

CAPITOLO 5

Le funzioni in C

Obiettivi

- Comprendere come costruire i programmi in modo modulare partendo da piccoli pezzi chiamati funzioni.
- Introdurre le comuni funzioni matematiche disponibili nella libreria standard del C.
- Essere in grado di creare nuove funzioni.
- Comprendere i meccanismi utilizzati per passare le informazioni alle funzioni.
- Introdurre le tecniche di simulazione che utilizzano la generazione di numeri casuali.
- Comprendere come scrivere e utilizzare le funzioni che richiamano se stesse.

5.1 Introduzione

La maggior parte dei programmi per computer che siano stati scritti per risolvere problemi del mondo reale sono più corposi dei programmi presentati sinora, in questi primi capitoli. L'esperienza ha dimostrato che il modo migliore, per sviluppare e amministrare un programma corposo, è di costruirlo partendo da pezzi più piccoli o *moduli* ognuno dei quali sia più maneggevole del programma originale. Questa tecnica è detta *dividi e conquista* (dal latino *divide et impera*). Questo capitolo descriverà le caratteristiche del linguaggio C che facilitano la progettazione, l'implementazione, il funzionamento e la manutenzione di programmi corposi.

5.2 I moduli di programma in C

I moduli in C sono chiamati *funzioni*. I programmi C sono scritti tipicamente combinando le nuove funzioni scritte dal programmatore con quelle “preconfezionate” disponibili nella *libreria standard del C*. In questo capitolo discuteremo di entrambi i tipi di funzione. La libreria standard del C fornisce una ricca collezione di funzioni per eseguire i comuni calcoli matematici, per la manipolazione delle stringhe e dei caratteri, per l'input/output e per molte altre operazioni utili. Ciò renderà più semplice il lavoro del programmatore, poiché le suddette funzioni forniranno molte delle capacità di cui egli avrà bisogno.



Buona abitudine 5.1

Familiarizzate con la ricca collezione di funzioni incluse nella libreria standard del C.



Ingegneria del software 5.1

Evitate di “inventare nuovamente la ruota”. Utilizzate le funzioni incluse nella libreria standard del C, qualora sia possibile, invece di scrivere delle nuove funzioni. Ciò ridurrà il tempo di sviluppo del programma.



Obiettivo portabilità 5.1

Utilizzare le funzioni della libreria standard del C aiuterà a rendere più portabili i programmi.

Per quanto le funzioni della libreria standard non facciano tecnicamente parte del linguaggio C, esse sono fornite immancabilmente con tutti i sistemi C. Le funzioni `printf`, `scanf` e `pow` che abbiamo utilizzato nei capitoli precedenti sono appunto delle funzioni della libreria standard.

Per definire dei compiti specifici, il programmatore potrà scrivere delle funzioni che possano essere utilizzate in molti punti del programma. Queste sono a volte chiamate *funzioni definite dal programmatore*. Le istruzioni che definiscono effettivamente la funzione, saranno scritte solo una volta e saranno nascoste alle altre funzioni.

Le funzioni sono *invocate* da una *chiamata di funzione* che specifica il nome della funzione e fornisce delle informazioni (gli *argomenti*) di cui la funzione chiamata ha bisogno, per completare le attività per le quali è stata progettata. Una tipica analogia per tutto ciò è quella della struttura gerarchica di un'azienda. Un capo (la *funzione chiamante* o *chiamante*) chiede a un operaio (la *funzione chiamata*) di eseguire un compito e tornare indietro a riferire, quando il lavoro sarà stato eseguito (Figura 5.1). Per esempio, una funzione che voglia visualizzare delle informazioni sullo schermo richiederà la funzione operaia `printf` per eseguire quel compito, in seguito `printf` visualizzerà le suddette informazioni e tornerà indietro a riferire, ovvero *restituirà il controllo del programma*, alla funzione chiamante quando il suo compito sarà stato completato. La funzione capo non sa in che modo quell'operaia eseguirà i compiti che le sono stati assegnati. L'operaia potrebbe anche richiamare altre funzioni operaie e il capo non ne sarebbe a conoscenza. Vedremo presto in che modo questo “incapsulamento” dei dettagli dell'implementazione promuoverà una buona progettazione del software. La Figura 5.1 mostra la funzione `main` mentre comunica con diverse operaie in modo gerarchico. Osservate che `worker1` agisce come una funzione capo per `worker4` e `worker5`. Le relazioni tra le funzioni potrebbero anche essere diverse dalla struttura gerarchica mostrata in questa figura.

