07_OneDefinitionRule

Table of contents

One Definition Rule (ODR)

"Posso avere tante dichiarazioni ma una sola definizione" sarebbe la frase che spiega la ODR, ma è imprecisa: le unità di traduzione possono comunicare tra di loro, e a queste servono regole per precisare come sono fatti i dati e le operazioni.
Negli header file per esempio vengono definiti i tipi di dato e le dichiarazioni delle funzioni: tante unità di traduzione possono avere modi diversi di calcolare un fattoriale.

Le unità di traduzione devono concordare con un'interfaccia (contenente le dichiarazioni di dati o funzioni); per evitare che un'unità si trovi a lavorare per sbaglio con un tipo/funzione errato, seguiamo una regola precisa: Don't Repeat Yourself (DRY) o "write once", le dichiarazioni vengono scritte una volta sola, in un *header file*.

La One Definition Rule detta che:

1. ogni unità di traduzione (il risultato della fase di preprocessing su un file sorgente(`HelloWorld.preproc.cpp`)) che forma il programma, può contenere *non più di una definizione di una data variabile, funzione, classe, enumerazione o template*;
2. ogni programma, deve contenere *esattamente 1 definizione di ogni variabile e di ogni funzione non-inline* usate nel programma;
3. ogni funzione inline deve essere *definita in ogni unità di traduzione* che la utilizza;
4. in un programma possono esserci *più definizioni di classe, enumerazione, funzione inline, template di classe e di funzione*, a condizione che:
    1. siano *sintatticamente identiche*;
    2. siano *semanticamente identiche*.

Soluzioni alle violazioni ODR

Soluzione all'inclusione ripetuta `#pragma once`

Una soluzione al problema può essere l'uso della direttiva speciale del processore

```
#pragma once
```

Serve per dire al nostro compilatore, che la nostra unità di traduzione deve essere inclusa una sola volta.

Il problema con questa soluzione è che non è standard: la possiamo usare e funziona come la soluzione che vedremo fra un attimo, ma non tutti i compilatori moderni la supportano; inoltre non assicura che una copia dello stesso header nel nostro progetto non sia inclusa.

Guardie contro l'inclusione ripetuta `#ifndef` & `#endif`

Le variabili del preprocessore, scritte tutte in maiuscolo per convenzione, ci aiuteranno a sostituire la `#pragma` che magari non disponibile nel nostro processore.

- `ifndef` pone la domanda al preprocessore: è già stata definita la variabile x?
  → `no`, definisco la variabile con `#define`;
  → `yes`, abortisco saltando tutta la direttiva d'inclusione fino a `#endif`.
- `endif` indica il punto di fine inclusione di unità di traduzione.

```
// Razionale.hh
#ifndef RAZIONALE_HH_INCLUDE_GUARD
#define RAZIONALE_HH_INCLUDE_GUARD 1

class Razionale {
        // ...
};

#endif
```

Guardie d'inclusione ripetuta nell'header file `iostream`

```
#ifndef _GLIBCXX_IOSTREAM
#define _GLIBCXX_IOSTREAM 1
// ...
#endif /* _GLIBCXX_IOSTREAM */
```

Espansione `inline` per ogni unità di traduzione

Cosa è una funzione `inline`?

Si dice dichiarata `inline`, una funzione il quale suo corpo viene ripetuto per intero, in un'unità di traduzione, a fini di ottimizzare i tempi di esecuzione. Questo è dovuto dal fatto che, alcuni programmi, risentono delle *function call*: un numero esagerato di queste chiamate verso un punto in memoria distante, causa rallentamenti. Ripetere il corpo della funzione nella stessa unità, allenta la presa sul processore.
Da notare che seppur utile, l'uso incosiderato della `inline` porta a problemi piuttosto che soluzioni: dichiarare una funzione molto corposa (con tante linee) `inline`, aumenta inutilmente le linee di codice da compilare. È buona convenzione dichiarare `inline`, le funzioni con corpo piccolo.

Proprio per come sono fatte le funzioni `inline`, il corpo di queste deve essere in ciascuna unità di traduzione. Con l'unione delle regole sopra viste, e l'uso degli header file, il problema viene eliminato alla radice.

```
// Example.hh
#ifndef EXAMPLE_H
#define EXAMPLE_H
inline int sum(int a, int b){
        return a + b;
}
#endif

// Main.cpp
#include "Example.hh"
#include <iostream>
int main() {
```

```
        std::cout << sum(1,3) << std::endl;
        return 0;
}
```

Esempi di violazione ODR

Punto 1

```
// definizione multipla
// ERRORE: ambiguità
struct S { int a; };
struct S { char c; double d };
int a;
int a;
```

Mettere le inclusioni in cima e non ripetere mai la dichiarazione di funzione sono due regole tuttavia non sufficienti per assicurare la non violazione della ODR.

```
// Razionale.hh
class Razionale {
        // ...
};

// Polinomio.hh
#include "Razionale.hh"
class Polinomio {
        // ...
};

// Calcolo.cc
#include "Razionale.hh"
#include "Polinomio.hh"
// ...
```

Se provassimo a compilare, ci viene dato errore siccome violata la regola n.1: troviamo 2 unità di traduzione nello stesso file (`Razionale.hh`) siccome la linea d'inclusione in `Calcolo.cc` lo dice.

Punto 2

```
// ERROE: foo() dichiarata ma non definita

// foo.hh
int foo(int a);

// file1.cc
#include "foo.hh"
int foo(int a) { return a + 1; }

// file2.cc
#include "foo.hh"
int foo(int a) { return a + 2; }

// file3.cc
#include "foo.hh"
int bar(int a) { return foo(a); }
```

Punto 4

```
// file1.cc
strcut S { int a; int b; };
S s;

// file2.cc - ERROR: sintassi diversa per S
// inversi 'a' e 'b'
struct S { int b; int a; };
extern S s;

// oppure

// file1.cc
typedef T int;
struct S { T a; T b; };

// file2.cc - ERROR: semantica diversa per T
// dico che 'T' è int ma poi scrivo 'double'
typedef T double;
struct S { T a; T b; };
```

07/03/2023