Metodi inline

La parola chiave `inline` si applica alle definizione di funzioni o funzioni membro come forma di ottimizzazione. Essa è una speciale direttiva al compilatore che, se eseguita, consiste nel sostituire la chiamata a funzione con il corpo della funzione stessa. In molti casi, per funzioni composte da poche istruzioni semplici e invocate frequentemente, ciò può comportare un incremento delle prestazioni, a scapito di un aumento delle dimensioni dell'eseguibile.

Tuttavia, ciò non è sempre vero. Rendere una funzione effettivamente `inline` dipende dal compilatore, che applicando le sue euristiche sul codice, può eseguire questa direttiva o ignorarla, così come applicarla a metodi o funzioni che non abbiamo esplicitamente definito come `inline`.

L'esempio seguente mostra la definizione di una funzione `inline`:

```
// Definizione di funzione inline
inline int sum(int a, int b)
{
        return a + b;
}


int main()
{
        /* l'invocazione di una funzione inline non è differente
        da quella di una funzione ordinaria */
        sum(2, 3);
        return 0;
}
```

Copy

Nel frammento successivo sono presenti due modalità per la definizione di metodi `inline` in una classe. La prima (A) consiste nel definire il metodo contestualmente alla definizione della classe, cosa che rende la parola `inline` opzionale. La seconda (B) consente di mantenere la dichiarazione della classe scissa dalla sua implementazione effettiva, rimuovendo dallo *header file* tutti i dettagli implementativi.

```
// A - Definizione di classe con metodo inline nello header
class SimpleMath
{
public:
         inline int sum(int a, int b)
         {
                 return a + b;
         }
};
// B - Definizione alternativa di classe con metodo inline
// simplemath.h
class SimpleMath
{
public:
         int sum(int a, int b);
};
// simplemath.cpp
```

```
inline int SimpleMath::sum(int a, int b)
{
   return (a + b);
}
```

Copy

C’è una relazione tra questa forma di ottimizzazione e l’incapsulamento, anche se non immediata, in quanto ha ricadute più sul piano pratico che teorico. A differenza delle **macro**, che sono implementate mediante direttive al preprocessore, il corpo dei metodi `inline` viene sostituito all’invocazione in fase di compilazione.

Ciò implica che la definizione del corpo del metodo deve essere presente in ogni **unità di compilazione**, costituita da ogni sorgente in cui tale metodo viene invocato più tutti i file *header* incorporati dal preprocessore. Questo risultato lo si può ottenere in due modi: includendo la definizione nella dichiarazione della classe (A) oppure ripetendo la definizione in ogni sorgente in cui viene invocata, quando la dichiarazione e la definizione del metodo `inline` avvengono in due contesti differenti (B), come viene illustrato nel listato seguente:

```
#include <iostream>
#include "simplemath.h" // versione B
// rimuovere la ri-definizione del metodo produrrà un errore di compilazione
inline int SimpleMath::sum(int a, int b)
{
   return (a + b);
}
int main()
{
        SimpleMath a;
        std::cout << a.sum(2, 3) << std::endl;
        return 0;
}
```

Copy

La prima opzione rende necessario esporre alcuni dettagli implementativi, mentre la seconda ha delle implicazioni di tipo pratico: la ripetizione di una frammento di codice è sempre fonte di problemi nel caso di aggiornamenti o correzioni, e contrasta uno dei benefici principali dell’incapsulamento, cioè il riutilizzo del codice.

L’uso della parola chiave `inline`, sebbene preferibile all’uso delle macro perché non inquina lo spazio dei nomi del contesto in cui viene applicata, pone quindi qualche problema di gestione dell’incapsulamento, e va riservato a casi in cui la sua efficacia è comprovata da risultati sperimentali. Nel dubbio è preferibile lasciare che l’ottimizzazione del codice venga fatta dal compilatore al momento opportuno, cioè dopo la compilazione dei sorgenti.