Universidad de San Carlos de Guatemala
Escuela de Ciencias Físicas y Matemáticas
Rubí Esmeralda Ramírez Milián
Laboratorio Avanzado
201804565

Tarea 1.II.3

Inciso 1

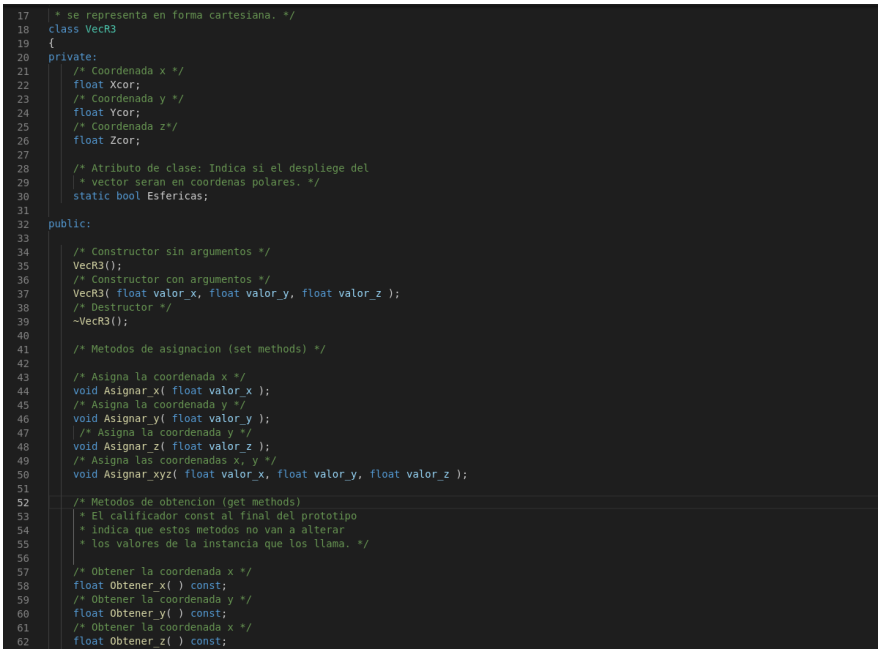
Utilizando la clase VecR2 como ejemplo, definir e implementar una nueva clase llamada VecR3, que son vectores en R^3. Esta clase debe sobrecargar los siguientes operadores:

- operator+ para suma de dos vectores.
- operator- para resta de dos vectores.
- operator- para negación de un vector.
- operator* para producto punto de dos vectores.
- operator* para multiplicar un vector por un flotante.
- operator/ para dividir un vector por un flotante.
- operator $\%$ para producto cruz de dos vectores.
- operator= para asignación.
- operator== para compraración de igualdad entre vectores. También se debe sobrecargar las siguientes funciones como amigas,
- operator* para multiplicar un flotante por un vector.
- operator $\ll$ para desplegar el vector con cout.

De la misma forma que en VecR2, se debe implementar un atributo estático que indique si al desplegar el vector, lo haga en forma cartesiana (x, y, z) o esférica (r, θ, ϕ).

Primero se debe hacer un archivo *.hpp* se encuentra en un repositorio, con el nombre VecR3

