



---

## PRIMERA PRÁCTICA

### VECTORES LIBRES DE TRES DIMENSIONES

- **Objetivo**
  - Se desea codificar en **C++** el tipo abstracto de datos **Vector3D**: vector libre de tres dimensiones
    - $\vec{v} = v_1 \vec{i} + v_2 \vec{j} + v_3 \vec{k} = (v_1, v_2, v_3)$
  - **Importante**
    - Se codificarán, al menos, **dos** versiones de la clase **Vector3D** utilizando dos representaciones diferentes de los atributos.
    - Cada versión se codificará en un directorio diferente:
      - version1
      - version2
- **Especificación de la clase Vector3D**
  - **Atributos**
    - Algunas de las posibles representaciones del vector son:
      - Tres datos de tipo *double*.
      - Array de datos de tres componentes de tipo *double*.
      - Vector de la STL con tres componentes de datos de tipo *double*.
  - **Constructores**
    - Constructor sin argumentos
      - **Vector3D()**
        - Crea un **Vector3D** nulo.
      - Postcondición
        - $(get1()=0) \text{ and } (get2()=0) \text{ and } (get3()=0)$
    - Constructor parametrizado
      - **Vector3D(v1, v2, v3:Real)**
        - Crea un nuevo vector a partir de los valores de las componentes.
          - $\vec{v} = v_1 \vec{i} + v_2 \vec{j} + v_3 \vec{k} = (v_1, v_2, v_3)$
      - Postcondición
        - $(get1()=v1) \text{ and } (get2()=v2) \text{ and } (get3()=v3)$
    - Constructor de copia
      - **Vector3D(v: Vector3D)**

- Crea un nuevo vector a partir de otro vector.
- Postcondición
  - $(get1() == v.get1())$   
 $and (get2() == v.get2())$   
 $and (get3() == v.get3())$
- **Observadores**
  - **Operaciones de consulta de cada una de las componentes del vector**
    - *Real get1()*
      - Obtiene la primera componente del vector:  $v_1$
    - *Real get2()*
      - Obtiene la segunda componente del vector:  $v_2$
    - *Real get3()*
      - Obtiene la tercera componente del vector:  $v_3$
    - **Observación**
      - En C++, estas funciones deben tener el calificador **const**
  - **Módulo**
    - *Real modulo()*
      - Calcula el módulo del vector.
        - $módulo(\vec{v}) = \|\vec{v}\| = \sqrt{v_1 * v_1 + v_2 * v_2 + v_3 * v_3}$
    - Postcondición:
      - $valorDevuelto == \text{sqrt}(get1() * get1() + get2() * get2() + get3() * get3())$
  - **Ángulos**
    - **Ángulo**
      - *Real angulo(v: Vector3D)*
        - Devuelve el ángulo en radianes entre el vector actual y vector pasado como argumento.
        - $Ángulo(\vec{u}, \vec{v}) = \arccos\left(\frac{\vec{u} * \vec{v}}{\|\vec{u}\| * \|\vec{v}\|}\right)$
      - Precondición
        - $modulo() * v.modulo() > 0$
      - Postcondición
        - $valorDevuelto == \text{acos}(dotProduct(v) / (modulo() * v.modulo()))$
    - **Alfa**
      - *Real alfa()*
        - Calcula el ángulo del vector con el eje X
      - Pre:
        - $modulo() > 0$
      - Postcondición
        - $valorDevuelto == angulo(Vector3D(1,0,0))$
    - **Beta**
      - *Real beta()*
        - Calcula el ángulo del vector con el eje Y
      - pre:

- $modulo() > 0$
  - Postcondición
    - $valorDevuelto == angulo(Vector3D(0,1,0))$
- **Gamma**
  - *Real gamma()*
    - Calcula el ángulo del vector con el eje Z
  - Pre:
    - $modulo() > 0$
  - Postcondición:
    - $valorDevuelto == angulo(Vector3D(0,0,1))$
- **Producto escalar**
  - *Real dotProduct(v: Vector3D)*
    - Calcula el producto escalar.
      - $\vec{u} * \vec{v} = u_1 * v_1 + u_2 * v_2 + u_3 * v_3$
    - Postcondición
      - $valorDevuelto == get1() * v.get1() + get2() * v.get2() + get3() * v.get3()$
- **Producto vectorial**
  - *Vector3D crossProduct(v: Vector3D)*
    - Devuelve un Vector3D que es el resultado del producto vectorial de dos vectores
      - $\vec{u} \wedge \vec{v} = \vec{w} = w_1 \vec{i} + w_2 \vec{j} + w_3 \vec{k}$
      - donde
        - $w_1 = u_2 * v_3 - u_3 * v_2$
        - $w_2 = -u_1 * v_3 + u_3 * v_1$
        - $w_3 = u_1 * v_2 - u_2 * v_1$
    - Postcondición
      - El vector  $\vec{w}$  es perpendicular a los vectores  $\vec{u}$  y  $\vec{v}$ 
        - $\vec{u} * \vec{w} = 0$ 
          - $dotProduct(valorDevuelto) == 0$
        - $\vec{v} * \vec{w} = 0$ 
          - $v.dotProduct(valorDevuelto) == 0$
      - El módulo de  $\vec{w}$  es igual al producto de los módulos de  $\vec{u}$  y  $\vec{v}$  por el seno del ángulo que forman
        - $|\vec{w}| = |\vec{u}| |\vec{v}| \sin(\alpha)$ 
          - $valorDevuelto.modulo() == modulo() * v.modulo() * sin(angulo(v))$
- **Producto mixto**
  - *Real productoMixto(v, w: Vector3D)*
    - Devuelve el resultado de calcular el producto escalar del vector actual con el vector obtenido al calcular el producto vectorial de otros dos vectores.
      - $productoMixto(\vec{u}, \vec{v}, \vec{w}) = \vec{u} * (\vec{v} \wedge \vec{w})$
    - Postcondición:
      - $valorDevuelto == dotProduct(crossProduct(v,w))$

- **Modificadores**

- **Operaciones de modificación de cada una de las componentes del vector**

- *set1(Real v)*
      - Asigna un nuevo valor “v” a la primera componente del vector.
      - Postcondición:
        - $get1() == v$
    - *set2(Real v)*
      - Asigna un nuevo valor “v” a la segunda componente del vector.
      - Postcondición
        - $get2() == v$
    - *set3(Real v)*
      - Asigna un nuevo valor “v” a la tercera componente del vector.
      - Postcondición
        - $get3() == v$

- **Operaciones de modificación de todas las componentes del vector**

- *sumConst(k: Real)*
      - Modifica el vector sumando una constante a cada componente
      - Postcondición
        - $(get1() == old.get1() + k)$   
 $and (get2() == old.get2() + k)$   
 $and (get3() == old.get3() + k)$
        - *donde “old” representa el vector original antes de ser modificado*
    - *sumVect(v: Vector3D)*
      - Modifica el vector sumando a cada componente la componente equivalente de otro vector.
      - Postcondición
        - $(get1() == old.get1() + v.get1())$   
 $and (get2() == old.get2() + v.get2())$   
 $and (get3() == old.get3() + v.get3())$
    - *multConst(k: Real)*
      - Modifica el vector multiplicando cada componente por una constante: escala el vector
      - Postcondición
        - $(get1() == old.get1() * k)$   
 $and (get2() == old.get2() * k)$   
 $and (get3() == old.get3() * k)$
    - *multVect(v: Vector3D)*
      - Modifica el vector multiplicando, por separado, cada componente por la componente de otro vector.
      - Postcondición
        - $(get1() == old.get1() * v.get1())$   
 $and (get2() == old.get2() * v.get2())$   
 $and (get3() == old.get3() * v.get3())$

- **Funciones de lectura y escritura**
  - *leerVector3D()*
    - Lee desde el teclado las componentes del vector
  - *escribirVector3D()*
    - Escribe por pantalla las componentes del vector con el formato:
      - $(v_1, v_2, v_3)$
- **Operadores**
  - Considérense los siguientes vectores para explicar los operadores
    - $\vec{u} = u_1 \vec{i} + u_2 \vec{j} + u_3 \vec{k} = (u_1, u_2, u_3)$
    - $\vec{v} = v_1 \vec{i} + v_2 \vec{j} + v_3 \vec{k} = (v_1, v_2, v_3)$
    - $\vec{w} = w_1 \vec{i} + w_2 \vec{j} + w_3 \vec{k} = (w_1, w_2, w_3)$
  - **Operador de igualdad**
    - $\vec{u} == \vec{v}$
    - Lógico operador  $==$  ( $v$ : *Vector3D*)
    - Compara el vector actual con otro vector y comprueba si las componentes son iguales una a una.
    - Postcondición
      - $valorDevuelto == (get1() == v.get1())$   
 $and (get2() == v.get2())$   
 $and (get3() == v.get3())$
    - **Observación**
      - Se deberá utilizar una cota de error para tener en cuenta la precisión de los números reales.
  - **Operador de asignación**
    - *Vector3D operador = (v: Vector3D)*
      - Devuelve el vector actual que ha sido modificado con las componentes de otro vector.
        - $\vec{u} = \vec{v}$
      - Postcondición
        - $(get1() == v.get1())$   
 $and (get2() == v.get2())$   
 $and (get3() == v.get3())$
  - **Operador de suma “+”**
    - **Operador “+” binario**
      - *Vector3D operador + (v: Vector3D)*
        - Devuelve otro vector que es la suma del vector actual y el vector pasado como parámetro.
          - $\vec{u} + \vec{v} = (u_1 + v_1, u_2 + v_2, u_3 + v_3)$
        - Postcondición:
          - $(valorDevuelto.get1() == get1() + v.get1())$   
 $and (valorDevuelto.get2() == get2() + v.get2())$   
 $and (valorDevuelto.get3() == get3() + v.get3())$