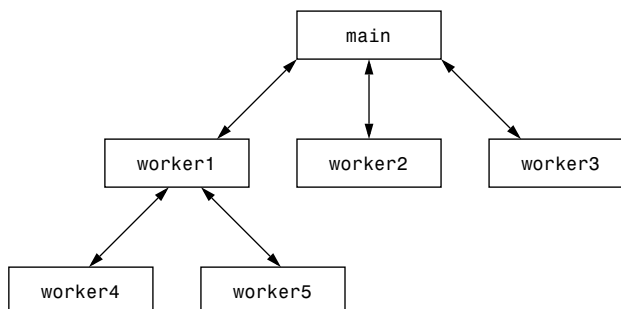


Figura 5.1 Relazione gerarchica tra la funzione capo e quelle operaie

5.3 Le funzioni della libreria matematica

Le funzioni della libreria matematica consentono al programmatore di eseguire certi tipici calcoli matematici. In questo contesto utilizzeremo varie funzioni matematiche per introdurre il concetto di funzione. Più tardi nel libro, discuteremo molte delle altre funzioni incluse nella libreria standard del C.

Normalmente le funzioni sono utilizzate in un programma scrivendo il nome delle stesse, seguito da una parentesi tonda aperta, dall'*argomento* (oppure da una lista di argomenti separati da virgole) della funzione e da una parentesi tonda chiusa. Per esempio, un programmatore che desideri calcolare e visualizzare la radice quadrata di 900,0 potrà scrivere:

```
printf( "%.2f", sqrt( 900.0 ) );
```

Nel momento in cui questa istruzione sarà eseguita, sarà invocata la funzione `sqrt` della libreria matematica per calcolare la radice quadrata del numero contenuto nelle parentesi (900.0). Il numero 900,0 è l'argomento della funzione `sqrt`. L'istruzione precedente visualizzerà 30,00. La funzione `sqrt` accetta un argomento di tipo `double` e restituisce un risultato `double`. Tutte le funzioni della libreria matematica restituiscono il tipo di dato `double`. Si osservi che i valori di tipo `double`, come i valori di tipo `float`, possono essere mandati in output usando la specifica di conversione `%f`.



Collaudo e messa a punto 5.1

Includete il file di intestazione matematico, utilizzando la direttiva del preprocessore `#include <math.h>`, quando utilizzate le funzioni della libreria matematica.

Gli argomenti della funzione possono essere delle costanti, variabili o espressioni. Supponendo che `c1 = 13,0`, `d = 3,0` e `f = 4,0`, l'istruzione

```
printf( "%.2f", sqrt( c1 + d * f ) );
```

calcolerà e visualizzerà la radice quadrata di $13,0 + 3,0 * 4,0 = 25,0$, vale a dire 5,00.

Nella Figura 5.2 sono riassunte alcune delle funzioni incluse nella libreria matematica del C. Nella figura le variabili `x` e `y` sono di tipo `double`.

5.4 Le funzioni

Le funzioni consentono al programmatore di suddividere in moduli un programma. Tutte le variabili dichiarate nelle definizioni di funzione sono *variabili locali*: esse sono note soltanto in quella in cui sono state definite. La maggior parte delle funzioni contiene una lista di *parametri*. Questi forniscono il mezzo per comunicare le informazioni tra le funzioni. Anche i parametri di una funzione sono delle variabili locali di quest'ultima.



Ingegneria del software 5.2

Nei programmi contenenti molte funzioni, `main` potrebbe essere implementato come un gruppo di chiamate a funzioni che eseguono il grosso del lavoro del programma.

