Introducción al cómputo y al desarrollo de software

Clase 4: Clases y objetos. Contenedores. Memoria dinámica.

Lic. Agustín Bernardo & MSc. Rodrigo Bonazzola

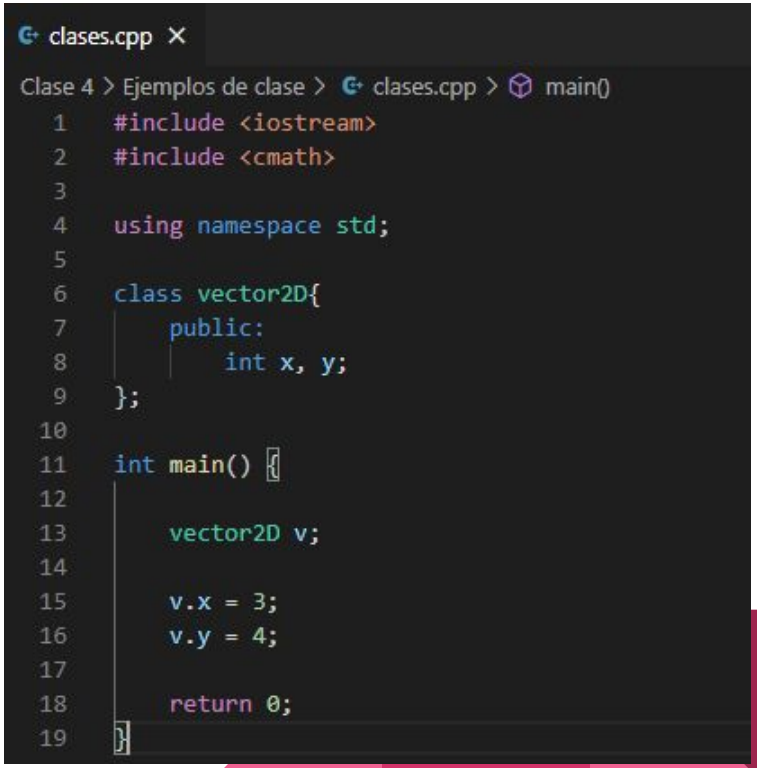
Programación orientada a objetos: Clases

Una clase es una representación abstracta de entidades.

Estas entidades se denominan objetos.

Por pertenecer a una clase, los objetos tienen **propiedades** y **métodos.**

Estos últimos pueden ser privados o públicos.



Clases: Métodos

Los métodos son funciones que pueden ser utilizadas por los objetos.

Estas acceden a todas las propiedades del mismo.

clases.cpp

Clase 4 > Ejemplos de clase > clases.cpp > main()

```cpp
#include <iostream>
#include <cmath>

using namespace std;

class vector2D{
    public:
        float x, y;

        float calculaModulo(){
            return sqrt(x*x+y*y);
        }

};

int main() {

    vector2D v;

    v.x = 3;
    v.y = 4;

    cout<<v.calculaModulo()<<endl;

    return 0;
}
```

Clases: Público y privado

Las propiedades o métodos privados sólo pueden ser usadas desde adentro de la clase.

Esto permite exponer la funcionalidad deseada al usuario.

¿Cómo hacemos para modificar los valores privados?

Usando el identificador “*this*”.

clases.cpp

Clase 4 > Ejemplos de clase > clases.cpp > vector2D > setValues(float, float)

```cpp
#include <iostream>
#include <cmath>

using namespace std;

class vector2D{
    private:
        float x, y;

    public:
        float calculaModulo(){
            return sqrt(x*x+y*y);
        }

        void setValues(float x, float y){
            this->x = x;
            this->y = y;
        }

};

int main() {

    vector2D v;

    v.setValues(3,4);


    cout<<v.calculaModulo()<<endl;

    return 0;
}
```

Clases: Encapsulamiento

Las propiedades o métodos privados sólo pueden ser usadas desde adentro de la clase.

Esto permite exponer la funcionalidad deseada al usuario.

¿Cómo hacemos para modificar los valores privados?

Usando el identificador “*this*”.

clases.cpp

Clase 4 > Ejemplos de clase > clases.cpp > vector2D > setValues(float, float)

```cpp
#include <iostream>
#include <cmath>

using namespace std;

class vector2D{
    private:
        float x, y;

    public:
        float calculaModulo(){
            return sqrt(x*x+y*y);
        }

        void setValues(float x, float y){
            this->x = x;
            this->y = y;
        }

};

int main() {

    vector2D v;

    v.setValues(3,4);


    cout<<v.calculaModulo()<<endl;

    return 0;
}
```

Constructor

Un constructor de clase nos permite crear un objeto inicializando adecuadamente sus propiedades.

