4.11 Classi e modularizzazione

Nel paragrafo 4.2 abbiamo accennato al fatto che le definizioni di funzioni proprie di una classe sono normalmente situate in un file diverso da quello dove la classe è definita. La classe viene infatti generalmente definita in un file header, con estensione ".h", mentre le sue funzioni vengono definite in un apposito file con estensione ".cpp". Nel nostro esempio, abbiamo definito la classe `tempo` in un file dal nome `tempo.h`, mentre le definizioni delle sue funzioni sono state raccolte nel file `tempo.cpp`. La funzione `main()`, nella quale possiamo pensare che occorrano tutte le istruzioni che usano oggetti della classe `tempo` viste in precedenza, è definita nel file `main.cpp`.

Poiché ogni file con estensione ".cpp" può essere compilato separatamente, è importante che al suo interno siano visibili tutti gli identificatori utilizzati. Nel nostro esempio, i file `tempo.cpp` e `main.cpp` fanno riferimento alla classe `tempo`, la cui definizione si trova nel file `tempo.h`. Per questo motivo va inserita la direttiva `#include "tempo.h"` all'inizio di entrambi i file. Come spiegato al paragrafo 3.1, il compilatore si comporta come se il file `tempo.h` fosse copiato fisicamente negli altri due nel punto in cui compare la direttiva. La presenza degli apici in `#include "tempo.h"` significa che stiamo chiedendo al compilatore di cercare il file `tempo.h` fra quelli del direttorio corrente; con questa modalità è possibile includere un file arbitrario, specificandone in maniera esplicita il direttorio. Le parentesi angolate nell'altra direttiva incontrata fino a questo punto (`#include <iostream.h>`) denotano invece il fatto che il file `iostream.h` va cercato fra quelli in un direttorio standard di inclusione.

Come vedremo nel seguito di questo capitolo, un file header può a sua volta includere altri file header, ed in effetti questa è una situazione tipica. A causa di ciò, il compilatore può, mentre sta compilando un file, ritrovarsi con una definizione multipla di una certa classe, poiché un file header viene incluso più volte. Per ovviare a questo problema, si inseriscono nei file header ulteriori direttive, chiamate "di isolamento", come esemplificato nel seguito per il file `tempo.h`.

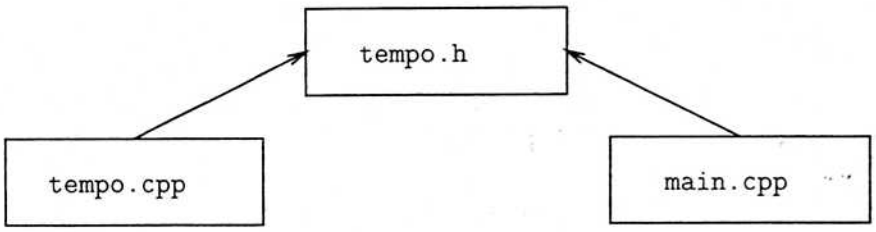

Figura 4.2 Schema di inclusione di file di dichiarazione e definizione di classi

```
// File tempo.h
#ifndef TEMPO_H
#define TEMPO_H
class tempo
{ // ...
};
#endif
```

`TEMPO_H` è un identificatore scelto in maniera arbitraria. Intuitivamente, il compilatore considera il codice compreso fra `#ifndef TEMPO_H` e `#endif` solamente se non ha mai incontrato precedentemente una direttiva `#define TEMPO_H` (`ifndef` sta per "if not defined", ovvero, "se non definito"). Pertanto nessun file che includa `tempo.h`, direttamente oppure tramite l'inclusione di altri file, potrà mai includere più volte il codice della classe `tempo`.

Esercizio 4.7 [Soluzione a pag. 536] Descrivere una semplice situazione in cui compilando un singolo file, il compilatore si trova di fronte ad una definizione multipla a causa di direttive `#include` di file header che non contengono direttive di isolamento. □

La figura 4.2 riassume le inclusioni per un progetto che consiste dei tre file prima menzionati. Come accennato nel paragrafo 3.7, il file `tempo.cpp` può essere compilato separatamente, generando il file `tempo.o` che contiene il codice oggetto. Quest'ultimo, dal quale non è possibile risalire al codice delle funzioni della classe `tempo`, può sostituire il file `tempo.cpp`, nel senso che il compilatore non ha più bisogno di `tempo.cpp` per creare il codice eseguibile relativo alla funzione `main()` (mentre ha chiaramente sempre bisogno dei file `main.cpp` e `tempo.h`).

Il concetto di information hiding risalta in maniera particolarmente evidente, perché un utente della classe `tempo` potrebbe non avere affatto a disposizione il file `tempo.cpp`, ma solamente il file `tempo.o`, ed in tal caso non avrebbe maniera di conoscere il codice delle funzioni proprie.

Esercizio 4.8 Studiare sul manuale del proprio compilatore come sia possibile compilare solamente il file `tempo.cpp`, e capire quali file vengano generati dal compilatore. Verificare successivamente che è possibile generare il codice eseguibile a partire da questi ultimi e dai file `main.cpp` e `tempo.h`. □

Come abbiamo visto al paragrafo 4.9.2, esistono delle funzioni, come `ostream& operator<<(ostream&,tempo)`, che sono fortemente correlate alla classe `tempo`, ma che per ragioni tecniche devono essere definite al suo esterno. È opportuno che la loro dichiarazione sia inserita nel file `tempo.h`, mentre la loro definizione sarà posta nel file `tempo.cpp`.

Chiudiamo il paragrafo segnalando che, quando in un grande progetto si scrivono molti file "`.cpp`" contenenti definizioni di funzioni proprie di classe, tipicamente viene costruito uno o più file di *libreria* a partire dai rispettivi file "`.o`". Un file di libreria viene usato dal collegatore per generare il codice eseguibile. Il vantaggio nell'usare librerie è che il collegatore è in grado di estrarre dalla libreria solamente i file "`.o`" necessari, esonerando il programmatore dal conoscere esattamente quali di questi file abbia bisogno per il suo progetto. Ogni compilatore fa uso di un direttorio standard di librerie, e quindi questo meccanismo tipicamente è, per le librerie predefinite, trasparente all'utente.