Funzione	Descrizione	Esempio
<code>sqrt(x)</code>	radice quadrata di x	<code>sqrt(900.0)</code> è 30,0 <code>sqrt(9.0)</code> è 3,0
<code>exp(x)</code>	funzione esponenziale e	<code>exp(1.0)</code> è 2,718282 <code>exp(2.0)</code> è 7,389056
<code>log(x)</code>	logaritmo naturale di x (in base e)	<code>log(2.718282)</code> è 1,0 <code>log(7.389056)</code> è 2,0
<code>log10(x)</code>	logaritmo di x (in base 10)	<code>log10(1.0)</code> è 0,0 <code>log10(10.0)</code> è 1,0 <code>log10(100.0)</code> è 2,0
<code>fabs(x)</code>	valore assoluto di x	<code>fabs(5.0)</code> è 5.0 <code>fabs(0.0)</code> è 0,0 <code>fabs(-5.0)</code> è -5,0
<code>ceil(x)</code>	arrotonda x all'intero più piccolo non minore di x	<code>ceil(9.2)</code> è 10,0 <code>ceil(-9.8)</code> è -9,0
<code>floor(x)</code>	arrotonda x all'intero più grande non maggiore di x	<code>floor(9.2)</code> è 9,0 <code>floor(-9.8)</code> è -10,0
<code>pow(x, y)</code>	x elevato alla potenza y ()	<code>pow(2, 7)</code> è 128,0 <code>pow(9, .5)</code> è 3,0
<code>fmod(x, y)</code>	resto di x/y in virgola mobile	<code>fmod(13.657, 2.333)</code> è 1,992
<code>sin(x)</code>	seno trigonometrico di x (x è espressa in radianti)	<code>sin(0.0)</code> è 0,0
<code>cos(x)</code>	coseno trigonometrico di x (x è espressa in radianti)	<code>cos(0.0)</code> è 1,0
<code>tan(x)</code>	tangente trigonometrica di x (x è espressa in radianti)	<code>tan(0.0)</code> è 0,0

Figura 5.2 Funzioni comunemente utilizzate della libreria matematica

Ci sono diverse motivazioni per “suddividere in funzioni” un programma. L’approccio dividi e conquista rende maneggevole lo sviluppo del programma. Un’altra motivazione è la *riusabilità del software*: utilizzare le funzioni esistenti come blocchi di costruzione per creare i nuovi programmi. La riusabilità del software è uno degli elementi principali del movimento per la programmazione orientata agli oggetti, che imparerete quando studierete linguaggi derivati dal C come il C++, Java e il C# (da pronunciare “C sharp”). Con un buon nome di funzione e una buona definizione, i programmi potranno essere creati da funzioni standardizzate che eseguano dei compiti specifici, piuttosto che essere costruiti utilizzando un codice personalizzato. Questa tecnica è conosciuta come *astrazione*. Noi usiamo l’astrazione ogni volta che scriviamo dei programmi che includano delle funzioni incluse nella libreria standard, come `printf`, `scanf` e `pow`. Una terza motivazione è di evitare la ripetizione del codice all’interno di un programma. Impacchettare il codice in forma di funzione consentirà allo stesso di essere eseguito in diversi punti del programma richiamando semplicemente la funzione.



Ingegneria del software 5.3

Ogni funzione dovrebbe limitarsi a eseguire un compito singolo e ben definito, mentre il nome della funzione dovrebbe esprimere efficacemente quel compito. Ciò faciliterà l'astrazione e promuoverà la riusabilità del software.



Ingegneria del software 5.4

Se non riuscite a scegliere un nome conciso che esprima l'attività svolta dalla vostra funzione, è probabile che questa stia tentando di eseguire troppi compiti diversi. In questi casi, di solito è meglio suddividerla in diverse funzioni più piccole.

5.5 Le definizioni di funzione

Ognuno dei programmi che abbiamo presentato è stato formato da una funzione chiamata `main` che, per eseguire i propri compiti, ha richiamato quelle della libreria standard. Consideriamo ora in che modo i programmatori possano scrivere le proprie funzioni personalizzate.

Considerate un programma che utilizzi una funzione `square` per calcolare e visualizzare i quadrati degli interi compresi tra 1 e 10 (Figura 5.3).

```

1  /* Fig. 5.3: fig05_03.c
2     Creazione e utilizzo di una funzione definita dal programmatore */
3  #include <stdio.h>
4
5  int square( int y ); /* prototipo della funzione */
6
7  /* l'esecuzione del programma inizia dalla funzione main */
8  int main()
9  {
10     int x; /* contatore */
11
12     /* itera 10 volte e calcola e visualizza il quadrato di x
13        a ogni ciclo */
14     for ( x = 1; x <= 10; x++ ) {
15         printf( "%d ", square( x ) ); /* chiamata di funzione */
16     }
17     printf( "\n" );
18
19     return 0; /* indica che il programma è terminato con successo */
20
21 } /* fine della funzione main */
22
23 /* La definizione della funzione square restituisce il quadrato
24    del parametro */
24 int square( int y ) /* y è una copia dell'argomento passato
25                       alla funzione */
25 {

```

Figura 5.3 Usare una funzione definita dal programmatore (continua)

```

26     return y * y; /* restituisce il quadrato di y come intero */
27
28 } /* fine della funzione square */

```

1	4	9	16	25	36	49	64	81	100
---	---	---	----	----	----	----	----	----	-----

Figura 5.3 Usare una funzione definita dal programmatore



Buona abitudine 5.2

Inserite una riga vuota tra le definizioni di funzione, per separarle e aumentare la leggibilità del programma.