- **Operador “+” unario**
  - *Vector3D operador + ()*
    - Devuelve otro vector que es una copia del vector actual.
      - $+\vec{v} = (v_1, v_2, v_3)$
    - Postcondición
      - $(\text{valorDevuelto.get1()} == \text{get1()})$   
 $\text{and } (\text{valorDevuelto.get2()} == \text{get2()})$   
 $\text{and } (\text{valorDevuelto.get3()} == \text{get3()})$
- **Operador de la resta “-”**
  - **Operador “-” binario**
    - *Vector3D operador - (v: Vector3D)*
      - Devuelve otro vector que es la diferencia entre el vector actual y el vector pasado como parámetro.
        - $\vec{u} - \vec{v} = (u_1 - v_1, u_2 - v_2, u_3 - v_3)$
      - Postcondición
        - $(\text{valorDevuelto.get1()} == \text{get1()} - v.\text{get1()})$   
 $\text{and } (\text{valorDevuelto.get2()} == \text{get2()} - v.\text{get2()})$   
 $\text{and } (\text{valorDevuelto.get3()} == \text{get3()} - v.\text{get3()})$
  - **Operador “-” unario**
    - *Vector3D operador - ()*
      - Devuelve otro vector que es el opuesto al vector actual.
        - $-\vec{v} = (-v_1, -v_2, -v_3)$
      - Postcondición
        - $(\text{valorDevuelto.get1()} == -\text{get1()})$   
 $\text{and } (\text{valorDevuelto.get2()} == -\text{get2()})$   
 $\text{and } (\text{valorDevuelto.get3()} == -\text{get3()})$
- **Producto “por un” número real “\*”**
  - Sea  $k$  un número real
  - **Operador prefijo:  $k * \text{vector}$** 
    - Devuelve otro vector cuyas componentes se obtienen multiplicando por “ $k$ ” las componentes del vector actual.
      - $k * \vec{v} = (k * v_1, k * v_2, k * v_3)$
    - **Observación**
      - Este operador se debe codificar en C++ como una función que **no** pertenece a la clase Vector3D
      - Véase su especificación más adelante.
  - **Operador postfijo:  $\text{vector} * k$** 
    - *Vector3D operador \* (k: Real)*
      - Devuelve otro vector cuyas componentes se obtienen multiplicando por “ $k$ ” las componentes del vector actual
        - $\vec{v} * k = (k * v_1, k * v_2, k * v_3)$
      - Postcondición
        - $(\text{valorDevuelto.get1()} == \text{get1()} * k)$   
 $\text{and } (\text{valorDevuelto.get2()} == \text{get2()} * k)$   
 $\text{and } (\text{valorDevuelto.get3()} == \text{get3()} * k)$