Siempre es público y lleva el nombre de la clase.

constructor.cpp

Clase 4 > Ejemplos de clase > constructor.cpp > vector2D > vector2D(float, float)

```cpp
#include <iostream>
#include <cmath>

using namespace std;

class vector2D
{
private:
    float x, y;

public:
    vector2D(float x_const, float y_const)
    {
        x = x_const;
        y = y_const;
    }
    float calculaModulo()
    {
        return sqrt(x * x + y * y);
    }
};

int main()
{

    vector2D v(3.0, 4.0);

    cout << v.calculaModulo() << endl;

    return 0;
}
```

Operadores

Podemos “overlodear” los operadores que pertenecen a la clase.

Podemos hacer que hagan lo que nosotros queremos.

Por ejemplo, que sumen dos vectores.

Se pueden overlodear muchos operadores.

http://www.cplusplus.com/doc/tutorial/templates/

plusOverload.cpp

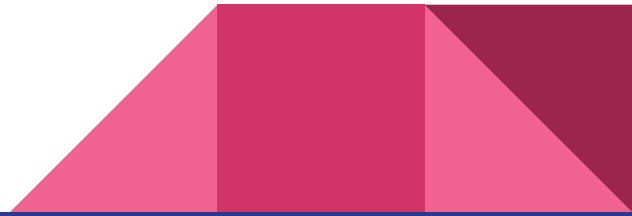
Clase 4 > Ejemplos de clase > plusOverload.cpp > main()

```cpp
#include <iostream>
#include <cmath>

using namespace std;

class vector2D
{
private:
    float x, y;

public:
    vector2D(float x_const, float y_const)
    {
        x = x_const;
        y = y_const;
    }

    vector2D operator+(vector2D other){
        float x_res = this->x + other.x;
        float y_res = this->y + other.y;

        return vector2D(x_res, y_res);
    }
};

int main()
{

    vector2D v(3.0, 4.0);
    vector2D u(4.0, 3.0);

    vector2D w = v+u;

    return 0;
}
```

Ejercicio

- **Cree una clase «vector3D» que permita sumar dos vectores y multiplicarlos por un número. Además, debe permitir imprimir el vector.**

- **Cree una clase sistema, que acepte un arreglo de fuerzas y posiciones y permita encontrar la fuerza resultante y el torque para algún punto.**

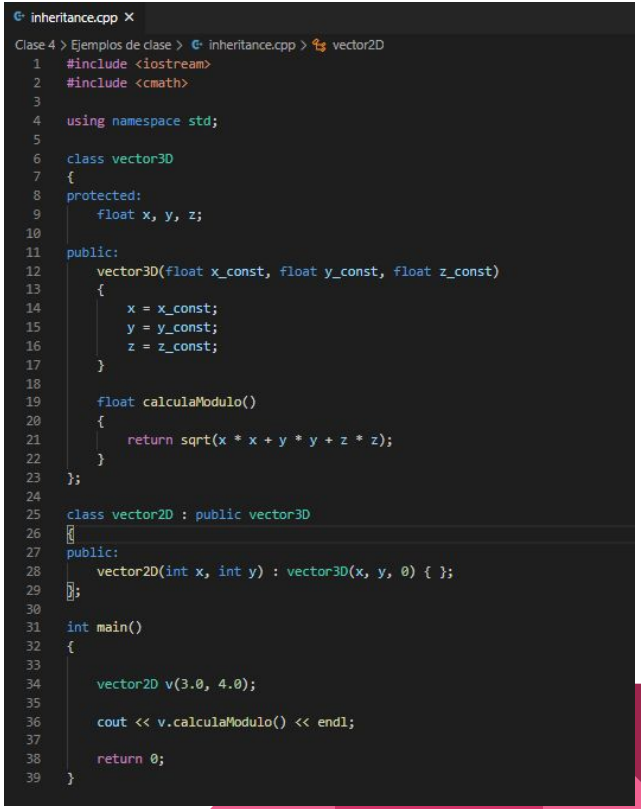


Herencia de clases

Se puede crear una clase que *herede* de otra clase.

Así, podemos empezar desde estructuras muy abstractas y llegar a objetos simples que representen nuestro problema.