La funzione `square` sarà *invocata* o *richiamata* nel corpo di `main` all'interno dell'istruzione `printf` (riga 14)

```
printf( "%d ", square( x ) ); /* chiamata di funzione */
```

La funzione `square` riceverà una copia del valore di `x` nel *parametro* `y` (riga 24). In seguito `square` calcolerà `y * y` (riga 26). Il risultato sarà restituito alla funzione `printf` all'interno del `main` nel punto dell'invocazione di `square` e `printf` visualizzerà il risultato. Questo processo sarà ripetuto 10 volte utilizzando il comando di iterazione `for`.

La definizione della funzione `square` mostra che questa attenderà il parametro intero `y`. La parola chiave `int` che precede il nome della funzione (riga 24) indica che `square` restituirà un risultato intero. L'istruzione `return` nella funzione `square` restituirà il risultato del calcolo alla funzione chiamante.

La riga 5

```
int square( int y ); /* prototipo della funzione */
```

è un *prototipo di funzione*. L'`int` all'interno delle parentesi informa il compilatore che `square` si aspetterà di ricevere un valore intero dal chiamante. L'`int` alla sinistra del nome della funzione `square` informa il compilatore che questa restituirà al chiamante un risultato intero. Il compilatore farà riferimento al prototipo della funzione, per controllare che le chiamate a `square` (riga 14) contengano il tipo di ritorno corretto, il numero e i tipi appropriati per gli argomenti e che questi siano forniti nell'ordine corretto. I prototipi di funzione saranno trattati in dettaglio nella Sezione 5.6.

Il formato di una definizione di funzione è

```

tipo-del-valore-di-ritorno nome-della-funzione( lista-dei-parametri )
{
    dichiarazioni
    istruzioni
}

```

Il *nome della funzione* è un qualsiasi identificatore valido. Il *tipo del valore di ritorno* è quello del risultato restituito al chiamante. Il *tipo del valore di ritorno* `void` indica che una funzione non restituirà alcun valore. Un *tipo del valore di ritorno* non specificato sarà sempre conside-

rato un `int` dal compilatore. Tuttavia, l'omissione del tipo del valore di ritorno è una pratica da non incentivare. Il *tipo del valore di ritorno*, il *nome-della-funzione* e la *lista-dei-parametri*, presi assieme, sono talvolta denominati *l'intestazione della funzione*.



Errore tipico 5.1

Omettere il tipo per il valore di ritorno in una definizione di funzione è un errore di sintassi, qualora il prototipo della funzione specifichi per il valore di ritorno un tipo diverso da `int`.



Errore tipico 5.2

Dimenticare di restituire un valore da una funzione che dovrebbe farlo potrà condurre a errori inattesi. Lo standard C stabilisce che il risultato di questo tipo di omissioni sarà indefinito.



Errore tipico 5.3

Restituire un dato da una funzione con tipo del valore di ritorno `void` è un errore di sintassi.



Buona abitudine 5.3

Stabilite sempre in modo esplicito il tipo di dato restituito, anche se la sua omissione farebbe restituire un `int` per default.

La *lista dei parametri* è un elenco che specifica i parametri, separati da virgole, che saranno ricevuti dalla funzione quando sarà richiamata. Qualora una funzione non riceva alcun valore, la *lista dei parametri* sarà `void`. Il tipo di ognuno dei parametri dovrà essere specificato esplicitamente, salvo che non siano di tipo `int`. Infatti, qualora non sia stato specificato il compilatore presumerà che si tratti di un `int`.



Errore tipico 5.4

Dichiarare i parametri della funzione, qualora siano dello stesso tipo, usando la forma `double x, y` invece che `double x, double y` potrebbe provocare degli errori nei vostri programmi. La dichiarazione `double x, y` in realtà renderebbe `y` un parametro di tipo `int`, poiché `int` è il default.



