Programmazione in linguaggio C

Atlas

1 Le basi del linguaggio

Il linguaggio C permette di costruire con precisione ed efficacia un programma ben strutturato.

Il **linguaggio C**, ideato da **Dennis M. Ritchie** e da **Brian W. Kernighan** attorno al 1970, è uno dei più importanti linguaggi per la programmazione strutturata ancora oggi utilizzato.

La versione ufficiale del linguaggio C è quella stabilita nel 1983 dalle specifiche dell'ente americano per gli standard ANSI (*American National Standard Institute*). Successive revisioni portarono alla versione del 1989, approvata dall'ente internazionale per gli standard ISO (*International Standardization Organization*) e denominata normalmente come "**ANSI C**" o "ISO C".

Iniziamo con un esempio molto semplice, che ci consente di mettere in evidenza le operazioni più importanti da svolgere per preparare ed eseguire un programma.
Supponiamo di voler scrivere un programma C per calcolare la somma di due numeri.
Il procedimento da seguire consiste nel prendere in ingresso due numeri, che indichiamo con *a* e *b*, sommarli e produrre in uscita il risultato, che indichiamo con *s*: cioè

$$s \leftarrow a + b.$$

a, *b* sono i dati di input, *s* è il dato di output, e l'addizione è l'attività di elaborazione richiesta.

L'algoritmo che risolve il problema in esame è il seguente:

Algoritmo Somma
inizio
    immetti a, b
    calcola s = a + b
    scrivi s
fine

Si noti che nella descrizione del procedimento non si usano numeri, ma lettere che stanno al posto dei numeri, cioè **variabili**: questo consente di scrivere procedimenti di carattere generale, offrendo la possibilità di eseguire più volte lo stesso procedimento con dati diversi.
Per far eseguire questo semplice algoritmo a un computer, occorre indicare la sequenza delle istruzioni usando il linguaggio di programmazione.
Oltre alle parole che fanno parte del linguaggio, in un programma compaiono gli **identificatori**, cioè i nomi che il programmatore assegna alle costanti e alle variabili, e che rappresentano i dati utilizzati nel programma.

In C il procedimento precedente viene scritto in questa forma:

```
/* Somma.c : somma di due numeri */

#include <stdio.h>

main() {
  int a, b, s;

  scanf("%d %d", &a, &b);
  s = a + b;
  printf("%d", s);
}
```

La prima riga è un **commento**, non obbligatorio, solo documentativo e delimitato dalle coppie di caratteri **/*** e ***/**, che indicano l'inizio e la fine del commento. In questo caso è stato scritto il nome con il quale il programma è stato salvato su disco (*Somma.c*), file di testo con l'estensione **.c**, e una breve descrizione delle funzionalità del programma stesso.

Il linguaggio C non possiede apposite istruzioni per la gestione dell'interazione con l'utente, ma si appoggia sulle funzioni delle librerie standard del linguaggio, dette **librerie di I/O** (*input/output*). Il programma che utilizza operazioni di input e output deve quindi contenere all'inizio un'apposita dichiarazione (**direttiva *include***) per includere in esso la libreria di I/O:

```
#include <stdio.h>
```

Il file *stdio.h* è un **file di header** (file di intestazione con estensione **.h**, il cui nome è scritto all'interno di una coppia di parentesi angolari < >), che contiene la descrizione (**prototipo**) delle funzioni di I/O, richiamabili poi da un qualsiasi punto del programma, in particolare le funzioni **scanf** per acquisire i dati dalla tastiera e **printf** per visualizzare i risultati sul video.

Il programma vero e proprio inizia con la parola **main** seguita da una parentesi graffa aperta, che indica l'inizio della sequenza delle istruzioni da eseguire. La sequenza termina con una parentesi graffa chiusa.
Per inserire le parentesi graffe, usando una tastiera che non riporta esplicitamente il tasto, si può usare una tra le seguenti combinazioni di tasti:

{	Alt + 123 (tastierino numerico)	Ctrl + Alt + Shift + [	Shift + AltGr + [
}	Alt + 125 (tastierino numerico)	Ctrl + Alt + Shift +]	Shift + AltGr +]

Subito dopo la parola *main* c'è una coppia di parentesi tonde, perché, come vedremo meglio in seguito, il programma può essere considerato una funzione che può ricevere argomenti: essi, se presenti, vengono indicati tra le parentesi tonde.

