Pasaje de objetos en C++ - Asignación

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

Facultad de Ingeniería
Universidad de Buenos Aires

Asignación

Asignación por copia

Asignación

```
1  Vector f(Vector v) {
2    Vector a(v);
3    Vector b = v;
4
5    Vector c(5);
6
7    c = v;
8
9    return v;
10 }
```


- En la línea 1 se recibe por copia un vector al que llamaremos `v`.
- En la línea 2 y 3 se crean 2 vectores más copiandose de `v`, ambos llaman al constructor por copia.
- En la línea 9 se retorna un vector por copia también salvo que `Vector` implemente el constructor por movimiento en cuyo caso `v` se mueve y no se copia.
- En la línea 7 sucede algo distinto. El vector `c` copia el contenido del vector `v`. Pero el objeto `c` ya estaba creado asi que en vez de llamar al constructor por copia llama al operador asignación por copia.

Asignación por copia: un objeto creado copiando de otro

```
1  struct Vector {
2    int *data;
3    int size;
4
5    Vector& operator=(const Vector &other) {
6      if (this == &other) {
7        return *this; // other is myself!
8      }
9
10     if (this->data)
11         free(this->data);
12
13     this->data = (int*)malloc(other.size*sizeof(int));
14     this->size = other.size;
15     memcpy(this->data, other.data; this->size);
16
17     return *this;
18   }
19 };
```


- Para copiar el contenido de un objeto en otro ya creado se usa el operador asignación.
- Como el objeto `this` ya esta creado, debemos recordar que todos sus atributos estan ya creados: no podemos cambiar ninguno de sus atributos constantes.
- Validar que no nos hayan quitado el ownership de nuestros recursos.
- Todos los objetos en C++ son copiables por asignación asi que si un objeto no implementa la sobrecarga del operador asignación, C++ le creara una implementación por default que hara una copia bit a bit naive.
- También es posible que nos asignemos a nosotros mismos (haciendo `vec = vec;`). Debemos programar el operador asignación de tal forma que evite copiarse a si mismo.
- El operador asignación no es el único operador que se puede sobrecargar. Ya veremos otros y en más detalle en las próximas clases.

Asignación

Asignación por movimiento

Asignación por movimiento

```
1  struct Vector {
2    int *data;
3    int size;
4
5    Vector& operator=(Vector&& other) {
6      if (this == &other) {
7        return *this; // other is myself!
8      }
9
10     if (this->data)
11         free(this->data);
12
13     this->data = other.data;
14     this->size = other.size;
15
16     other.data = nullptr;
17     other.size = 0;
18
19     return *this;
```

Ej Asignación por movimiento: swap de objetos

```
10 void swap(Vector& a, Vector& b) {
11     Vector t = a;  // copia (constructor)
12     a = b;  // copia (asignacion)
13     b = t;  // copia (asignacion)
14 }
```

```
10 void swap(Vector& a, Vector& b) {
11     Vector t = std::move(a); // a se mueve a t (constructor)
12     a = std::move(b); // b se mueve a a (asignacion)
13     b = std::move(t); // t se mueve a b (asignacion)
14 }
```


Asignación

Objetos no copiables

Objetos no copiables

```
1  struct File {
2    public:
3    File copy(const char *to_where) { ... }
4
5    private:
6    File(const File &other) = delete;
7    File& operator=(const File &other) = delete;
8
9  };
```


- En C++11 podemos decir que tanto el constructor por copia como el de asignación estan borrados (`delete`). Si en algun momento intentamos hacer una copia el compilador dara un error.
- Pero si trabajamos en C++98, debemos usar algun workaround: declarar y definir el constructor por copia y el operador asignación y que su implementación sea fallar (lanzar una excepción). El intento fallido de copia se detecta en runtime.
- Otra forma seria declarar pero no definir ni el constructor por copia ni el operador asignación y hacerlos privados. El intento fallido de copia se detecta en tiempo de compilación y linkeo.