* Los miembros de una clase se utilizan mediante el operador **punto**, en con nombre del objeto de la clase, o a través del operador **flecha** en conjunción con un puntero a un objeto de la clase.
* Los miembros de una clase que se especifican como **private** están para las funciones miembro de la clase y sus funciones amigas.
* Los miembros de una clase que se especifican como **public** están accesibles en cualquier parte en la que un objeto de la clase se encuentro dentro del alcance.
* La **asignación predeterminada a nivel de miembros** (realizada por el operador de asignación, =) se puede utilizar para asignar un objeto de la clase a otro objeto de la misma clase.
* La **inicialización de miembros** se debe usar para inicializar los miembros constantes de una clase.
* Una función no miembro se debe declarar como **friend** de una clase para tener acceso a los datos miembro **private** de esa clase.
* El operador **new** asigna memoria en forma dinámica para un objeto de un tipo especicado, y se duevuelve un **puntero** a ese tipo.
* Un objeto constante debe **inicializarse**, no se puede modificar después de crearlo.
* Un miembro de datos **static** representa la información a nivel de clase.
* Las funciones miembro no static de un objeto tienen acceso a un “auto puntero” al objeto, conocido como puntero **this**.
* La palabra clave **const** especifica que un objeto o variable no puede modificarse después de inicializarlo.
* Si no se proporciona un inicializador de miembros para un objeto miembro de una clase, se hace una llamada al **constructor predeterminado**.
* Una función miembro debe declararse static si no accede a los datos miembro **no static** de una clase.
* Los objetos miembro se construyen **antes** del objeto de su clase circundante.
* El operador **delete** reclama la memoria previamente asignada por new.
* Suponga que A y B son variables enteras y que formamos la suma A+B, ahora suponga que C y D son variables de punto flotante y formamos la suma C+D, los dos operadores + de aquí se están utilizando claramente para distintos fines. Este es un ejemplo de **sobrecarga de operadores**.
* La palabra clave **operator** introduce la definición de una función de operador sobrecargado.
* Para usar operadores en objetos de clases, estos deben sobrecargarse con la excepción de los operadores **asignación (=)**, **dirección (&)** y **coma (,)**.
* La **precedencia** (qué operador se evalúa antes), **asociatividad** y **aridad** (operadores binarios para que actúen como unarios o viceversa) de un operador no se puede modificar al sobrecargar ese operador.
* La **herencia** es una forma de reutilización de software en la que nuevas clases absorben los datos y comportamientos de las clases existentes y aportan a estas clases nuevas capacidades.
* Los miembros **protected** de una clase base pueden ser utilizados sólo en la definición de la clase base o en las definiciones de la clase derivada.
* En una relación **“es un”** o  **de herencia** un objeto de una clase privada se puede tratar también como un objeto de su clase base.
* En una relación **“tiene un”** o **composición** o **agregación** el objeto de una clase tiene uno o más objetos de otras clases como miembros.
* En la herencia simple, una clase existe en una relación **jerárquica** con sus clases derivadas.
* Los miembros **public** de una clase base se pueden utilizar dentro de una clase base y en cualquier clase parte donde el programa tenga un puntero de esa clase o a un objeto de sus clases derivadas.
* Los miembros de acceso protected de una clase base tienen un nivel de protección entre los de acceso public y **private.**
* C++ cuanta con **herencia múltiple**, la cual permite a una clase derivada heredar de muchas clases base, incluso aunque las clases base no estén relacionadas.
* Cuando se instancia un objeto de una clase derivada, el **constructor** de la clase base se llama de manera implícita o explícita para realizar la inicialización necesaria de los datos miembro de la clase base en el objeto de la clase derivada.
* Al derivar una clase de una clase base con herencia public, los miembros public de la clase base se convierten en miembros **public** de la clase derivada y los miembros protected de la clase base se convierten en miembros **protected** de la clase derivada.
* Al derivar una clase de una clase base con herencia protected, los miembrospublic de la clase base se convierten en miembros **protected** de la clase derivada y los miembros protected de la clase base se convierten en miembros **protected** de la clase derivada.
* Tratar a un objeto de la clase base como un **objeto de la clase derivada** puede producir errores lógicos.
* El polimorfismo ayuda a eliminar la lógica de **switch**.
* Si una clase contiene al menos una función virtual pura, es una clase **abstracta**.
* Las clases a partir de las cuales pueden instanciarse objetos se llaman **concretas**.
* El operador **dynamic\_cast** se puede usar para realizar conversiones descendentes con los punteros de la clase base en forma segura.
* El operador typeid devuelve la dependencia de un objeto **type\_info**.
* El **polimorfismo** implica el uso de un puntero o referencia de la clase base para invocar funciones virtuales en objetos de la clase base y clase derivada.
* Las funciones que pueden sobrecargarse se declaran mediante la palabra clave **virtual**.
* Al proceso de convertir un puntero de la clase base en un puntero de la clase derivada se la conoce como **conversión descendente**.

Verdadero/Falso

* Los constructores de la clase base no son heredados por las clases derivadas.

**Verdadero.**

* Una relación “tiene un” se implementa mediante herencia.

**Falso, se implemente mediando la composición, una relación “es un” se implementa mediante herencia.**

* Una clase auto tiene una relación “es un” con las clases volante, dirección y frenos.

**Falso, es un ejemplo de una relación “tiene un”.**

* La herencia fomenta la reutilización de software comprobado, de alta calidad.

**Verdadero.**

* Cuando se destruye un objeto de la clase derivada, los destructores se llaman en el orden inverso al de los constructores.

**Verdadero.**

* Todas las funciones virtuales en una clase base abstracta se deben declarar como funciones virtuales puras.

**Falso, una clase abstracta puede incluir funciones virtuales en implementaciones.**

* Es peligroso tratar de hacer referencia a un objeto de la clase derivada con un manejador de la clase base.

**Falso, es peligroso hacer referencia a un objeto de la clase base con un manejador de la clase derivada.**

* Para hacer una clase abstracta, se declara como virtual.

**Falso, las clases nunca se declaran virtual, en vez de ello una se hace abstracta al definir por lo menos una función virtual pura.**

* Si una clase base declara una función virtual, una clase derivada debe implementar la función para convertirse en una clase concreta.

**Verdadero.**

* La programación polimórfica puede eliminar la necesidad de lógica de switch.

**Verdadero.**

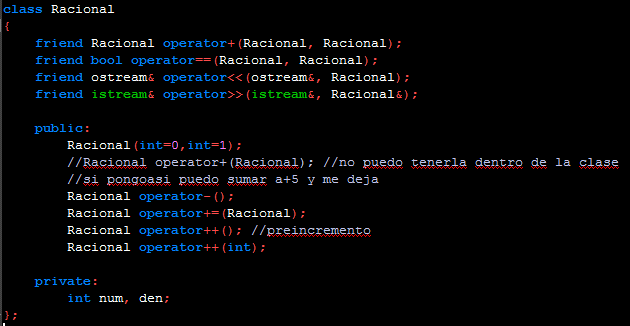
Manejo de excepciones

Una **excepción** es una indicación de un problema que ocurre durante la ejecución de un programa.

El manejo de excepciones nos permite crear programas que puedan resolver los problemas que ocurren en tiempo de ejecución, por lo general esto permite a los programas continuar su ejecución como si no se hubiera encontrado ningún problema. Los problemas más graves pueden requerir que el programa notifique al usuario acerca del problema, antes de terminar alguna manera controlada.

Sobrecarga de operadores

Racional.h



funciones.cpp

