Programación y Estructuras de Datos

SEMINARIO C++

Guía rápida

SEMINARIO - SESIÓN 4

"C++ paso a paso"
Capítulo 5 : "Sobrecarga de operadores"

- **5.2 Puntero THIS**

  - Fíjate en las características básicas del puntero THIS : devuelve una instancia del propio objeto de clase y es de sólo lectura (pág. 83)

- **5.3 Modificador CONST**

  - Fíjate muy bien en este punto, ya que el papel de CONST no siempre es entendido, y se convierte en básico en nuestras prácticas : a menudo se te va a presentar la duda de si usarlo o no, de dónde usarlo y de cómo usarlo para evitar errores de ejecución .

  - Fíjate en el comentario : "Para indicar que un argumento de una función NO se va a modificar" (pág. 84). Por tanto, observa que es diferente declarar un método así :

    PRIMER CASO

```
ostream& operator<<(ostream &s, TCoordenada &obj)
        {......}
```

    ... que declararlo así :

    SEGUNDO CASO

```
ostream& operator<<(ostream &s, const TCoordenada &obj)
        {......}
```

    En el PRIMER CASO : en cualquier **main.cc**, el parámetro "obj" NO podrá ser informado por un valor constante o que otro método haya devuelto por VALOR.
    Ejemplo :

```
(. . . . .)
TCoordenada a, b;
cout << a+b ; // → DARÁ ERROR
(. . . . .)
```

    (la suma de 2 items **TCoordenada** se devuelve por valor, comprueba la declaración en el código de la **página 235**, líneas **197-198**)

    En el SEGUNDO CASO : en cualquier **main.cc**, el parámetro "obj" SÍ podrá ser informado por un valor constante o que otro método haya devuelto por VALOR, ya que "obj" ha sido declarado CONST . Vale el mismo ejemplo :

```
(. . . . .)
TCoordenada a, b;
cout << a+b ; // → NO DARÁ ERROR
(. . . . .)
```

  - Ahora fíjate en el párrafo que dice : "Un objeto constante no puede ser modificado …. Los métodos constantes sí que pueden invocar sobre objetos constantes" (pág. 84). Mira cómo se declara un método CONST.
    Así que, siguiendo el ejemplo que te he puesto antes, en este método, que trata con un parámetro de entrada CONST, se puede invocar a otros métodos :

```
ostream& operator<<(ostream &s, const TCoordenada &obj)
{
......
        obj.metodoClase1(. . .);
......
        obj.metodoClase2(. . .);
......

}
```

… Pero, ¡OJO!, al haber declarado "obj" como CONST te crea una nueva servidumbre : el compilador te obligará a declarar como CONST todos y cada uno de los métodos invocados en "`operator<<`" (**`metodoClase1 y metodoClase2,`** este caso) . Haz esto de la manera en que aparece el primer ejemplo de la pág. 84 (líneas 2 y 6, método "algo").

- **5.4 Paso por referencia**
  - Fíjate también muy bien en este punto, ya que el papel de "&" también se convierte en básico en nuestras prácticas.
  - Fíjate en el comentario sobre el iniciador de referencia (pág. 85), y en su ejemplo asociado.

    Fíjate en el comentario : "Si una función tiene un argumento por referencia, no se podrá invocar con un argumento constante o con un valor de retorno POR VALOR", y en su **ejemplo 5.3** asociado (pág. 86). Es un caso similar al que te he puesto antes , recuerda :

```
ostream& operator<<(ostream &s, TCoordenada &obj) // → NO hay CONST

(. . . . .)

TCoordenada a, b;
cout << a+b ; // → DARÁ ERROR
```

- **5.5 Sobrecarga de operadores**
  - Fíjate en el significado de invocar a una sobrecarga de operador (pág. 88) :

    `a = b + c` equivale a `a.operator=(b.operator+(c))`

- **5.6 Restricciones al sobrecargar un operador**
  - Fíjate especialmente en los operadores NO sobrecargables : '.', '.*', '::' , '?:' (pág. 88) :

- **5.7 Función miembro o función no miembro**
  - Fíjate especialmente en qué distingue a las funciones miembro (el operando izquierdo es de la clase) , de las funciones NO miembro (el operando izquierdo puede ser de FUERA de la clase) (pág. 89).
  - Fíjate que las funciones NO miembro pueden tener que definirse como **friend ,** (ya vimos sus características en la Sesión anterior) , para acceder a las variables de la parte PRIVATE (pág. 89).

