

Clases en C++ - Const y Member Initialization List

Di Paola Martín
martinp.dipaola <at> gmail.com

Facultad de Ingeniería
Universidad de Buenos Aires

Extra footage

Constantes

1

Métodos constantes: no modifican al objeto

```
1 struct Vector {  
2     int *data;  
3     int size;  
4  
5     void set(int pos, int val) {  
6         this->data[pos] = val;  
7     }  
8  
9     int get(int pos) const {  
10        return this->data[pos];  
11    }  
12  
13    /* ... */  
14};
```

2

- Un método constante es un método que no modifica el estado interno del objeto. Esto es, no cambia ningún atributo ni llama a ningún método salvo que este sea también constante.
- Sirve para detectar errores en el código en tiempo de compilación: si un método no modifica el estado debería poderse ponerle la keyword `const`; si el compilador falla es por que hay un bug en el código y nuestra hipótesis de que el método no cambiaba el estado interno del objeto es errónea.

Objetos constantes

```
17 void f() {  
18     Vector v(5);  
19  
20     v.set(0, 1); // no const  
21     v.get(0); // const  
22 }  
  
24 void f() {  
25     const Vector v(5); // objeto constante  
26  
27     v.set(0, 1); // no const  
28     v.get(0); // const  
29 }
```

3

Const como promesa

```
17 void f() {  
18     Vector v(5);  
19  
20     g(v);  
21 }  
22  
23 void g(const Vector &v) {  
24     v.set(0, 1); // no const  
25     v.get(0); // const  
26 }
```

4

- Es comun recibir parámetros constantes. La función promete que no va a cambiar al objeto recibido como parámetro.

Atributos constantes

```

1 struct Vector {
2     int * const data; // no confundir con int const * data;
3     const int size; // equivalente a int const size;
4
5     void set(int pos, int val) {
6         this->data[pos] = val;
7     }
8
9     int get(int pos) const {
10         return this->data[pos];
11     }
12
13     /* ... */
14 };

```

5

- También podemos tener atributos constantes. Estos toman un valor cuando se crean y lo mantienen durante toda la vida del objeto.
- Pequeña aclaración: `const int` y `int const` son equivalentes así como también `const int *` y `int const *`. Sin embargo es distinto `int * const`. Confuso?
- `const int * p` se lee como "p es un puntero; a `int`; constante" mientras que `int * const p` se lee como "p es constante; puntero; a `int`". El primero apunta a `ints` constantes mientras que el segundo es el puntero quien es constante.

Extra footage

Member Initialization List

6

Member Initialization List

```

1 struct Vector {
2     int *data;
3     int size;
4
5     Vector(int size) {
6         // atributos ya contruidos; aca solo los re-asigno
7         this->data = malloc(size*sizeof(int));
8         this->size = size;
9     }
10
11 struct Vector {
12     int *data;
13     int size;
14
15     Vector(int size) : data(malloc(size*sizeof(int))),
16                       size(size) {
17
18     }
19 }

```

7

- Al ejecutarse el cuerpo del constructor todos sus atributos ya estan creados.
- Si se necesita construir alguno o todos sus atributos con parámetros especiales hay que usar la member initialization list.
- Esto es útil no sólo para crear objetos que no pueden cambiar una vez contruidos (como los atributos `const` y las referencias) sino que también es necesario si queremos construir otros objetos con parámetros custom, sean nuestros atributos o nuestros ancestros (herencia).

Iniciación de atributos constantes	
<pre>1 struct Vector { 2 int * const data; 3 const int size; 4 5 Vector(int size) { 6 // atributos ya contruidos; aca solo los re-asigno 7 this->data = malloc(size*sizeof(int)); 8 this->size = size; 9 } 10 11 struct Vector { 12 int * const data; 13 const int size; 14 15 Vector(int size) : data(malloc(size*sizeof(int))), 16 size(size) { 17 18 }</pre> <div>8</div>	<ul style="list-style-type: none">La member initialization list es el único lugar para inicializar atributos constantes y referencias.
Iniciación de atributos no-default	
<pre>1 struct DoubleVector { 2 Vector fg; 3 Vector bg; 4 5 DoubleVector(int size) { 6 // fg, bg?? 7 } 8 } 9 10 struct DoubleVector { 11 Vector fg; 12 Vector bg; 13 14 DoubleVector(int size) : fg(size), bg(size) { 15 } 16 }</pre> <div>9</div>	<ul style="list-style-type: none">La member initialization list es el único lugar para inicializar atributos que son objetos que no tienen un constructor por default o sin parámetros.