La prima riga del programma specifica quali sono le variabili utilizzate nel programma e di che tipo sono (**int**, cioè numeri interi).

Nelle righe successive vengono scritte le istruzioni da eseguire, in sequenza, nell'ordine in cui sono state scritte: **scanf** indica la lettura dei dati dalla tastiera (*input*), l'istruzione s = a + b specifica che il risultato del calcolo a + b deve essere assegnato alla variabile *s*, e infine **printf** ordina la visualizzazione del risultato di *output* (variabile *s*) sullo schermo del computer.

La lettera **f** finale nelle due istruzioni specifica che l'input e l'output seguono un formato specificato (input e output **formattati**): per trattare i dati numerici interi si usa il formato indicato con **%d** (*specificatore di formato*).

Nell'istruzione *scanf*, poi, davanti al nome delle variabili *a* e *b*, è stato scritto il carattere **&** (*ampersand* o *e-commerciale*): esso indica che il valore acquisito dalla tastiera deve essere registrato all'indirizzo delle variabili *a* e *b* in memoria centrale.

Tutte le righe terminano con il *punto e virgola*, come se fossero le frasi di un testo scritto in un linguaggio naturale, ad eccezione delle righe di commento e della dichiarazione di inclusione *#include*.

Un programma C può, quindi, essere ricondotto ad uno schema generale, che comprende principalmente tre sezioni:

- intestazione
- zona delle dichiarazioni
- sezione esecutiva.

Il linguaggio C è un linguaggio **case-sensitive**, nel senso che distingue tra lettere maiuscole e minuscole, per cui la variabile *numero* è diversa dalla variabile *Numero* e dalla variabile *NUMERO*. Nella consuetudine della programmazione nel linguaggio C, si usano normalmente i caratteri minuscoli sia per le parole chiave del linguaggio, sia per gli identificatori e le variabili; i nomi composti sono indicati spesso con l'uso delle maiuscole nelle iniziali (per esempio, *TriangoloRettangolo*).

Vediamo ora più in dettaglio come si rappresentano in un programma C i dati e le istruzioni.
I dati utilizzati all'interno di un programma possono essere:

- **costanti**, se non cambiano il loro valore durante l'esecuzione del programma
- **variabili**, se cambiano il valore.

I dati trattati in un programma possono essere:

- **numerici**, quali età, importi, stipendi, misure
- **alfanumerici** (o **stringhe**), quali nomi, descrizioni, codici.

Sui dati di tipo numerico si possono effettuare le usuali operazioni aritmetiche, che vengono rappresentate con gli **operatori**:

+ per l'addizione
– per la sottrazione
***** per la moltiplicazione
/ per la divisione
% per il calcolo del resto della divisione tra interi.

2 La dichiarazione delle costanti e delle variabili

Le **costanti** utilizzate nel programma vengono elencate attraverso la direttiva **#define**, secondo la seguente sintassi:

#define *nome valore*

con l'identificatore della costante separato con uno spazio dal valore assegnato.

Per esempio:

```
#define PIGRECO 3.14
#define RISPOSTA 's'
#define SIGLAPROV "TO"
```

Si noti che i valori di costanti di tipo carattere, o *stringhe* di caratteri, sono racchiusi tra virgolette (per esempio, la sigla della provincia di Torino, "TO"); la costante formata da un singolo carattere è racchiusa tra apici ('s'). Per i numeri la separazione tra cifre intere e decimali è indicata dal carattere . (punto decimale).
Nella consuetudine della programmazione in C, i nomi delle costanti sono scritte in maiuscolo.

Le **variabili** utilizzate nel programma devono essere dichiarate, all'inizio del programma o comunque prima di essere usate, indicando il tipo e il nome assegnato alla variabile (con il punto e virgola finale):

tipo nome;

Si può anche utilizzare un'unica dichiarazione per diverse variabili dello stesso tipo, separando con la virgola i loro nomi:

tipo nome1, nome2, nome3;

I principali **tipi standard** delle variabili in C sono:

Tipo	Descrizione	Numero di bit utilizzati
short int	numeri interi compresi tra –32768 e +32767	16
int	numeri interi compresi tra -2.147.483.648 e 2.147.483.648	32
long int	numeri interi compresi tra -2.147.483.648 e 2.147.483.648	32
float	Numeri in virgola mobile (*floating point*) in singola precisione	32
double	Numeri in virgola mobile (*floating point*) in doppia precisione	64

