C++
me gusta++

Dr. Arno Formella
http://www.ei.uvigo.es/~formella
Universidad de Vigo
Departamento de Informática
E-32004 Ourense

21 de febrero de 2003

Índice

1. Clases

Ejemplo: `c0.cpp`

> Consejo: Definimos para cada clase siempre 4 métodos: 1. por lo menos un constructor, 2. el constructor de copia, 3. el destructor, 4. el operator de asignación !

Todos los miembros y métodos de una clase se puede declarar como privado (`private`), es decir, visto solamente dentro de la clase, protejido (`protected`), es decir, visto sólo en la jeraquía de las clases, y público (`public`), es decir, visto desde fuera de la clase.

> Consejo: Si no queremos definir algunos de dichos métodos, los declaramos como privados !

2. Punteros y referencias

Cada variable, o mejor decir, objeto (datos o funciones) está colocado dentro de la memoria que puede acceder el programa. Punteros no son nada mas que indirecciones hacía el comienzo de dicha memoria.
Para obtener la direccion de un objeto se coloca una `&` delante de un objeto, para obtener el objeto al cual apunto un puntero, se coloca un `*` delante del puntero.
Ejemplo: `c1.cpp`
Como puntero *a nada* se usa 0.

> Consejo: Se intenta siempre definir un puntero en su declaración!

3. Const

> Consejo: Usamos `const` siempre cuando podemos, es decir, por defecto!

¿Dónde se puede usar constantes?

- constantes (en vez de `defines`)
- métodos de clases
- parámetros

El compilador evita usar memoria para los constantes, intenta usar el valor directo durante la generación de código. Eso implica que los constantes son locales para el fichero y no se puede usar desde otro fichero (pero permite entonces definir constantes en ficheros `.h`.
Excepción: se declara la constante con `extern` y exactamente una vez se define como tal.
Para objetos complejos se usa memoria, pero no se puede cambiar, bueno, casi, pero...
El compilador no inventa un constructor por defecto para objetos que se construye de forma constante!
Se puede usar el modificador `mutable` con miembros de una clase para permitir que

- un método constante puede cambiar dicho miembro
- un objeto constante de la clase puede ser cambiado en dicho miembro

Ejemplo: `c2.cpp`
Ejemplo: `c4.cpp`

> Consejo: Aprovechamos de `const Type&` siempre cuando podemos, es decir, por defecto!

Una constante tiene que ser definida durante su primer uso, entonces hay que definirla durante su declaración o durante el constructor si se trata de un miembro de una clase (Ojo: el compilador no detecta si se nos ha olvidado inicializar una constante dentro de una clase!).

> Consejo: Para uniformizar usamos siempre el constructor si inicializamos una variable, y el = si se trata de un puntero !

4. Sobrecargar Funciones

Una función o método no solo se destingue por su nombre, sino también por los tipos de sus parámetros (y de su modificador `const` en caso de un método).
Ejemplo: `c6.cpp`

> Consejo: Sabemos que C++ convierte los tipos intrínsicos implícitamente!

Ejemplo: `c3.cpp`
Podemos sobrecargar funciones con plantillas.
Ejemplo: `c7.cpp`

5. Operadores

C++ permite de definir el comportamiento de casi todos los operadores para las clases (excepciones son `.` (acceso a miembro), `*` (dereferencia de puntero) y `?:` (selección condicional).
Lo vemos en una clase de vector muy simple:
Ejemplo: `c5.cpp`
Los métodos `friend` siempre son públicos.
`this` es un puntero disponible que apunta al propio objeto.

> Consejo: Hay que considerar el caso de asignar un objeto a si mismo cuando se implementa el operador =!

6. Herencia y Polimorfismo

Heredemos de una clase base para aumentar su funcionamiento, y aprovechamos de un puntero a la clase base para acceder a los objetos derivados con tipos diferentes pero con la misma interfaz.
Ejemplo: `c9.cpp`

> Consejo: Cuando una clase tiene un método virtual se declara también el destructor virtual!

Aumentamos nuestro ejemplo con una nueva forma (triángulo relleno) que se deriva de un triángulo ya definido.
Ejemplo: `c12.cpp`

7. Leda

Leda es una librería comercial que proporciona un amplio conjunto de estructuras de datos y algoritmos. Destacan los algoritmos sobre grafos y los tipos númericos exactos (p.e. números racionales, es decir, las fracciones).
Ejemplo: miramos el manual de Leda.
Un simple ejemplo usando una lista de Leda para trabajar con enteros:
Ejemplo: `c10.cpp`
Como se ve, es fácil cambiar el tipo de datos que se usa dentro de la lista.

8. gsl

gsl es una librería en C bajo licencia de GNU (es decir, hay que añadir el código fuente al programa de forma gratiuta, si se distribuye el programa en cualquier forma) que proporciona un amplio conjunto de funciones matemáticas incluyendo vectores y matrices (p.e. soporte de las funciones BLAS).
Ejemplo: miramos el manual de gsl.
Queremos aprovechar de dicha librería para nuestro desarrollo en C++ (es decir, usamos lo bueno que ya hay!) y por eso envolvemos las “estructuras de datos” de gsl en propias clases para acceder a su funcionalidad con la expresividad de C++.
Un simple ejemplo implementando una clase `Matrix` que usa internamente los matrices de gsl:
Ejemplo: `c11.cpp`
Ejemplo: `Matrix.h`

> Consejo: Reusamos código que nos parece adecuado para resolver el problema propuesto!

> Consejo: No “reinventamos la rueda” de nuevo, sino ponemos los “pneumáticos” cómodos alrededor!

9. Bibliografía

- Bruce Eckel. *Thinking in C++*. Addison Wesley.

10. Enlaces

- la página principal de gsl:
  `http://sources.redhat.com/gsl`
- `la página principal de Leda:`
  `http://www.algorithmic-solutions.de/as_html/products/products.html`
- `la página principal de una librería muy amplia para trabajar com`
  `imágenes: http://vxl.sourceforge.net/`