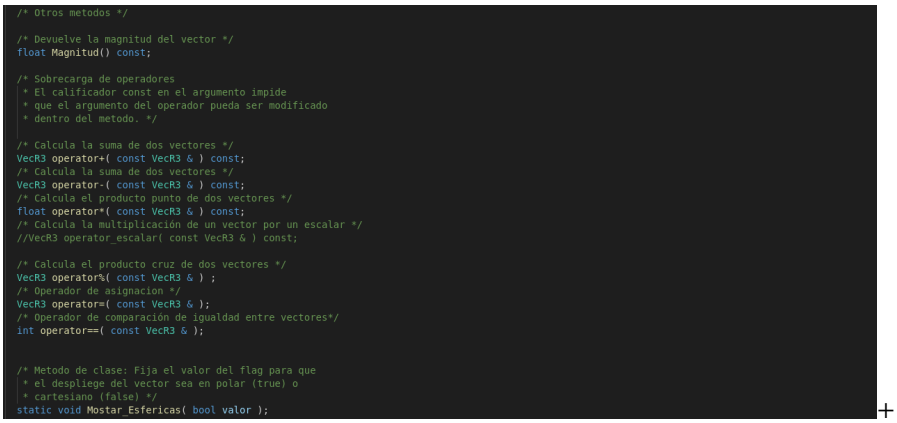
En la cual, se ponen los atributos privados las coordenadas del vector y en modo público secrean los constructores y el destructoy además de todos los métodos, los primero son los métodos «set» y de «get».



```
 * se representa en forma cartesiana. */
class VecR3
{
private:
    /* Coordenada x */
    float Xcor;
    /* Coordenada y */
    float Ycor;
    /* Coordenada z*/
    float Zcor;

    /* Atributo de clase: Indica si el despliege del
     * vector seran en coordenas polares. */
    static bool Esfericas;

public:

    /* Constructor sin argumentos */
    VecR3();
    /* Constructor con argumentos */
    VecR3( float valor_x, float valor_y, float valor_z );
    /* Destructor */
    ~VecR3();

    /* Metodos de asignacion (set methods) */

    /* Asigna la coordenada x */
    void Asignar_x( float valor_x );
    /* Asigna la coordenada y */
    void Asignar_y( float valor_y );
     /* Asigna la coordenada y */
    void Asignar_z( float valor_z );
    /* Asigna las coordenadas x, y */
    void Asignar_xyz( float valor_x, float valor_y, float valor_z );

    /* Metodos de obtencion (get methods)
     * El calificador const al final del prototipo
     * indica que estos metodos no van a alterar
     * los valores de la instancia que los llama. */

    /* Obtener la coordenada x */
    float Obtener_x( ) const;
    /* Obtener la coordenada y */
    float Obtener_y( ) const;
    /* Obtener la coordenada x */
    float Obtener_z( ) const;
```

Figura 1: privado y público

Y la sobre carga de operadores, el operador magnitud devuelva un flotante, el operador de suma, de resta debuelven un vector, el operador * es el produto punto por lo que devuelve un flotante y los operadores de producto cruz y de asignación devuelven un vector y el operador de comparación devuelve un entero.



```
/* Otros metodos */

/* Devuelve la magnitud del vector */
float Magnitud() const;

/* Sobrecarga de operadores
 * El calificador const en el argumento impide
 * que el argumento del operador pueda ser modificado
 * dentro del metodo. */

/* Calcula la suma de dos vectores */
VecR3 operator+( const VecR3 & ) const;
/* Calcula la suma de dos vectores */
VecR3 operator-( const VecR3 & ) const;
/* Calcula el producto punto de dos vectores */
float operator*( const VecR3 & ) const;
/* Calcula la multiplicación de un vector por un escalar */
//VecR3 operator_escalar( const VecR3 & ) const;

/* Calcula el producto cruz de dos vectores */
VecR3 operator%( const VecR3 & ) ;
/* Operador de asignacion */
VecR3 operator=( const VecR3 & );
/* Operador de comparación de igualdad entre vectores*/
int operator==( const VecR3 & );


/* Metodo de clase: Fija el valor del flag para que
 * el despliege del vector sea en polar (true) o
 * cartesiano (false) */
static void Mostar_Esfericas( bool valor );
```

Figura 2: sobrecarga de operadores

Y las funciones amigas que tienen acceso a los elementos privados en este caso a las coordenadas.

```
    /* Funciones amigas */

    /* Despliega un vector con cout */
    friend std::ostream &operator<<( std::ostream &, const VecR3 & );
    /* Multiplica un flotante por un vector */
    friend VecR3 operator*( const float &, const VecR3 & );
    /* Calcula la división de un vector por un escalar */
    friend VecR3 operator/( const float &,  const VecR3 & );

};

#endif /* __VECR3_HPP__ */
```

Figura 3: funciones amigas

Ahora se hace el archivo .cpp que se encuentra con el nombre VecR3.cpp

```
#include "VecR3.hpp"

bool VecR3::Esfericas = false;
VecR3::VecR3( )
{
    Xcor = 0;
    Ycor = 0;
    Zcor=  0;
}
VecR3::VecR3( float valor_x, float valor_y, float valor_z )
{
    Xcor = valor_x;
    Ycor = valor_y;
    Zcor = valor_z;
}
VecR3::~VecR3()
{}
void VecR3::Asignar_x( float valor_x )
{
    Xcor = valor_x;
}
void VecR3::Asignar_y( float valor_y )
{
    Ycor = valor_y;
}
void VecR3::Asignar_z( float valor_z )
{
    Zcor = valor_z;
}
void VecR3::Asignar_xyz( float valor_x, float valor_y, float valor_z )
{
    Xcor = valor_x;
    Ycor = valor_y;
    Zcor = valor_z;
}
float VecR3::Obtener_x( ) const
{
    return Xcor;
}
float VecR3::Obtener_y( ) const
{
    return Ycor;
}
float VecR3::Obtener_z( ) const
{
    return Zcor;
}
```

Figura 4: definir los primeros métodos.