Per i dati di tipo alfanumerico, formati da più caratteri (**stringhe**), si utilizza una sequenza di caratteri, indicando dopo il nome della variabile, tra parentesi quadre, il numero massimo previsto per i caratteri della stringa:

char *nome[n];*

Esempi di dichiarazione di variabili:

```
int eta;
int anni;
float statura;
float AreaCerchio;
float AreaTriang;
char risposta;
char nome[30];
char CodiceFiscale[17];
```

L'identificatore di una variabile o di una costante è una sequenza qualsiasi di caratteri alfabetici e cifre, che inizia comunque con una lettera; si può usare anche il segno _ per definire nomi composti, per esempio *area_cerchio*.

È possibile anche, in fase di dichiarazione, assegnare alla variabile un valore iniziale, usando la seguente sintassi:

tipo nome = *valore;*

Per esempio:

```
int contatore = 0;
char nazione[7] = "Italia";
```

Questa operazione si chiama **inizializzazione** di una variabile.

Il numero indicante la lunghezza della stringa *nazione* è aumentato di 1, perché occorre considerare il carattere **terminatore di stringa \0**, che viene aggiunto automaticamente.

I tipi di dati del linguaggio C

La seguente tabella riassume tutti i tipi predefiniti di dati del linguaggio C, per i dati di tipo non numerico e per i tipi numerici, interi e non interi.
La parola **unsigned** indica il tipo senza segno, cioè il tipo che non può assumere valori negativi.

	Tipo	Intervallo	Bit utilizzati
caratteri	**unsigned char**	Da 0 a 255	8 bit
	char	Da -128 a 127	8 bit
interi	**unsigned short int**	Da 0 a 65.535	16 bit
	short int	Da -32.768 a 32.767	16 bit
	int	Da -2.147.483.648 a 2.147.483.647	32 bit
	unsigned long int	Da 0 a 4,294,967,295	32 bit
	long int	Da -2.147.483.648 a 2.147.483.647	32 bit
reali	**float**	Da $1{,}17549435 \cdot 10^{-38}$ a $3{,}40282347 \cdot 10^{+38}$	32 bit
	double	Da $2{,}2250738585072014 \cdot 10^{-308}$ a $1{,}7976931348623157 \cdot 10^{+308}$	64 bit
	long double	Da $3{,}4 \cdot 10^{-4932}$ a $1{,}1 \cdot 10^{4932}$	80 bit

3 Le frasi di commento

All'interno del programma possono essere inserite righe, o paragrafi di più righe, contenenti **commenti** o annotazioni del programmatore, con le quali è possibile documentare il significato delle variabili o delle costanti utilizzate, oppure la funzione svolta da una parte del programma.

Le frasi di commento vengono racchiuse tra i delimitatori **/*** e ***/**.

Tutto quello che viene scritto a partire dal primo delimitatore /* fino al delimitatore */ viene considerato commento, anche se disposto su più righe.

```
int eta;    /* età di una persona */
```

I commenti possono essere collocati in un punto qualsiasi del programma.

4 L'assegnazione dei valori alle variabili

L'istruzione di **assegnazione** ha una sintassi del tipo:

variabile = *valore*;

Il valore viene assegnato alla variabile scritta a sinistra del simbolo =.

Per valore si intende un numero o il risultato di un'espressione.

Per esempio, il calcolo dell'area di un cerchio viene indicato con l'istruzione:

```
area = raggio * raggio * 3.14;
```

Nelle espressioni di tipo numerico si usano gli operatori aritmetici già visti **+**, **–**, *****, **/**, per le quattro operazioni elementari e **%** per il resto della divisione tra interi.

Per esempio:

```
resto = a % b;
velocita = spazio / tempo;
```

Nei linguaggi di programmazione sono, in genere, consentite anche istruzioni del tipo:

```
conta = conta + 1;
```

con la variabile *conta* che compare sia a sinistra che a destra del segno =. La precedente istruzione significa che il nuovo valore di *conta* si ottiene aggiungendo 1 al precedente valore.

Nel linguaggio C si può anche scrivere:

```
conta++;
```

per indicare, con l'**operatore ++**, un incremento unitario del valore di una variabile.

