C++ V

-
- Orientación a objetos
  - Objetos
  - Clases
    - Puntero this
  - Métodos
    - Constructor
    - Polimorfismo
  - Atributos
  - Encapsulación
  - Instanciación
  - **Herencia**
    - **Clases Abstractas**

- **Templates**
- **Sobrecarga de operadores**

Herencia

La herencia es un mecanismo fundamental en la programación orientada a objetos que permite crear una nueva clase a partir de una clase existente, llamada clase base o superclase. La clase creada a partir de la superclase se denomina clase derivada o subclase, y hereda todas las propiedades y métodos de la clase base, además de poder añadir nuevos miembros o métodos específicos.

En la herencia, la clase base se utiliza como plantilla para la creación de la subclase, y ésta puede añadir nuevas propiedades o métodos, o bien modificar o sobrescribir los existentes. Además, la subclase también puede heredar de su superclase las propiedades y métodos públicos y protegidos, que se pueden utilizar en la subclase de la misma manera que en la superclase.

La herencia es un concepto importante en la programación orientada a objetos, ya que permite la creación de clases con características comunes que se pueden reutilizar en diferentes contextos, evitando la duplicación de código y mejorando la modularidad y la flexibilidad del diseño.

https://github.com/jmejiagu/BigData-UdeARootBasicCurse-20231/blob/main/codes/Unidad1/mod/Star.h

Herencia

- **Herencia pública:** Se refiere a la herencia en la que todos los miembros públicos y protegidos de la clase base conservan esos mismos niveles de acceso respectivamente en las clases derivadas.

```
class ClaseDerivada : public ClaseBase
```

- **Herencia protegida:** Se refiere a la herencia en la que todos los miembros públicos de la clase base adquieren el nivel de acceso protegido en las clases derivadas, mientras que los miembros protegidos conservan su nivel de acceso. Lo anterior indica que una clase derivada puede luego heredar a otra clase los miembros protegidos que heredó de su clase base.

```
class ClaseDerivada : protected ClaseBase
```

- **Herencia privada:** Se refiere a la herencia en la que todos los miembros públicos y protegidos de la clase base adquieren el nivel de acceso privado en las clases derivadas. De ahí se desprende que una clase derivada que haya heredado mediante herencia privada no puede heredar a otras clases los miembros que ha heredado de otras clases.

```
class ClaseDerivada : private ClaseBase
```

Tomado de:
https://www.codingame.com/playgrounds/50747/herencia-en-c-practica-3/concepto-de-herencia

Sobrecarga de operadores

- Se puede sobrecargar operadores en c++ y definir el comportamiento al usarlos entre objetos
- Para la sobrecarga se usa la palabra clave **operator** seguido del operador a sobre cargar.
- Parecido a una función, tengo que definir el parámetro que recibo y el valor que retorno.

```
Vector& operator=(const Vector& v) {
    if (this != &v) {
        if (size != v.size) {
            delete[] data;
            size = v.size;
            data = new double[size];
        }
        for (int i = 0; i < size; i++) {
            data[i] = v.data[i];
        }
    }
    return *this;
}

Vector operator+(const Vector& v) const {
    Vector w(size);
    for (int i = 0; i < size; i++) {
        w.data[i] = data[i] + v.data[i];
    }
    return w;
}

Vector operator-(const Vector& v) const {
    Vector w(size);
    for (int i = 0; i < size; i++) {
        w.data[i] = data[i] - v.data[i];
    }
    return w;
}
```

Templates

- En C++, los templates son una característica que permite escribir código genérico que puede ser utilizado con diferentes tipos de datos, sin necesidad de escribir código específico para cada uno de ellos. En otras palabras, los templates permiten definir clases y funciones que pueden aceptar tipos de datos genéricos como parámetros, lo que los hace extremadamente flexibles y reutilizables.
- Por ejemplo, un vector es una estructura de datos que puede almacenar un conjunto de elementos de un tipo determinado, como enteros o cadenas. Sin embargo, es posible que necesitemos almacenar elementos de diferentes tipos, como enteros, cadenas o cualquier otro tipo de datos. En este caso, podemos utilizar plantillas para crear un vector genérico que pueda ser utilizado con cualquier tipo de datos.

https://github.com/jmejiagu/BigData-UdeARootBasicCurse-20231/blob/main/codes/Unidad1/template/VectorT.cxx