- **5.8 Consejos**
  - Fíjate en TODOS los puntos de este apartado : son todos muy útiles (pág. 89).
  - Ejemplo : para implementar el "operator<<" de cualquier clase, usarás el punto 1.
  - Ejemplo : para implementar el "operator=" de cualquier clase, usarás el punto 3.
  - Ejemplo : para implementar el "operator+" de cualquier clase, usarás el punto 4.
  - Ejemplo : para implementar el "operator[]" de cualquier clase de tipo lineal VECTOR, usarás el punto 6. En este caso, échale ya un vistazo por adelantado al apartado **5.16** de este capítulo , que explica el OPERADOR CORCHETE.

- **5.9 Operador asignación**

    - Fíjate en los dos temas principales de la asignación : sobrecargar siempre la asignación y declarar el parámetro por REFERENCIA (pág. 90).

    - Teclea y compila el **ejemplo 5.7** (añadiendo a **tcoordenada.h** y a **tcoordenada.cc**) ; este ejemplo es más completo que el **ejemplo 5.4 ,** y se explica en el comentario de la autoasignación (pág. 92)

    - Teclea y compila el MAIN del **ejemplo 5.5** y el MAIN del **ejemplo 5.6.** Comprueba resultados y salidas**.**

- **5.10 Constructor de copia y operador asignación**

    - Fíjate en el tema del Constructor de copia respecto a la asignación (pág. 93).

    - Teclea y compila el **ejemplo 5.9** y el **ejemplo 5.10** (añadiendo, respectivamente, a **tcoordenada.h** y a **tcoordenada.cc**) (pág. 94).

- **5.11 Operadores aritméticos**

    - Teclea y compila el **ejemplo 5.11** y el **ejemplo 5.12** (añadiendo, respectivamente, a **tcoordenada.h** y a **tcoordenada.cc**) , complementándolo con la prueba del MAIN del **ejemplo 5.13**.

    - Fíjate en lo que produce el ERROR : es la **línea 18** del **ejemplo 5.13.** La explicación viene en la página 97**.** Volvemos a un error similar al que te he planteado antes , lo que pasa es que ahora el error lo da el "operator=", y antes lo daba el "operator<<" . Recuerda :

```
ostream& operator<<(ostream &s, TCoordenada &obj) // → NO hay CONST
(. . . . .)
TCoordenada a, b;
cout << a+b ; // → DARÁ ERROR
```

    - Subsana el error mediante el **ejemplo 5.14** y compila**.**

- **5.12 Operadores incremento y decremento**

    - Teclea y compila el **ejemplo 5.15 (**PRE-incremento**)** y el **ejemplo 5.16 (**POST-incremento**)** añadiendo a **tcoordenada.h** y a **tcoordenada.cc** en ambos casos.

- **5.13 Operadores abreviados**

    - Teclea y compila el **ejemplo 5.17** ("operator+=", "operator-=") añadiendo a **tcoordenada.h** y a **tcoordenada.cc**.

- **5.14 Operadores comparación**

    - Teclea y compila el **ejemplo 5.18 (**"operator=="**)** y el **ejemplo 5.19 (**"operator!="**)** añadiendo a **tcoordenada.h** y a **tcoordenada.cc** en ambos casos.

- **5.15 Operadores entrada / salida**

    - Fíjate en su carácter de funciones **friend** (pág. 101).

    - Teclea y compila el **ejemplo 5.20 (**"operator>>"**)** y el **ejemplo 5.21 (**"operator<<"**)** añadiendo a **tcoordenada.h** y a **tcoordenada.cc** en ambos casos.

    - Recuerda una vez más mi comentario anterior, porque viene al caso :

```
ostream& operator<<(ostream &s, CONST TCoordenada &obj) // → Hay CONST

(. . . . .)
TCoordenada a, b;
cout << a+b ; // → NO DARÁ ERROR
(. . . . .)
```

- **5.16 Operador corchete**

    - Fíjate en su carácter bivalente : como operando de LECTURA / de LECTURA-ESCRITURA, **ejemplo 5.22** (pág.103)

    - Teclea y compila el **ejemplo 5.22** y el **ejemplo 5.23** (añadiendo a **tcoordenada.h)** y el **ejemplo 5.24** a (añadiendo a **tcoordenada.cc**) , complementándolo con la prueba del MAIN del **ejemplo 5.25**.

- **5.18 Ejercicios de PROGRAMACIÓN / 5.20 Respuesta a ejercicios de PROGRAMACIÓN**

    - El **ejemplo 5.18.4** propone la clase **TCalendario** .

    - La **solución 5.20.4** resuelve inicialmente lo que se pide de la clase **TCalendario .**