- **Producto escalar “\*” de dos vectores**
  - *Real operador \* (v: Vector3D)*
    - Devuelve el producto escalar de dos vectores.
      - $\vec{u} * \vec{v} = u_1 * v_1 + u_2 * v_2 + u_3 * v_3$
    - Postcondición
      - $valorDevuelto == get1() * v.get1() + get2() * v.get2() + get3() * v.get3()$
- **Producto vectorial “^” de dos vectores**
  - *Vector3D operador ^ (v: Vector3D)*
    - Devuelve un Vector3D que es el resultado del producto vectorial de dos vectores
      - $\vec{u} \wedge \vec{v} = \vec{w} = w_1 \vec{i} + w_2 \vec{j} + w_3 \vec{k}$
    - donde
      - $w_1 = u_2 * v_3 - u_3 * v_2$
      - $w_2 = -u_1 * v_3 + u_3 * v_1$
      - $w_3 = u_1 * v_2 - u_2 * v_1$
    - Postcondición:
      - El vector  $\vec{w}$  es perpendicular a los vectores  $\vec{u}$  y  $\vec{v}$ 
        - $\vec{u} * \vec{w} = 0$ 
          - $dotProduct(valorDevuelto) == 0$
        - $\vec{v} * \vec{w} = 0$ 
          - $v.dotProduct(valorDevuelto) == 0$
      - El módulo de  $\vec{w}$  es igual al producto de los módulos de  $\vec{u}$  y  $\vec{v}$  por el seno del ángulo que forman
        - $|\vec{w}| = |\vec{u}| |\vec{v}| \sin(\alpha)$ 
          - $valorDevuelto.modulo() == modulo() * v.modulo() * sin(angulo(v))$
- **Funciones externas a la clase Vector3D**
  - Funciones que utilizan un objeto de la clase **Vector3D**, pero que no pertenecen a la clase **Vector3D**
  - **Operador prefijo de producto por un número real:  $k * vector$** 
    - *Vector3D operador\* (k: Real, v: Vector3D);*
    - Devuelve otro vector cuyas componentes se obtienen multiplicando por “k” las componentes del vector “objeto” pasado como parámetro
      - $k * \vec{v} = k * v_1 \vec{i} + k * v_2 \vec{j} + k * v_3 \vec{k} = (k * v_1, k * v_2, k * v_3)$
    - *Postcondición*
      - $(valorDevuelto.get1() == v.get1() * k)$   
 $and (valorDevuelto.get2() == v.get2() * k)$   
 $and (valorDevuelto.get3() == v.get3() * k)$
    - Prototipo de C++
      - *Vector3D & operador\* (double k, Vector3D const & v);*

- **Sobrecarga del operador de entrada**
  - Lee desde el flujo de entrada las componentes del vector “v” separadas por espacios
  - Prototipo de C++
    - `istream &operator>>(istream &stream, Vector3D &v);`
- **Sobrecarga del operador de salida**
  - Escribe en el flujo de salida las componentes del vector “v” con el formato:
    - $(v_1, v_2, v_3)$
  - Prototipo de C++
    - `ostream &operator<<(ostream &stream, Vector3D const &v);`
- **Observaciones**
  - Duración de la práctica 1: tres sesiones de dos horas cada una.
  - **Plazo máximo de entrega**
    - **16:00 horas del lunes 5 de marzo de 2018**
  - Se proporciona un fichero comprimido denominado “practica-1-usuario.zip” con el enunciado de la práctica 1 y dos subdirectorios
    - version1
    - version2
  - En cada subdirectorio, se incluyen los siguientes ficheros:
    - makefile
      - make:
        - Compila el código y crea un programa ejecutable denominado “principal.exe” que permite probar la implementación de la clase Vector3D.
      - make ndebug:
        - Compila el código sin incluir los asertos de comprobación de las pre y postcondiciones
      - make doc:
        - Genera la documentación de doxygen
      - make clean:
        - Borra ficheros superfluos
    - Doxyfile:
      - Fichero de configuración de doxygen
    - macros.hpp:
      - Permite utilizar macros de pantalla
    - principal.cpp:
      - Programa de prueba de la clase Vector3D
    - funcionesAuxiliares.hpp y funcionesAuxiliares.cpp:
      - Prototipo y código de la función que muestra el menú del programa principal
    - Vector3D.hpp y Vector3D.cpp
      - Ficheros que permite implementar la clase Vector3D
      - Estos ficheros deben ser completados por cada estudiante
  - Al terminar la práctica, se deberá subir un fichero comprimido denominado “practica-1-usuario.zip”,
    - donde “usuario” es el login de cada estudiante.



- y que contenga los subdirectorios
  - version1
  - version2
- **Observación**
  - Se debe usar el espacio de nombres de la asignatura: **ed**
- **Evaluación**
  - La calificación de la práctica se basará
    - en la calidad y completitud del trabajo realizado.
    - y en la **defensa presencial de cada estudiante.**
  - **Se valorará**
    - La correcta implementación de las **dos** versiones de la clase **Vector3D**
    - El correcto funcionamiento del programa principal propuesto como ejemplo.
    - La ampliación y mejora del menú del programa principal para añadir más opciones.
    - La documentación del código con doxygen.
    - La claridad del código.
    - El uso de macros de pantalla para mejorar la visualización de la información
    - **Y sobre todo**
      - Un profundo conocimiento de la práctica codificada.