejercicios

Tarea 0: Prerrequisito

Es necesario haber clonado desde *VSCode* el repositorio de prácticas que tienes asignado, tal como se ha explicado en el vídeo disponible en Aula Virtual.

Tarea 1: Iniciar el nuevo proyecto

Se deben descargar desde el Aula Virtual todos los archivos con código fuente proporcionados como material inicial. Replica en la carpeta "`Pr1`" del repositorio de prácticas la estructura de carpetas de la práctica 0 (ejemplo) para organizar adecuadamente tu proyecto. Utiliza para ello el navegador de archivos del sistema operativo. Ubica en esta estructura de carpetas los archivos descargados como material inicial (carpeta "*`src`*" o "*`include`*", según corresponda).

Abre la carpeta "`Pr1`" desde *VSCode* y comprueba que es posible navegar y editar los archivos disponibles.

Verifica ahora el funcionamiento del guion de compilación (*`Makefile`*) sobre este proyecto, tal como se ha explicado en el vídeo disponible en Aula Virtual:

1. Ejecuta en la ventana "Terminal" de *VSCode* la orden *`make`*, o bien, *`make all`*. El programa *`make`* tomará como entrada el archivo *`Makefile`*, ya incluido en la carpeta "`Pr1`" del repositorio, y ejecutará la secuencia completa de compilación de archivos allí especificada.
2. Debes comprobar que los archivos se compilan sin errores.

Debes hacer *`commit`* de los cambios realizados en el proyecto y propagarlos al servidor (*`push`*). Indica como mensaje de commit el texto "`Pr1: Carga inicial de archivos`".

Revisa el código proporcionado, clase Punto2d y programa main.cpp. El programa principal contiene una secuencia de pruebas para realizar las comprobaciones necesarias durante la práctica. La función *`main()`* tiene definida la secuencia de test a realizar y las correspondientes llamadas a funciones deben ser comentadas/descomentadas durante el desarrollo de la práctica para comprobar que se han realizado correctamente las tareas planteadas.

Ejecuta la aplicación mediante la orden *`make run`* (en la ventana "Terminal" de *VSCode*). Comprueba que los resultados mostrados en pantalla coinciden con los indicados en los comentarios del código para el *`Test1`*.

Tarea 2: Construir la clase Triángulo

Implementa la clase *`Triangulo`* e incorpórala al proyecto de la práctica. Como se ha comentado en la introducción, un triángulo es una figura geométrica que se puede representar con 3 puntos (objetos de la clase *`Punto2d`*) que permiten localizar los 3 vértices del triángulo[1] y que lo caracterizan completamente con el fin de, por ejemplo, componer otros objetos más complejos.

[1] Es habitual que los vértices se ordenen en sentido antihorario.

La nueva clase, además de contener sus tres vértices, tiene definidas las siguientes operaciones:

Operación	Axioma/Comportamiento
`Triangulo() → Triangulo`	Crea un triángulo cuyos vértices son: (0,0); (2,0); (1,2)
`SetV0(Punto2d) → Triangulo`	Establece el primer vértice del triángulo en el punto indicado.
`SetV1(Punto2d) → Triangulo`	Establece el segundo vértice del triángulo en el punto indicado.
`SetV2(Punto2d) → Triangulo`	Establece el tercer vértice del triángulo en el punto indicado.
`Desplazar(float,float) → Triangulo`	Desplaza la posición del triángulo (sus vértices), de acuerdo con el desplazamiento indicado por los argumentos.
`Rotar(float) → Triangulo`	Realiza una rotación del triángulo (de sus vértices) respecto al origen de coordenadas, de acuerdo con el ángulo (en radianes) indicado por el argumento.

Adicionalmente, la clase sobrecarga los operadores == y << para permitir la comparación de triángulos e imprimir sus vértices por pantalla. Se considera que dos triángulos son iguales si sus tres vértices coinciden (en el mismo orden). Imprimir el triángulo significa escribir en pantalla las coordenadas de los puntos asociados a sus vértices.

Al construir esta clase se deben tener en cuenta dos restricciones:

- Los triángulos siempre se deben poder dibujar completamente sobre el escenario en el que se trabaja. Ese escenario tendrá unas determinadas dimensiones de anchura y altura (x e y). En esta práctica, se supone que se va a trabajar sobre una pantalla con dimensiones 1920x1080. No puede existir ningún triángulo que tenga alguno de sus vértices fuera de las dimensiones de la pantalla. Estas dimensiones se deben definir como constantes al implementar la clase *`Triangulo`*.
- Los 3 vértices de un triángulo deben ser diferentes entre sí para que la figura esté correctamente especificada. Si dos vértices coinciden es imposible formar un triángulo[2].