Esiste anche l'**operatore ––** per il decremento unitario:

```
conta--;
```

che è equivalente all'istruzione di assegnazione

```
conta = conta - 1;
```

Per incrementi e decrementi non unitari si possono usare gli operatori **+=** e **–=**.

Per esempio, l'istruzione:

```
a += b;
```

significa che il valore di *a + b* viene memorizzato nella variabile *a* ed è equivalente all'istruzione di assegnazione:

```
a = a + b;
```

L'istruzione

```
a -= b;
```

significa che il valore di *a - b* viene memorizzato nella variabile *a* ed è equivalente all'istruzione di assegnazione:

```
a = a - b;
```

Si possono usare in modo analogo anche gli operatori ***=**, **/=** e **%=** per la moltiplicazione, per la divisione e per il resto della divisione intera.

La tabella seguente riassume le equivalenze tra i vari operatori di assegnamento:

Operatore	Utilizzo	Esempio	Istruzione equivalente
++	Incremento unitario	x++;	x = x + 1;
−−	Decremento unitario	x−−;	x = x − 1;
+=	Incremento	x += y;	x = x + y;
−=	Decremento	x −= y;	x = x − y;
*=	Moltiplicazione	x *= y;	x = x * y;
/=	Divisione	x /= y;	x = x / y;
%=	Resto della divisione intera	x %= y;	x = x % y;

Nelle istruzioni di assegnazione e nelle espressioni di calcolo possono poi comparire le **funzioni**, ossia procedure di calcolo predefinite (*built-in*) del linguaggio che, ricevendo un valore, restituiscono un valore calcolato.

Tra le più usate, possono essere citate le seguenti:

- **pow(x, y)**, per indicare il calcolo della potenza di *x* con esponente *y* (*x* e *y* devono essere di tipo *double*, così come il risultato ottenuto del calcolo).
- **sqrt(x)**, per indicare il calcolo della radice quadrata del numero x
- **ceil(x)**, per indicare il valore di un numero arrotondato all'intero superiore
- **floor(x)**, per ottenere un numero senza parte decimale.

Le ultime due necessitano di un esempio chiarificatore:
ceil(3.7) produce 4
floor(3.7) produce 3.

Per calcolare l'ipotenusa di un triangolo rettangolo, si può scrivere un'istruzione del tipo:

```
ipotenusa = sqrt(pow(cateto1,2.0) + pow(cateto2,2.0));
```

I numeri 2.0 come esponenti dei cateti stanno ad indicare che gli argomenti della funzione *pow* sono di tipo *double*.
Tutte le funzioni illustrate in precedenza appartengono alla libreria delle funzioni matematiche predefinite del linguaggio. Pertanto i programmi che usano queste funzioni devono contenere all'inizio la direttiva

```
#include <math.h>
```

per l'inclusione del file di intestazione **math.h** con i prototipi delle funzioni matematiche.

L'uso delle parentesi (sempre tonde) determina l'ordine di esecuzione delle operazioni indicate in un'espressione aritmetica, come nell'algebra elementare:

```
a = (b + c) * (d - e);
```

Si noti che nel linguaggio di programmazione non è possibile indicare le frazioni con la consueta notazione:

$$x = \frac{(a - b)^2}{c * d}$$

Questa frazione deve essere indicata in un'istruzione di assegnazione nel seguente modo:

```
x = pow((a-b),2.0)/(c*d);
```

Nelle istruzioni di assegnazione, occorre fare attenzione all'uso dei tipi per le variabili, considerando i problemi di incompatibilità tra tipi che si possono verificare.
Per esempio, se si hanno:

```
int a;
char c[10];
```

le istruzioni di assegnazione a = c e c = a rappresentano entrambe un errore di sintassi.
Questi tipi di errore vengono segnalati con un messaggio di incompatibilità di tipi (in inglese, *incompatibile types in assignment*).

Il casting per la conversione di tipo

Per **conversione di tipo** si intende un'operazione che trasforma un valore di un certo tipo in un valore di altro tipo.