http://www.cplusplus.com/doc/tutorial/templates/



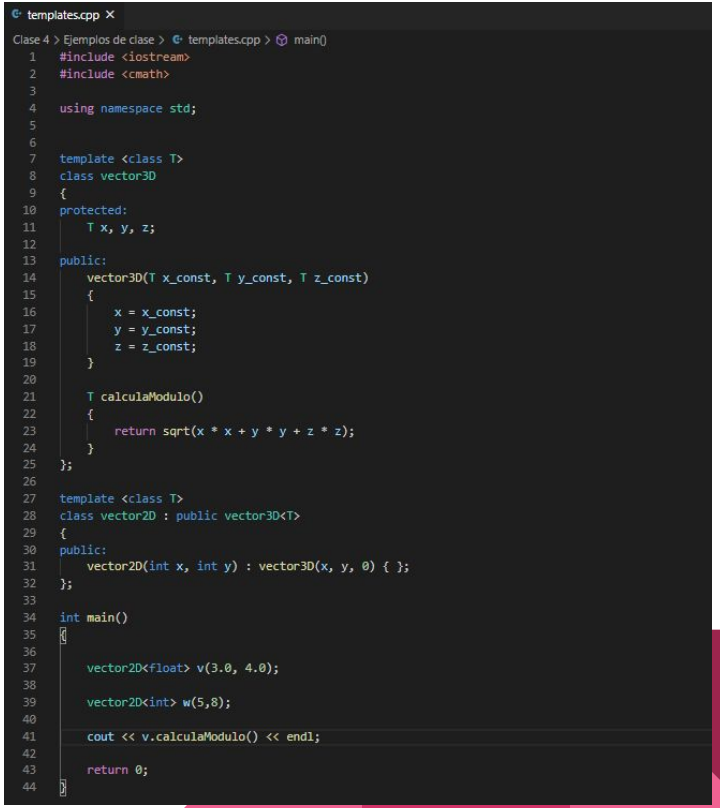
Templates

¿Qué pasa si no nos alcanza la precisión float?

¿O si queremos ahorrar espacio y usamos ints?

Podemos usar templates.

http://www.cplusplus.com/doc/tutorial/templates/



Friendship

Cuando una función de una clase debe acceder a los miembros privados de otra, se la declara como función “amiga”.

Así podemos modificar el operador << de cout para imprimir lo que queramos de nuestro objeto.

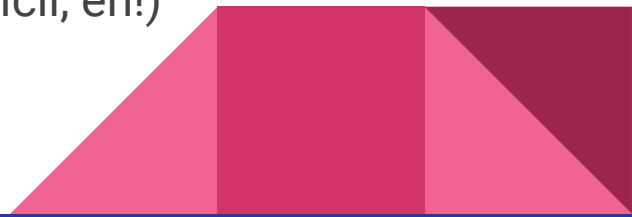
friendship.cpp

Clase 4 > Ejemplos de clase > friendship.cpp > vector3D<T>

```cpp
#include <iostream>
#include <cmath>

using namespace std;


template <class T>
class vector3D
{
protected:
    T x, y, z;

public:
    vector3D(T x_const, T y_const, T z_const)
    {
        x = x_const;
        y = y_const;
        z = z_const;
    }

    T calculaModulo()
    {
        return sqrt(x * x + y * y + z * z);
    }

    friend ostream& operator<<(ostream& os, const vector3D& vector)
    {
        os <<"(" <<vector.x<<","<<vector.y<<","<<vector.z<<")";
    }
};
```

Ejercicios

- Crear una clase Cuadrilátero, que nos permita dar cuatro lados.
- Crear una clase rectángulo, que herede de cuadrilátero y permita calcular perímetro y área.
- Crear una clase cuadrado, que herede de rectángulo y permita obtener su perímetro y área. Cuando imprimamos el cuadrado, que lo muestre.
- Utilizar la clase vector2D para crear la una clase “polígono”, que permita obtener el perímetro del mismo (¡difícil, eh!)

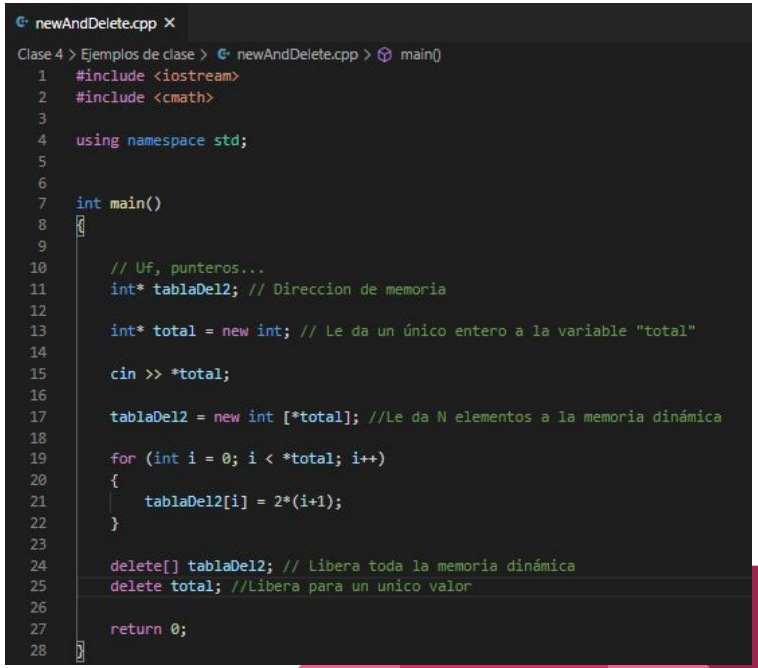


Memoria dinámica

¿Cómo hacemos para crear un vector de elementos con una cantidad que decidamos mientras el programa va andando?

