Virtual Table, Dynamic & Static Binding

Created	@April 14, 2025 1:52 PM
Class	Prog2

Virtual Table

La Virtual table è una struttura creata dal compilatore per ogni classe che contiene almeno un metodo `virtual`. È essenzialmente una **tabella di puntatori a funzione**, in cui ogni voce punta alla versione corretta di ciascun metodo virtuale definito per quella classe.

Quando si istanzia un oggetto di una classe con metodi virtuali, **quell'oggetto conterrà un puntatore alla vtable della sua classe**. In questo modo, se si accede all'oggetto tramite un puntatore alla classe base, la vtable consente comunque al programma di trovare e chiamare la versione corretta del metodo, quella appartenente alla classe effettiva dell'oggetto.

Questo meccanismo è ciò che permette il cosiddetto **dynamic binding**, o **late binding**.

🔁 Binding:

In C++, ogni volta che chiami una funzione, il compilatore (o il programma) deve decidere **quale versione di quella funzione eseguire**. Questo processo si chiama **binding** e può avvenire in due modi:

🔷 Static Binding (early binding)

Con lo **static binding**, la decisione su quale funzione chiamare viene presa **durante la compilazione**. Questo è il comportamento predefinito in C++ per tutte le funzioni **non virtuali**. Il compilatore sa già dove si trova la funzione nel codice, quindi genera direttamente il codice macchina per chiamarla.

Il vantaggio è che è molto efficiente: niente controlli a runtime, nessuna indagine sul tipo reale dell'oggetto. Tuttavia, non consente il polimorfismo dinamico. Se stai usando un puntatore alla classe base, verrà sempre chiamata la versione del metodo definita nella base, anche se l'oggetto è di una classe derivata.

🔷 Dynamic Binding (late binding)

Il **dynamic binding**, invece, entra in gioco con le funzioni `virtual`. In questo caso, la decisione su quale funzione chiamare viene rimandata a **runtime**, quando il programma sa esattamente **di quale tipo è l'oggetto** che stai usando. Grazie alla **virtual table**, il programma può consultare la tabella e trovare l'indirizzo della funzione corretta da chiamare.

Questo è ciò che permette il **polimorfismo dinamico**: ovvero, la possibilità di scrivere codice che lavora con riferimenti o puntatori a una classe base, ma che si comporta correttamente anche con oggetti di classi derivate, invocando le versioni sovrascritte dei metodi.