

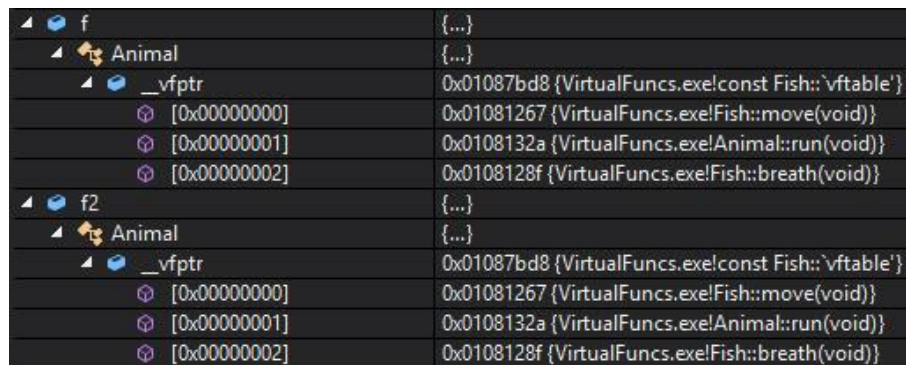
Práctica 15 – Funciones Virtuales

Julián Jiménez González

1. ¿Dónde está situada la tabla de funciones virtuales?

La tabla de funciones virtuales no tiene un lugar fijo, cada compilador puede situarla en un lugar distinto, aunque lo normal es que se haga en el **Data Segment**. Lo que guarda cada instancia de un objeto con funciones virtuales, es un **puntero a esa tabla**.

Como se puede ver a continuación, creando 2 instancias de la clase **Fish**, ambas almacenan un puntero a la tabla, situada en **0x01087bd8**.



f	{...}
Animal	{...}
_vfptr	0x01087bd8 {VirtualFuncs.exe!const Fish::`vftable'}
[0x00000000]	0x01081267 {VirtualFuncs.exe!Fish::move(void)}
[0x00000001]	0x0108132a {VirtualFuncs.exe!Animal::run(void)}
[0x00000002]	0x0108128f {VirtualFuncs.exe!Fish::breath(void)}
f2	{...}
Animal	{...}
_vfptr	0x01087bd8 {VirtualFuncs.exe!const Fish::`vftable'}
[0x00000000]	0x01081267 {VirtualFuncs.exe!Fish::move(void)}
[0x00000001]	0x0108132a {VirtualFuncs.exe!Animal::run(void)}
[0x00000002]	0x0108128f {VirtualFuncs.exe!Fish::breath(void)}

2. ¿Cuánto espacio ocupa adicionalmente un objeto por tener una tabla de funciones virtuales?

Viendo el mismo ejemplo de antes, el objeto en sí ocupa tan sólo un puntero a mayores (depende de la arquitectura, típicamente 4 u 8 bytes). La tabla sí que ocupa a mayores un puntero por cada función declarada en la clase que sea virtual.

3. ¿Qué pasa si llamo a un método virtual desde el constructor?

No ocurre nada extraño si el método está definido en la clase base, o en su defecto es un método virtual puro en la base y está definido en la propia clase, y **la llamada se realiza desde el constructor de la derivada**.

Si existe una llamada en el constructor de la clase base, lo que ocurre es que antes de ejecutar el constructor de la derivada se llama al de la base. Como todavía no está creado el objeto, la tabla de funciones virtuales del objeto todavía no está definida, y por defecto se va a usar la de la clase base. En el siguiente ejemplo se puede ver, dadas las clases Fish y Animal, que antes de ejecutar el constructor de Fish su `_vfptr` apunta a la tabla de Animal.

```
Animal() { this->run(); }
```