¿Cómo manejamos la memoria?

Con new, new[], delete y delete[].

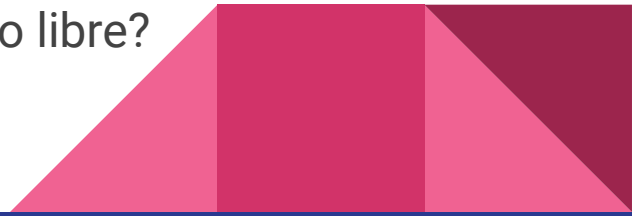


Ejercicio

- Cree, en la memoria dinámica, un arreglo de N vectores de números flotantes 2D.

- Calcule su suma.

- ¿Será eso un diagrama de cuerpo libre?



Contenedores: la librería estándar.

Así como hoy creamos la clase vector2D y vector3D, existen clases que vienen con C++ que ya resuelven estos problemas.

vector<>, array<>, map<>, deque<>, etcétera.

Hablaremos de ella en los ejercicios.

http://www.cplusplus.com/reference/stl/

Container class templates

Sequence containers:

array C++11	Array class (class template)
vector	Vector (class template)
deque	Double ended queue (class template)
forward_list C++11	Forward list (class template)
list	List (class template)

Container adaptors:

stack	LIFO stack (class template)
queue	FIFO queue (class template)
priority_queue	Priority queue (class template)

Associative containers:

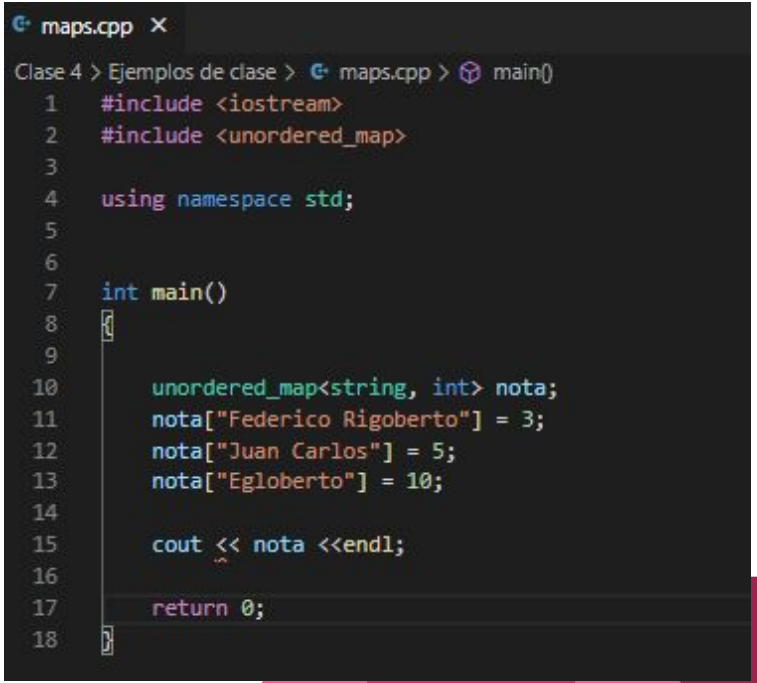
set	Set (class template)
multiset	Multiple-key set (class template)
map	Map (class template)
multimap	Multiple-key map (class template)

Unordered associative containers:

unordered_set C++11	Unordered Set (class template)
unordered_multiset C++11	Unordered Multiset (class template)
unordered_map C++11	Unordered Map (class template)
unordered_multimap C++11	Unordered Multimap (class template)

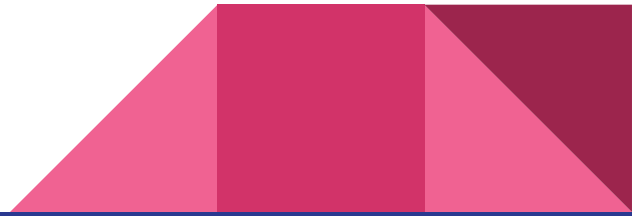
Mapas

Los mapas son «tablas de Hash» que hacen las veces de vector, pero podemos entrar con una palabra (key) y sacar un valor (value).



Ejercicios

- Utilizando la librería vector<> de C++, rehaga el ejercicio de suma de la suma de N vectores.
- Utilice un mapa desordenado para guardar la nota de cada alumno de este curso.
- Escriba una cola de supermercado que dé prioridad a embarazadas.



Más ejercicios en la guía.

¿Preguntas?