Las dos anteriores restricciones tienen como consecuencia que al implementar la clase *`Triangulo`* se debe incluir lo que se denomina un *invariante de clase*. El invariante de clase es un aserto que se debe cumplir siempre para cualquier objeto de la clase, si no se cumple significa que los valores del objeto son incorrectos. Estrictamente hablando, el invariante de clase se debería comprobar al principio y al final de todos los métodos de la clase que pueden modificar el objeto (métodos no *const*). Sin embargo, en la práctica es suficiente con que lo comprobaremos **al final de los métodos que modifican el objeto**, incluidos los constructores[3]. Se debe, por tanto, definir el invariante de la clase *`Triangulo`* e implementarlo en un método privado *`Triangulo::Invariante()`* e invocarlo adecuadamente (como aserto de postcondición) en los métodos de la clase que corresponda.

[2] Evidentemente, 3 puntos diferentes pero alineados (mismo ángulo respecto al origen) tampoco forman un triángulo, pero por simplicidad aquí no se considera esa restricción.
[3] Actúa como una postcondición de cualquier método de la clase que asegura al finalizar siempre se obtiene un objeto válido.

La definición del invariante de clase se debe ajustar a la siguiente especificación formal:

```
func Invariante (c: Triangulo) dev b: Booleano
{Pre: cierto}
{Post:
```

$$b = (\forall \alpha: 0 \leq \alpha \leq 2: (\ 0\ <=\ vertice[\alpha].x\ <=\ MAX\_X\ ) \wedge (\ 0\ \leq\ vertice[\alpha].y\ \leq\ MAX\_Y\ )) \wedge (vertice[0]\ \neq\ vertice[1]) \wedge (vertice[0]\ \neq\ vertice[2]) \wedge (vertice[1]\ \neq\ vertice[1])$$

```
}
```

En la anterior especificación, $vertice[0]([1]\ y\ [2])$ hacen referencia a los tres vértices del triángulo y MAX_X y MAX_Y hacen referencia a las dimensiones máximas del escenario en anchura y altura (1920 y 1080, respectivamente).

Para realizar esta tarea es muy importante leer el código de la clase *Punto2d*, con especial interés en los comentarios que acompañan a cada método, para entender el significado y la forma de utilizar las operaciones que contiene. Este proceso es fundamental para implementar correctamente la clase *Triangulo*, que necesariamente se debe basar en la clase *Punto2d*.

Registra en Git los cambios realizados en el código (*commit*). Utiliza mensajes descriptivos de los cambios realizados.

Tarea 3: Verificar la clase Triangulo

El programa principal del proyecto, ya proporcionado, incluye una batería de tests para comprobar el correcto funcionamiento de todo el proyecto. Efectúa de manera sistemática los 4 test previstos y comprueba que en cada caso la ejecución del programa muestra la salida correcta, indicada en los comentarios del código. Para ello, debes "activar" en la función principal solo la línea correspondiente a la función de test que se desea realizar (comenta y descomenta adecuadamente esas líneas).

Es posible que, como resultado de esta tarea detectes errores en la clase *Triangulo*. Corrígelos y vuelve a realizar el proceso de verificación.

Recuerda registrar en Git los cambios realizados en el código (*commit*). Utiliza mensajes descriptivos de los cambios realizados.

Tarea 4: Generación de documentación

El código del programa deberá estar correctamente documento con Doxygen, de acuerdo con la guía de estilo. Una vez acabado el programa, se deberá ejecutar la orden *make doxygen* para generar la documentación del programa. Dicha documentación deberá ser generada exclusivamente en formato en *html*. Además, el título del proyecto en la documentación debe corresponder con la práctica realizada. Mantener el título por defecto se penalizará con -0.5 ptos. Para ello, se debe modificar adecuadamente el archivo de configuración proporcionado.

La documentación NO debe ser entregada ni subida al repositorio. El profesor comprobará la generación de la documentación durante la corrección.

5: Evaluación de la práctica

Se evaluará el material disponible en el repositorio de cada pareja de prácticas. No es necesario subir ninguna documentación ni material a través de Aula Virtual. Para la corrección se utilizará el código fuente y la fecha del último *commit* realizado antes de la fecha límite de entrega de cada grupo. Cualquier versión subida con posterioridad a esa fecha será considerada como fuera de plazo y, por tanto, descartada.