Luego para la magnitd se utiliza la librería «cmath» para utilizar «sqrt», Para sumar y restar vectores se debe utilizar dos vectores, uno de los vectores a restar o sumar es el que llama el operador, por lo que se accede a el por el puntero ”this”. El otro es el que se pasa como argumento. Como se esta dentro de las definiciones de clase, se puede acceder a los atributos privados.

```cpp
float VecR3::Magnitud() const
{
    return std::sqrt( Xcor*Xcor + Ycor*Ycor + Zcor*Zcor);
}

/* Sobrecarga de operadores */

/* Calcula la suma de dos vectores */
VecR3 VecR3::operator+( const VecR3 &avec) const
{
    /* Se declara un vector temporal para almacenar los resultados */
    VecR3 tmp;
    /* Uno de los vectores a sumar es el que llama el operador,
     * por lo que se accede a el por el puntero "this". El otro
     * es el que se pasa como argumento.
     * Como se esta dentro de las definiciones de clase, se puede
     * acceder a los atributos privados. */
    tmp.Xcor = this->Xcor + avec.Xcor;
    tmp.Ycor = this->Ycor + avec.Ycor;
    tmp.Zcor = this->Zcor + avec.Zcor;
    /* Se devuelve la suma de los vectores */
    return tmp;
}

/* Calcula la resta de dos vectores, la resta NO es conmutativa */
VecR3 VecR3::operator-( const VecR3 &avec) const
{
    /* Se declara un vector temporal para almacenar los resultados */
    VecR3 tmp;
    /* Uno de los vectores a restar es el que llama el operador,
     * por lo que se accede a el por el puntero "this". El otro
     * es el que se pasa como argumento.
     * Como se esta dentro de las definiciones de clase, se puede
     * acceder a los atributos privados. */
    tmp.Xcor = this->Xcor - avec.Xcor;
    tmp.Ycor = this->Ycor - avec.Ycor;
    tmp.Zcor = this->Zcor - avec.Zcor;
    /* Se devuelve la suma de los vectores */
    return tmp;
}
```

Figura 5: magnitud, operador suma y resta.

El operador del producto interno también requiere dos vectores también pero devuelve un flotante. Y se multiplica coordenada a coordenada. Y el producto punto también utiliza el puntero this y se define coordenada a coordenada. Y el vector de asignación llama a otro vector y le asigna sus coordenadas al vector que lo llamó.

```cpp
float VecR3::operator*( const VecR3 &avec ) const
{
    /* Ver los comentarios de operator+ */
    float tmp;

    tmp = this->Xcor * avec.Xcor + this->Ycor * avec.Ycor + this ->Zcor*avec.Zcor;

    return tmp;
}

/* Calcula el producto cruz de dos vectores */
VecR3 VecR3::operator%( const VecR3 &avec )
{
    /* Ver los comentarios de operator+ */
    VecR3 tmp;

     tmp.Xcor= this->Ycor * avec.Zcor - this->Zcor * avec.Ycor;
     tmp.Ycor= this->Zcor * avec.Xcor - this->Xcor * avec.Zcor;
     tmp.Zcor= this->Xcor * avec.Ycor - this->Ycor * avec.Xcor;

    return tmp;
}


/* Operador de asignacion */
VecR3 VecR3::operator=( const VecR3 &avec)
{
    /* El vector que llama el operador es el que
     * esta al lado izquierdo de este, y el que
     * esta al lado derecho se pasa como argumento
     * por lo que a "this" se le debe asignar el
     * valor del argumento */
    this->Xcor = avec.Xcor;
    this->Ycor = avec.Ycor;
    this->Zcor = avec.Zcor;

    return (*this);
}
```

Figura 6: producto punto y producto cruz.