Errore tipico 5.5

È un errore di sintassi inserire un punto e virgola, dopo la parentesi destra che chiude l'elenco dei parametri in una definizione di funzione.



Errore tipico 5.6

È un errore di sintassi ridefinire un parametro della funzione come variabile locale alla stessa.



Buona abitudine 5.4

Includere il tipo di dato per ogni parametro presente nel relativo elenco, anche qualora fosse del tipo di default `int`.



Buona abitudine 5.5

Per quanto non sia scorretto farlo, non utilizzate gli stessi nomi per gli argomenti passati a una funzione e per i corrispondenti parametri inseriti nella relativa definizione. Ciò aiuterà a evitare ambiguità.

Le *dichiarazioni* e le *istruzioni* inserite all'interno delle parentesi graffe formano il *corpo della funzione*. Il corpo della funzione è chiamato anche *blocco*. Le variabili potranno essere dichiarate in qualsiasi blocco e questi potranno essere nidificati. *Una funzione non può mai essere definita all'interno di un'altra funzione.*



Errore tipico 5.7

Definire una funzione all'interno di un'altra funzione è un errore di sintassi.



Buona abitudine 5.6

Scegliere dei nomi di funzione e di parametro significativi renderà i programmi più leggibili e aiuterà a evitare l'uso eccessivo di commenti.



Ingegneria del software 5.5

Una funzione non dovrebbe essere più lunga di una pagina. Meglio ancora, una funzione non dovrebbe essere più lunga di una mezza pagina. Le funzioni piccole favoriscono la riusabilità del software.



Ingegneria del software 5.6

I programmi dovrebbero essere scritti come collezioni di funzioni piccole. Ciò li renderà più semplici da scrivere, collaudare, mettere a punto e modificare.



Ingegneria del software 5.7

Una funzione che richieda un gran numero di parametri potrebbe dover svolgere troppi compiti. Considerate la possibilità di suddividerla in funzioni più piccole che eseguano dei compiti distinti. L'intestazione della funzione dovrebbe rientrare in una riga, se possibile.



Ingegneria del software 5.8

Il prototipo, l'intestazione e la chiamata della funzione dovrebbero tutti concordare nel numero, nel tipo e nell'ordine degli argomenti e dei parametri oltre che nel tipo del valore restituito.

Esistono tre modi per restituire il controllo da una funzione chiamata al punto in cui quest'ultima è stata invocata. Nel caso in cui la funzione non restituisca alcun risultato, il controllo sarà restituito semplicemente quando sarà raggiunta la parentesi graffa destra che chiude la funzione, oppure eseguendo l'istruzione

return;

Nel caso che la funzione restituisca un risultato, l'istruzione

return espressione;

restituirà al chiamante il valore dell'*espressione*.

Il nostro secondo esempio utilizzerà la funzione `maximum` definita dall'utente, per determinare e restituire il maggiore fra tre interi (Figura 5.4). I tre interi saranno presi in input con `scanf` (riga 15). In seguito, gli interi saranno passati alla funzione `maximum` (riga 19) che determinerà quale sia il maggiore di essi. Questo valore sarà restituito al `main` dall'istruzione `return` della funzione `maximum` (riga 39). Il valore restituito sarà assegnato alla variabile `largest` che sarà visualizzata in seguito da `printf` (riga 19).