Questa operazione è molto comune in tutti i linguaggi, anche se spesso il programmatore non se ne rende conto. Per esempio, quando in un'espressione si utilizzano dati dello stesso tipo, il tipo del risultato di calcolo è ben definito ed è lo stesso degli operandi. Il problema nasce quando in un'operazione gli operandi sono di diverso tipo.
Per esempio, nel calcolo dell'area del seguente frammento di codice, si tratta di decidere di che tipo sia il valore della variabile *area*:

```
int base;
float altezza;

area = base * altezza ;
```

Non sempre una conversione di tipo preserva il valore: per esempio nella conversione da *float* a *int* in generale si riscontra una perdita di precisione. In effetti in una conversione da *float* a *int* il compilatore non fa altro che scartare la parte frazionaria; se poi il valore non è rappresentabile dal tipo *int* il risultato è indefinito.
È opportuno sottolineare che le operazioni di conversione riguardano il valore della variabile in quel particolare contesto e non il tipo della variabile stessa.
Le conversioni eseguite direttamente dal compilatore si chiamano **conversioni implicite**.
Per esempio, si consideri il programma seguente nel quale un valore non intero viene assegnato ad una variabile di tipo *int*.

```
#include <stdio.h>

main() {
  int a = 5;
  float b = 3.56;

  a = b;
  printf("%d",a);

}
```

L'output del programma è 3.

Nel seguente programma, invece, un valore intero è assegnato ad una variabile di tipo *float*.

```
#include <stdio.h>

main() {
  int a = 5;
  float b = 3.56;

  b = a;
  printf("%f",b);

}
```

In questo caso l'output del programma è 5.0.

I caratteri **%f** nell'istruzione *printf* indicano lo *specificatore di formato* per i numeri reali (*float*).

Il linguaggio C consente al programmatore stesso di richiedere le conversioni di tipo: in questo caso si parla di **conversioni esplicite** (cioè decise dal programmatore) o *casting*.

> Il **casting** è l'azione che consente di trasformare la rappresentazione di un dato dal suo tipo originale a un altro.

L'operatore **cast**, rappresentato da una coppia di parentesi tonde che racchiudono il tipo di dato, è un operatore a tutti gli effetti e ha la seguente sintassi:

(tipo) espressione;

Per esempio:

```
b = (int) a;
```

I due programmi seguenti evidenziano l'importanza del casting nelle operazioni con numeri di tipo diverso.

```
#include <stdio.h>

main() {
  int a = 5;
  int b = 2;
  float c = 3.56;
  float p;

  p = a/b + c;
  printf("%f",p);

}
```

```
#include <stdio.h>

main() {
  int a = 5;
  int b = 2;
  float c = 3.56;
  float p;

  p = (float)a/b + c;
  printf("%f",p);

}
```

Con il programma di sinistra si ottiene in output 5.56, perché la divisione intera a/b produce il valore 2 che viene poi sommato a 3.56.
Con il programma di destra si ottiene invece in output il valore 6.06, perché il casting, applicato alla divisione a/b, produce correttamente il valore 2.5 che viene poi sommato a 3.56.

5 Gli operatori di relazione e logici

Nel programma C, gli **operatori di confronto** si indicano con i simboli

== per uguale
> per maggiore
< per minore
<= per minore o uguale
>= per maggiore o uguale
!= per diverso.

Con questi simboli si possono rappresentare espressioni booleane del tipo:

```
a >= c
b < d
f != g
```

I **connettivi logici** che possono essere utilizzati in C sono:

&& indica l'operazione di congiunzione (*And*)
|| indica l'operazione di disgiunzione (*Or*)
! indica la negazione (*Not*).

Utilizzando poi le operazioni di congiunzione, disgiunzione e negazione, si possono costruire espressioni booleane del tipo:

```
(a > b) && (c == 3)
(a == 5) || (a == 7)
!(a > 4)
```

Il valore di queste espressioni può essere 0 (Falso) oppure 1 (Vero).

6 Le istruzioni di ingresso e uscita

L'istruzione per acquisire dati dalla tastiera (*input*) si indica in C con **scanf**. La lettera *f* finale indica che l'acquisizione dei dati è fatta secondo un formato (*input formattato*).

Per esempio:

```
scanf("%d", &raggio);
```

fa entrare da tastiera un numero intero e lo assegna alla variabile di nome *raggio*. L'istruzione ha due argomenti, scritti tra le parentesi tonde e separati dalla virgola: il primo, tra virgolette, indica il formato; il secondo contiene la variabile il cui valore viene acquisito dalla tastiera.

