Dispatch e rappresentazione degli oggetti in C++

Static dispatch

I metodi non dichiarati come `virtual` utilizzano static dispatch: il compilatore determina in base al tipo *statico* il metodo da chiamare, e la chiamata a runtime è equivalente a quella di una normale funzione.

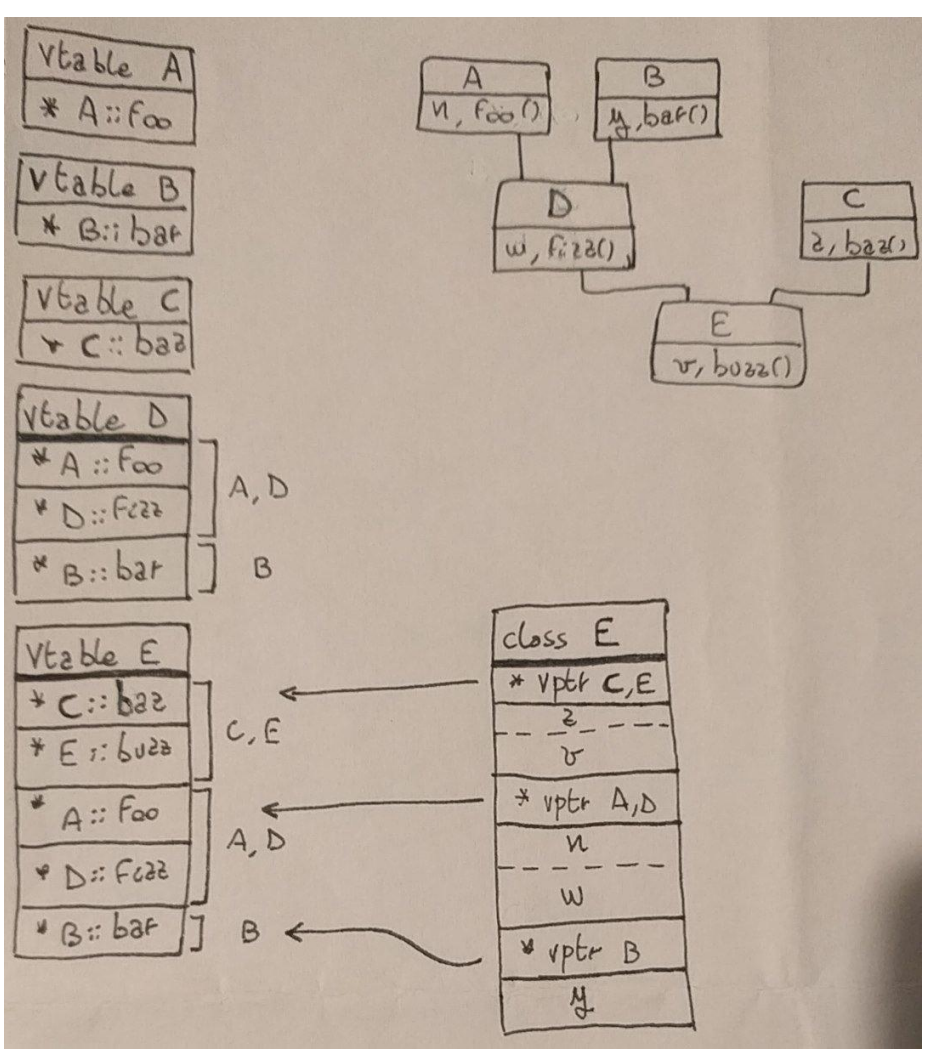
```
class A {
  public:
      char foo() { return 'A'; }
}

class B : public A {
  public:
      char foo() { return 'B'; }
}

A obj = B();
obj.foo();
// => 'A'
```

Se un'invocazione è ambigua perché metodi con signature identica vengono forniti da superclassi distinte (ereditarietà multipla), questa deve essere disambiguata con la sintassi `obj.A::foo()` (altrimenti errore di compilazione).

Metodi virtuali

I metodi con il qualificatore `virtual` supportano dynamic dispatch. A causa dell'ereditarietà multipla, non è possibile impiegare l'approccio della JVM (offset dei metodi mantenuto costante nelle sottoclassi). Soluzione:

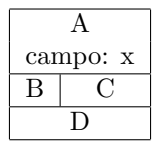
- ad ogni *classe* con metodi virtuali corrisponde una *vtable* (tabella di puntatori a codice di metodi) *per ogni vtable* di ogni superclasse estesa (o una sola se non estende niente);
- le vtable di una sottoclasse sono costruite copiando quelle delle superclassi, aggiungendo alla prima vtable i metodi introdotti dalla sottoclasse, e rimpiazzando i metodi sovrascritti mantenendo comunque la stessa posizione nella tabella (a questo livello si può gestire come ereditarietà singola);
- gli oggetti appartenenti a una classe con metodi virtuali conservano dei *vptr*, puntatori alle vtable della propria classe, tra i campi sull'heap (ciascuno prima dei campi associati alle classi rappresentate dalla vtable puntata).



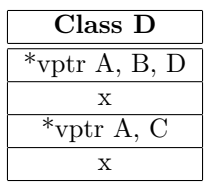
Se sia `D` che `C` fornissero lo stesso metodo `baz`, si utilizzerebbe la vtable appropriata per eseguire il codice giusto (e nel sorgente sarebbe obbligatorio disambiguare come con metodi statici).

Ereditarietà con replicazione

Avendo la seguente gerarchia di classi:

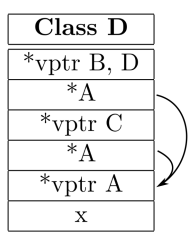


un oggetto di classe `D` istanzia due oggetti di `A` (chiamando due volte il costruttore). La sua rappresentazione sull'heap contiene un vptr alla vtable `A,B,D` (copiata da `B` con l'aggiunta dei metodi di `D`), e uno alla vtable `A,C` (copiata da `C`).



Ereditarietà con condivisione

Alternativamente, se `B` e `C` utilizzano *virtual inheritance* (`class X : virtual public A`) viene creata un'unica copia delle superclassi in comune, e sull'heap avremo:



Chiamare un metodo di `A` ha un piccolo overhead per via del livello aggiuntivo di indirezione. Inserire il puntatore `*A` anche dopo `C` permette di usare `D` con tipo statico `C` facendo un offset sul descrittore.