5.6 I prototipi di funzione

Una delle più importanti caratteristiche del C è il *prototipo di funzione*. Il comitato dello standard C prese in prestito questa caratteristica dagli sviluppatori del C++. Un prototipo di funzione indica al compilatore il tipo del dato restituito dalla funzione, il numero dei parametri che quella si aspetta di ricevere, il tipo dei parametri e l'ordine in cui questi sono attesi. Il compilatore utilizzerà i prototipi per convalidare le chiamate di funzione. Le versioni precedenti del C non eseguivano questo tipo di controllo, perciò era possibile richiamare le funzioni in modo improprio senza che il compilatore individuasse degli errori. Durante la fase di esecuzione, tali chiamate avrebbero potuto causare degli errori fatali, o anche non fatali, ma che avrebbero provocato degli errori logici subdoli e difficili da individuare. I prototipi di funzione pongono rimedio a questa lacuna.

```

1  /* Fig. 5.4: fig05_04.c
2     Trovare il maggiore di tre interi */
3  #include <stdio.h>
4
5  int maximum( int x, int y, int z ); /* prototipo della funzione */
6
7  /* l'esecuzione del programma inizia dalla funzione main */
8  int main()
9  {
10     int number1; /* primo intero */
11     int number2; /* secondo intero */
12     int number3; /* terzo intero */
13
14     printf( "Enter three integers: " );
15     scanf( "%d%d%d", &number1, &number2, &number3 );
16
17     /* number1, number2 e number3 sono gli argomenti
18        della chiamata di funzione a maximum */
19     printf( "Maximum is: %d\n", maximum( number1, number2, number3 ) );
20
21     return 0; /* indica che il programma è terminato con successo */
22
23 } /* fine della funzione main */
24
25 /* Definizione della funzione maximum */
26 /* x, y e z sono parametri */
27 int maximum( int x, int y, int z )
28 {

```

Figura 5.4 La funzione `maximum` definita dal programmatore (continua)

```

29     int max = x; /* si assume che x sia il massimo */
30
31     if ( y > max ) { /* se y è maggiore di max, assegna y a max */
32         max = y;
33     } /* fine del comando if */
34
35     if ( z > max ) { /* se z è maggiore di max, assegna z a max */
36         max = z;
37     } /* fine del comando if */
38
39     return max; /* max è il valore massimo */
40
41 } /* fine della funzione maximum */

```

```

Enter three integers: 22 85 17
Maximum is: 85

```

```

Enter three integers: 85 22 17
Maximum is: 85

```

```

Enter three integers: 22 17 85
Maximum is: 85

```

Figura 5.4 La funzione maximum definita dal programmatore



Buona abitudine 5.7

Includete per tutte le funzioni i relativi prototipi in modo da trarre vantaggio dalle capacità di controllo di tipo implementate nel linguaggio C. Utilizzate le direttive #include del preprocessore per ottenere, dai file di intestazione appropriati, i prototipi delle funzioni incluse nella libreria standard o per ottenere le intestazioni contenenti i prototipi di funzione per le funzioni sviluppate da voi e dai membri del vostro gruppo.

Il prototipo di funzione per la maximum della Figura 5.4 (riga 5) è

```
int maximum( int x, int y, int z ); /* prototipo della funzione */
```

Questo prototipo di funzione stabilisce che maximum riceverà tre argomenti int e restituirà un risultato dello stesso tipo. Osservate che il prototipo della funzione maximum è uguale alla prima riga della sua definizione.



Buona abitudine 5.8

A volte, i nomi dei parametri saranno inclusi nei prototipi di funzione (una nostra preferenza) per scopi documentativi. In ogni modo il compilatore li ignorerà.



Errore tipico 5.8

Dimenticare il punto e virgola alla fine del prototipo di funzione è un errore di sintassi.

Una chiamata di funzione che non corrisponda al suo prototipo è un errore di sintassi. Sarà generato un errore anche qualora il prototipo e la definizione della funzione non concordino. Per esempio, se il prototipo della funzione nella Figura 5.4 fosse stato scritto come

```
void maximum( int x, int y, int z );
```

il compilatore avrebbe generato un errore perché il tipo di ritorno `void`, indicato nel prototipo della funzione, sarebbe stato diverso da quello `int` specificato nell'intestazione della stessa.

Un'altra importante caratteristica dei prototipi di funzione è la *coercizione degli argomenti*, ovvero la conversione forzata degli argomenti al tipo appropriato. Per esempio, la funzione della libreria matematica `sqrt` potrà essere richiamata con un argomento intero, funzionando ancora correttamente anche se il suo prototipo in `math.h` specifica che l'argomento debba essere di tipo `double`. L'istruzione

```
printf( "%.3f\n", sqrt( 4 ) );
```

valuterà correttamente `sqrt(4)` e visualizzerà il valore `2,000`. Il prototipo di funzione fa in modo che il compilatore converta il valore intero `4` in quello di tipo `double` `4,0`, prima che lo stesso sia passato alla `sqrt`. In generale, i valori degli argomenti che non corrispondano precisamente ai tipi dei parametri definiti nel prototipo saranno convertiti in modo appropriato, prima che la funzione sia richiamata. Tali conversioni potrebbero però causare dei risultati scorretti, qualora non fossero rispettate le *regole di promozione* del C. Tali regole specificano in quale modo i vari tipi di dato possano essere convertiti tra loro senza perdita di informazioni. Nel nostro esempio con `sqrt`, un `int` sarà convertito automaticamente in un `double` senza che ne sia modificato il valore. Nondimeno un `double` convertito in `int` troncherebbe la parte frazionaria del valore `double`. Anche convertire i tipi più grandi degli interi in quelli più piccoli (per esempio, un `long` in uno `short`) potrà produrre dei valori modificati.

