Ejercicios: Programar una librería de cálculo de traslación/rotación/escalado

- Clases a implementar:
  - Clase Vector4<typename t>:
    - Representa un vector de 4 componentes, de tipo "indefinido" (puede ser integer, float, etc...).
    - Métodos de constructor, acceso a componentes, normalización, operadores "suma", "resta" , "multiplicación escalar (dot product)" y "multiplicación vectorial (cross product)"
  - Clase Matriz4x4<typename t>
    - Representa una matriz de 4x4 elementos. Se puede implementar como un array de 4 Vector4<t>, una matriz t[4][4], o una matriz t[16] (lo que más cómodo sea, cada implementación tiene sus ventajas e inconvenientes)
    - Métodos de constructor: Dependiendo del párrafo anterior, puede recibir un número de parámetros distinto. Se deja libertad de implementación, como mínimo debe haber un constructor por defecto para una matriz identidad y una matriz "0".

Ejercicios: Programar una librería de cálculo de traslación/rotación/escalado

- Clase Matriz4x4<typename t>
  - Métodos de acceso a datos: Se deja libertad de implementación, pero el método debe dejar acceder a una posición de la patriz dada por "fila/columna"
  - Métodos de operaciones:
    - Multiplicación Matriz4x4*Matriz4x4 (resultado tipo Matriz4x4)
    - Multiplicación Matriz4x4*Vector4 (resultado tipo Vector4)
    - Multiplicación Matriz4x4*escalar (resultado tipo Matriz4x4)
    - División Matriz4x4
    - Sumas/restas Matriz4x4

Ejercicios: Programar una librería de cálculo de traslación/rotación/escalado

- Clase Quaternion<typename t>
  - Hereda/extiende la clase Vector4
  - Método constructor para crear un cuaternión a partir de un vector y un ángulo en grados ó radianes (podéis decidir, pero las librerías matemáticas normalmente usan radianes)
  - Métodos para obtener ángulo y vector.
  - Método para obtener una matriz de rotación creada a partir de este cuaternión

Ejercicios: Programar una librería de cálculo de traslación/rotación/escalado

- Clase Render:
  - Representa el framebuffer de nuestra aplicación. Se puede implementar como una matriz bidimensional de tipo "char". Cada casilla de la matriz representa un pixel, que puede ser "1" ó "0"
  - El framebuffer tiene la coordenada "0,0" lo más cerca del centro de la matriz
  - El constructor recibirá por parámentros un "ancho" y un "alto", representando la resolución de pantalla que se usará. La clase creará una matriz llamada "buffer" con el tamaño dado.
  - El buffer tiene la coordenada "0,0" lo más cerca del centro de la matriz. Ej:
    - Para un buffer de 5x5:
      - La coordenada 0,0 (centro de la pantalla) del buffer es la posición "2,2" de la matriz.
      - La coordenada "-2,-2" (esquina inferior izquierda de la pantalla) del buffer es la posición 0,0 de la matriz.
      - La coordenada "2,2" (esquina superior derecha de la pantalla) es la posición "4,4" de la matriz
      - Cualquier acceso mayor que 2 o menor que –2 es inválido (se ignora)
  - Métodos:
    - PutPixel(int x, int y): Activará un pixel (pondrá el valor 1) en la posición "x,y" de la matriz
    - resetBufer(): Limpiará el buffer actual (pondrá un 0 en todas lasposiciones de la matriz)
    - Draw(): Mostrará por terminal el estado actual del buffer.

Ejercicios: Programar una librería de cálculo de traslación/rotación/escalado

- Programa principal:
  - Usando las clases creadas anteriormente, se pide implementar un programa que interactúe con el usuario por terminal, y que realice las siguientes operaciones:
    - Crear una clase "Render" de tamaño 11x11 píxeles
    - Crear una variable "punto" usando la clase Vector4, posicionarlo en la coordenada x=3, y=3, z=0.
    - Dibujarlo en el buffer (put pixel), y mostrar el buffer por terminal
    - Moverlo 6 unidades a la izquierda
    - Dibujarlo en el buffer y mostrarlo
    - Rotar el punto 45º respecto del eje Z (origen de coordenadas)
    - Dibujarlo en el buffer y mostrarlo
- Se pueden realizar las rotaciones usando cuaterniones o matrices de rotación euler.