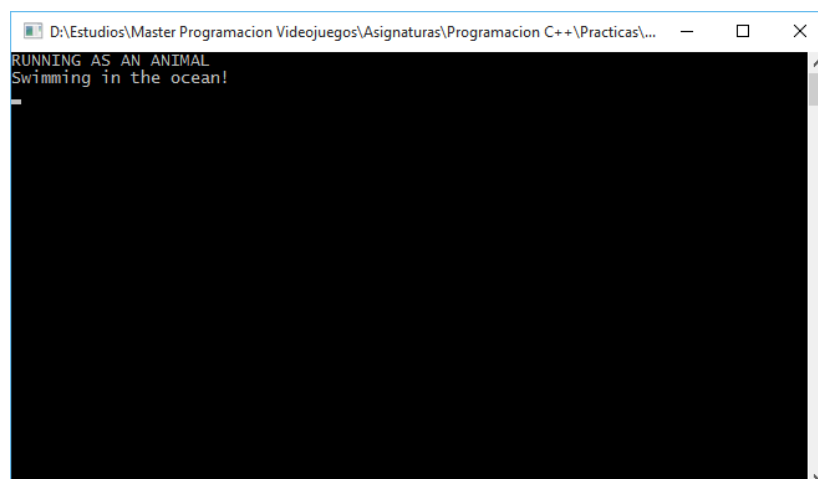
```
Fish::Fish() {
    this->move();
}
```

▲ this	0x0050fafc {...}	Fish * {A
▲ [Animal]	{...}	Animal
▲ _vfptr	0x00107b34 {VirtualFuncs.exe!const Animal::vftable'} {0x00101028 {VirtualF	void * *
[0x00000000]	0x00101028 {VirtualFuncs.exe!Animal::move(void)}	void *
[0x00000001]	0x00101339 {VirtualFuncs.exe!Animal::run(void)}	void *
[0x00000002]	0x00101357 {VirtualFuncs.exe!__purecall}	void *
▲ Animal	{...}	Animal
▲ _vfptr	0x00107b34 {VirtualFuncs.exe!const Animal::vftable'} {0x00101028 {VirtualF	void * *
[0x00000000]	0x00101028 {VirtualFuncs.exe!Animal::move(void)}	void *
[0x00000001]	0x00101339 {VirtualFuncs.exe!Animal::run(void)}	void *
[0x00000002]	0x00101357 {VirtualFuncs.exe!__purecall}	void *

Una vez que se entra en la ejecución del constructor de Fish, se crea su propia tabla y _vfptr apunta a la nueva dirección.

▲ this	0x0050fafc {...}	Fish *
▲ Animal	{...}	Animal
▲ _vfptr	0x00107b80 {VirtualFuncs.exe!const Fish::vftable'} {0x0010126c {VirtualFun	void * *
[0x00000000]	0x0010126c {VirtualFuncs.exe!Fish::move(void)}	void *
[0x00000001]	0x0010132a {VirtualFuncs.exe!Fish::run(void)}	void *
[0x00000002]	0x00101299 {VirtualFuncs.exe!Fish::breath(void)}	void *

Por lo tanto, antes de ejecutar el constructor de Fish, se ejecuta el de Animal, que tiene una llamada a run(), y después se ejecuta el propio de Fish, que llama a move(). Este es el resultado:



4. Comparar la llamada a una función virtual con la llamada a una función no virtual. ¿Cuántos pasos adicionales tienen que realizarse para llamar a una función cuando esta es virtual?

En el caso de una función no virtual, el objeto tiene un puntero al código en memoria de la clase, que es común para todas las instancias, por lo que hay 1 salto.

En el caso de una función virtual, hay 1 salto del objeto a la tabla de funciones virtuales mediante `_vfptr`, y después otro salto desde la tabla de funciones virtuales al código de la función en sí, ya que hay que recordar que la tabla de funciones virtuales guarda las direcciones del código de las funciones.

Por lo tanto hay 1 paso adicional, que además se realiza en tiempo de ejecución, mientras la llamada a una función no-virtual se realiza en tiempo de compilación.