Le regole di promozione si applicano automaticamente alle espressioni che contengono dei valori di due o più tipi di dato, dette anche espressioni di *tipo misto*. Ogni valore in un'espressione di tipo misto sarà promosso automaticamente a quello "più alto" dell'espressione (in realtà, sarà creata e utilizzata una versione temporanea di ognuno dei valori: quelli originali rimarranno invariati). La Figura 5.5 elenca i tipi di dato nell'ordine da quello più alto a quello più basso, con le relative specifiche di conversione per `printf` e `scanf`.

Tipi di dato	Specifiche di conversione per printf	Specifiche di conversione per scanf
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
short	%hd	%hd
char	%c	%c

Figura 5.5 Gerarchia di promozione per i tipi di dato

Convertire dei dati nei tipi più bassi produce normalmente dei valori scorretti. Di conseguenza, i valori potranno essere convertiti in un tipo più basso solo assegnandoli esplicitamente a una variabile di tipo più basso, o utilizzando l'operatore di conversione. I valori degli argomenti di funzione saranno convertiti ai tipi dei parametri definiti nel relativo prototipo, come se fossero stati assegnati direttamente a variabili di quei tipi. Qualora la nostra funzione `square` che utilizza un parametro intero (Figura 5.3) fosse stata richiamata con un argomento in virgola mobile, questo sarebbe stato convertito in un `int` (un tipo più basso) e `square` avrebbe restituito un valore scorretto. Per esempio, `square(4.5)` restituirebbe 16 invece di 20,25.



Errore tipico 5.9

Convertire un tipo di dato più alto, rispetto alla gerarchia di promozione, in uno più basso potrà modificare il valore dei dati.

Il compilatore formerà un suo prototipo di funzione, qualora questo non sia stato incluso in un programma, utilizzando la prima occorrenza della funzione: ovvero sia, la definizione o una sua chiamata. Per default, il compilatore assumerà che la funzione restituisca un `int` e non presumerà nulla relativamente agli argomenti. Ne consegue che, qualora gli argomenti passati alla funzione non siano corretti, gli errori non saranno individuati dal compilatore.



Errore tipico 5.10

Dimenticare un prototipo provocherà un errore di sintassi, qualora il tipo restituito dalla funzione non sia un `int` e la definizione di quella appaia nel programma dopo una sua chiamata. In caso contrario, dimenticare il prototipo di una funzione potrà causare un errore durante la fase di esecuzione, oppure un risultato inatteso.



Ingegneria del software 5.9

Un prototipo inserito all'esterno di ogni definizione di funzione si applicherà a tutte le relative invocazioni che appariranno nel file dopo il prototipo. Un prototipo inserito all'interno di una funzione si applicherà solo alle chiamate eseguite all'interno della stessa.

5.7 I file di intestazione

Ogni libreria standard ha un corrispondente *file di intestazione* che contiene i prototipi per tutte le funzioni incluse in quella libreria, oltre alle definizioni dei vari tipi di dato e delle costanti necessari a quelle funzioni. La Figura 5.6 elenca alfabeticamente i file di intestazione della libreria standard che potranno essere inclusi nei programmi. Il termine “macro”, usato diverse volte nella Figura 5.6, sarà trattato in dettaglio nel Capitolo 13, Il preprocessore del C.

Il programmatore potrà creare dei file di intestazione personalizzati. Anche i file di intestazione definiti dal programmatore dovranno terminare in `.h` e potranno essere inclusi utilizzando la direttiva del preprocessore `#include`. Per esempio, se il prototipo per la nostra funzione `square` si trovasse nel file di intestazione `square.h`, potremmo includere quest'ultimo nel nostro programma utilizzando la seguente direttiva all'inizio del programma stesso:

```
#include "square.h"
```

La Sezione 13.2 fornirà ulteriori informazioni sull'inclusione dei file di intestazione.