Come già visto nel primo esempio, per il calcolo della somma di due numeri, il nome della variabile è preceduto dal carattere **&**, perché l'istruzione *scanf* richiede come argomento l'indirizzo della variabile e non il nome della variabile. Questo vale per i dati di tipo numerico, intero e reale, e per i caratteri; invece per le stringhe formate da più di un carattere non si deve indicare il carattere **&**.

I formati sono rappresentati con sequenze di caratteri, dette **specificatori di formato**: essi iniziano con il carattere **%**, seguito da uno o più caratteri.

Gli specificatori di uso più comune sono i seguenti:
%d dati numerici interi
%f dati numerici in virgola mobile (*float* e *double*)
%c dati formati da un solo carattere
%s dati stringa.

L'istruzione *scanf* indica quindi un'operazione sullo **standard input**, cioè l'unità standard del computer per acquisire dati dall'esterno (tastiera).
L'operazione di lettura con *scanf*, permette di leggere con un'unica operazione tutti i caratteri digitati fino alla pressione del tasto *Invio*. Se si vogliono leggere più dati, si devono indicare, nelle parentesi di *scanf*, tanti specificatori di formato e tante variabili quanti sono i dati da inserire:

```
scanf("%d %d", &base, &altezza);
```

L'utente del programma può inserire i dati richiesti sulla stessa riga separandoli con la barra spaziatrice o con il ritorno a capo.

L'istruzione per visualizzare dati e messaggi sul video (*output*) si indica in C con **printf**. La lettera *f* finale indica che la visualizzazione è ottenuta specificando un formato (*output formattato*). Per esempio:

```
printf("%f", area);
```

scrive sul video il valore numerico in virgola mobile contenuto nella variabile *area*. L'istruzione *printf* utilizza gli stessi specificatori di formato visti per *scanf*.
L'istruzione *printf* indica quindi un'operazione sullo **standard output**, cioè l'unità standard del computer per comunicare i risultati (video).
Eventuali messaggi, utili per illustrare i risultati, devono essere inseriti tra le virgolette, insieme agli specificatori di formato, nella posizione dove devono comparire. Per esempio, se la variabile *area* possiede il valore 10, l'istruzione:

```
printf("Risultato del calcolo = %f", area);
```

produce in output la seguente riga:

```
Risultato del calcolo = 10.0
```

È possibile impostare il numero delle posizioni utilizzate per rappresentare il risultato e il numero delle cifre dopo il punto decimale. I valori delle impostazioni sono indicati prima dei caratteri dello specificatore di formato. Per esempio:

```
printf("Risultato del calcolo = %15.2f", area);
```

visualizza il valore di *area* allineato a destra all'interno di 15 posizioni e con 2 cifre decimali. L'output è il seguente:

```
Risultato del calcolo =   10.00
```

L'istruzione *printf* visualizza il messaggio e il valore di *area* lasciando il cursore sulla stessa riga del video; l'output di una successiva istruzione *printf* verrà visualizzato subito dopo. Per portare il cursore all'inizio della riga successiva del video (cioè per andare a capo nella visualizzazione), occorre aggiungere all'interno delle virgolette, alla fine della specificazione del formato, i due caratteri **\n**; l'output successivo comparirà all'inizio di una nuova riga.
Per esempio:

```
printf("%f \n", area);
```

Per facilitare l'inserimento dei dati da parte dell'utente del programma, è buona norma far precedere all'istruzione *scanf* un'istruzione *printf*, che fa comparire un messaggio, per ricordare quali dati occorre inserire, di quale tipo (numeri o stringhe) ed eventualmente in quale formato.

```
printf("Introduci il lato: ");
scanf("%d", &lato);
```

Se si vuole stampare sulla carta della **stampante** anziché sul video, occorre utilizzare, anziché l'istruzione *printf*, l'analoga **fprintf** (stampa formattata su file), che funziona con le stesse modalità di *printf*. Occorre solo specificare il nome della stampante, **stdprn** (stampante standard o predefinita), per indicare la ridirezione dell'output dal video alla stampante (da *stdout*, standard output o video, a stampante).
Per esempio:

```
fprintf(stdprn,"Totale importi =  %f \n ", Tot);
```