El operador de comparación analiza las coordenadas de ambos vectores y retorna un cero si son guales y uno si no son iguales, e imprime si son iguales o no. El método de Mostrar_Esfericas cambia el valor del flag .Esfericas". Y ostream pueden escribir secuencias de caracteres y representar otros tipos de datos. Se proporcionan miembros específicos para realizar estas operaciones de salida. Si el flag Esfericas es true el método Mostrar_Esfericas muestra al vector en coordenadas esféricas.

```cpp
int VecR3::operator==( const VecR3 &avec)
{   if( this ->Xcor == avec.Xcor, this->Ycor == avec.Ycor,this->Zcor == avec.Zcor){


    std::cout << "Son vectores iguales" << std::endl;
    return 0;
    }
    std::cout << "No son iguales" << std::endl;
    return 1;

}


/* Metodo de clase: Fija el valor del flag para que
 * el despliege del vector sea en polar (true) o
 * cartesiano (false) */
void VecR3::Mostar_Esfericas( bool valor )
{
    Esfericas = valor;
}

/* Funciones amigas.
 * Estas funciones fueron declarada como amigas (friend) en la
 * definicion de la clase. Las funciones amigas no son metodos
 * de los objetos, pero pueden acceder a los atributos privados de
 * estos.*/

/* Despliega un vector con cout */
std::ostream &operator<<( std::ostream &salida, const VecR3 &avec )
{
    /* Se decide el tipo de salida en funcion del valor del atributo
     * de clase Polar. */
    if( VecR3::Esfericas )
    {
        /* Se calcula el angulo polar y azimutal del vector. La magnitud
         * se obtiene del metodo ya implementado */
        float theta = std::atan2( avec.Ycor , avec.Xcor );
        float phi=  std::atan2(sqrt(avec.Ycor*avec.Ycor + avec.Xcor*avec.Xcor),avec.Zcor);
        salida << "( " << avec.Magnitud() << " < " << theta <<" < " << phi<<" )";
    }
    else
        salida << "( " << avec.Xcor << ", " << avec.Ycor <<", " << avec.Zcor << " )";

    return salida;
}
```

Figura 7: Otros operadores.

Y por últimos se utiliza las funciones amigas que reciben un escalar y devuelven un vector, pueden acceder a los atributos privados.

```
/* Esta funcion sobrecarga el operador* para permitir
 * la multiplicacion de un escalar por un vector
 * de la forma aesc*avec. Es importante hacer notar
 * que el orden de la multiplicacion NO ES CONMUTATIVO,
 * es decir, esta funcion no es llamada si el orden
 * escalar * vector es invertido. */
VecR3 operator*( const float &aesc, const VecR3 &avec )
{
    VecR3 tmp;
    tmp.Xcor = aesc*avec.Xcor;
    tmp.Ycor = aesc*avec.Ycor;
    tmp.Zcor = aesc*avec.Zcor;

    return tmp;
}



/* Calcula la suma de dos vectores */
VecR3 operator/( const float &aesc, const VecR3 &avec)
{
    VecR3 tmp;
    tmp.Xcor = avec.Xcor/aesc;
    tmp.Ycor = avec.Ycor/aesc;
    tmp.Zcor = avec.Zcor/aesc;
    /* Se devuelve la suma de los vectores */
    return tmp;
}
```

Figura 8: de nuevo funciones amigas.

Se comprueba el uso de estos operadores en otro archivo Prueba.cpp, crean dos vecores y se imprime en consola las salidas de cada operador.

```
#include <cstdlib>
#include <iostream>
#include "VecR3.hpp"

using namespace std;

/* Programa para mostrar el uso de la clase persona.
 * Se crean instancia y llaman metodos para experimentar
 * que pasa. */
int main()
{
    VecR3 K(1,2,3);
    VecR3 Q(2,3,4);

    Q.Mostar_Esfericas(false);


    std::cout <<  Q.operator+(K) << std::endl;
    std::cout <<  Q.operator-(K) << std::endl;
    std::cout <<  Q.operator*(K) << std::endl;
    std::cout <<  Q.operator%(K) << std::endl;
    std::cout <<  operator*(2,K) << std::endl;
    std::cout <<  operator/(2,K) << std::endl;
    Q.Mostar_Esfericas(true);
    Q.operator==(K);
    std::cout <<  Q.operator=(K) << std::endl;

    return 0;
}
```

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE
[rubilin@fedora T1_II_3]$ ./a.out
( 3, 5, 7 )
( 1, 1, 1 )
20
( 1, -2, 1 )
( 1, 2, 3 )
( 2, 4, 6 )
( 0.5, 1, 1.5 )
Son vectores iguales
[rubilin@fedora T1_II_3]$ g++ VecR3.cpp prueba.cpp
[rubilin@fedora T1_II_3]$ ./a.out
( 3, 5, 7 )
( 1, 1, 1 )
20
( 1, -2, 1 )
( 2, 4, 6 )
( 0.5, 1, 1.5 )
No son iguales
( 3.74166 < 1.10715 < 0.640522 )
[rubilin@fedora T1_II_3]$
```

Figura 9: comprobación del funcioamiento de los operadores.