Nella fase di testing di un programma che richiede l'output su stampante oppure nel caso in cui la stampante non fosse disponibile, si può sostituire *stdprn* con **stdout**, per ottenere i risultati sul video, anziché sulla carta della stampante.
Per l'input e output standard di singoli caratteri, il linguaggio C possiede anche le funzioni **getchar** per l'acquisizione e **putchar** per la visualizzazione.
Per le stringhe le funzioni sono **gets** e **puts**. Queste funzioni sono di uso meno frequente rispetto a *scanf* e *printf* per l'input e l'output formattati.
Occorre ricordare, infine, che tutte queste funzioni appartengono alla libreria delle funzioni standard per l'input e output dei programma: pertanto i programmi che utilizzano queste funzioni devono contenere all'inizio, come già visto negli esempi precedenti, la direttiva

```
#include <stdio.h>
```

che indica appunto l'inclusione del file di intestazione **stdio.h** per le operazioni di I/O standard.

Input e output formattati

La tabella seguente riassume il significato degli **specificatori di formato** più importanti:

Specificatore di formato	Tipo di dato
%d	Intero decimale
%i	Intero decimale
%c	Carattere singolo
%s	Stringa di caratteri
%o	Numero ottale
%x	Numero esadecimale
%u	Intero senza segno
%f	Numero reale (*float*)
%lf	Numero reale (*double*)
%e	Formato scientifico
%%	Per visualizzare il carattere % stesso

Nella specificazione del formato, dopo il carattere % e prima della lettera che identifica il formato, si possono aggiungere altre opzioni di formato; nell'ordine:

- il segno – per indicare l'allineamento a sinistra del dato; oppure
- uno zero (0) per indicare che le posizioni a sinistra, occupate dal numero da stampare, devono essere riempite con zeri;
- un numero intero per impostare il numero di posizioni all'interno delle quali il valore deve essere visualizzato; oppure
- due numeri, separati dal punto, per indicare il numero di posizioni e il numero di cifre decimali della rappresentazione.

Per esempio, se il valore della variabile *area* è 10, la seguente istruzione:

```
printf("Risultato del calcolo = %-15.2f", area);
```

produce sul video il seguente output con il numero allineato a sinistra, all'interno di 15 posizioni, con 2 cifre decimali:

Risultato del calcolo = 10.00

I seguenti esempi servono a chiarire meglio le modalità di utilizzo delle specificazioni di formato e gli output che si ottengono. La colonna di destra si riferisce alla visualizzazione delle variabili *a, b, nazione,* in corrispondenza delle istruzioni indicate nella colonna di sinistra:

```
int a = 5;
float b = 3.4567;
char nazione[7] = "Italia";
```

Istruzioni	Output
`printf("%d \n", a);`	`5`
`printf("%15d \n", a);`	`              5`
`printf("%015d \n", a);`	`000000000000005`
`printf("%f \n", b);`	`3.456700`
`printf("%15.2f \n", b);`	`           3.46`
`printf("%10d %15.2f \n", a, b);`	`         5           3.46`
`printf("%-10d %-15.2f \n", a, b);`	`5          3.46`
`printf("%s \n", nazione);`	`Italia`
`printf("%20s \n", nazione);`	`              Italia`
`printf("%-20s \n", nazione);`	`Italia`
`printf("%-20s %15.2f \n", nazione, b);`	`Italia                    3.46`

Si noti l'uso dei caratteri **\n** per specificare il ritorno a capo, come già visto in precedenza. Questi due caratteri si chiamano *sequenza di escape*.

La tabella seguente presenta l'elenco completo delle **sequenze di escape** utilizzate dal linguaggio C:

Sequenza di escape	Descrizione	Terminologia inglese
\n	A capo riga	*new line*
\t	Tabulazione	*tab*
\r	Ritorno a capo della stessa riga	*carriage return*
\"	Doppi apici	*double quote*
\\	Barra contraria	*backslash*
\b	Una battuta indietro	*backspace*
\'	Apice singolo	*single quote*
\?	Punto di domanda	*question mark*
\a	Segnalazione acustica	*bell*
\0	Fine stringa	*end of string*
\f	Salto pagina	*form feed*

ESEMPIO: Calcolo del valore in dollari corrispondente al cambio di una cifra in euro

Si deve costruire un programma che richieda all'utente l'importo in euro da cambiare e il cambio dollaro/euro, e che comunichi il corrispondente valore in dollari.

PROGRAMMA C

```
/* Cambio.c : cambio di denaro da euro a dollari */

#include <stdio.h>

main() {
   /* input */
   float euro, cambio;
   /* output */
   float dollari;

   printf("Importo in euro? ");
   scanf("%f", &euro);
   printf("Cambio dollari/euro? ");
   scanf("%f", &cambio);
   dollari = cambio * euro;
   printf("%-10s %-10s %-10s \n", "Euro", "Cambio", "Dollari");
   printf("%-10.2f %-10.4f %-10.2f \n", euro, cambio, dollari);
}
```

Il programma mostra esempi di utilizzo degli output formattati per visualizzare in modo più ordinato i risultati dell'elaborazione. Le variabili utilizzate sono state dichiarate di tipo *float*, in quanto sia i dollari sia gli euro sono numeri con i decimali e perché il valore del cambio può avere una parte decimale.

ESERCIZI

1. Calcolare il doppio di un numero fornito da tastiera.
2. Dato il lato, trovare il perimetro e l'area di un quadrato.
3. Dato il raggio, calcolare la circonferenza e l'area del cerchio.
4. Scrivere il programma per scambiare il contenuto di due variabili (occorre usare una terza variabile temporanea).
5. Calcolare la misura dell'ipotenusa di un triangolo rettangolo, noti i cateti.
6. Date le età di tre persone, calcolare l'età media delle persone (somma delle età diviso 3).
7. In quali casi il seguente programma determina un errore di esecuzione nel calcolo della radice quadrata?

```
main() {
   int x;
   float y;

   scanf("%d", &x);
   y = sqrt(x);
   printf("%f", y);
}
```

8. Individuare nel seguente programma le istruzioni che non sono corrette.

```
#includere <stdio.h>

main() {
  /* input
  int i;

  printf(Introduci un numero);
  scanf("%d", i);
  i*=2;
  scrivi("%d \n",i)
}
```

9. Scrivere un programma che richieda in input le coordinate di due punti e scriva le coordinate del punto medio del segmento che unisce i due punti.
10. Scrivere un programma che richieda in input le coordinate di due punti e che scriva l'equazione della retta passante per essi (va escluso, a priori, che la retta risulti verticale).
11. Dato il prezzo di un prodotto e la percentuale di sconto, calcolare il prezzo scontato.
12. Dato in input il valore di un deposito bancario e il tasso di interesse annuo, calcolare gli interessi maturati dopo 25 giorni.
13. Scrivere un programma che legga due numeri interi e che comunichi il risultato della divisione tra i due numeri presentando 3 cifre intere e 5 decimali.
14. Dato il raggio, calcolare la circonferenza e l'area del cerchio. Presentare i risultati con 3 cifre decimali.
15. Dati in input la descrizione, la quantità, il prezzo unitario di un articolo venduto e l'aliquota IVA, comunicare in output la descrizione e il prezzo totale aumentato dell'IVA, visualizzando i dati in modo formattato.
16. Scrivere un programma che richieda in input il nome dell'utente e scriva sul video il messaggio «Buongiorno,» seguito dal nome fornito.

7 Le strutture di controllo

Le istruzioni di un algoritmo, che rappresenta il procedimento risolutivo di un problema, sono organizzate secondo schemi classificabili in tre costrutti fondamentali, chiamati **strutture di controllo**.

- Istruzioni organizzate in **sequenza**.
- Istruzioni che vengono eseguite in **alternativa** con altre.
- Istruzioni che devono essere eseguite in **ripetizione**.

Verranno ora descritti semplici esempi di programmi in linguaggio C in cui sono utilizzate le tre strutture di base.

La **sequenza** si rappresenta costruendo un blocco di istruzioni, ognuna terminante con il punto e virgola, delimitato all'inizio e alla fine da una coppia di parentesi graffe aperta e chiusa **{** … **}**. Per esempio, la parte di programma C che calcola la differenza tra due numeri viene codificata in questo modo:

```
#include <stdio.h>

main() {
  int num1, num2, differenza;

  printf("Due numeri: ");
  scanf("%d %d", &num1, &num2);
  differenza = num1 - num2;
  printf("risultato = %d",differenza);
}
```

Le istruzioni comprese tra le due parentesi graffe possono comparire all'interno di un'altra struttura di controllo (selezione o ripetizione), oppure possono essere le istruzioni che compongono la parte esecutiva del programma C.

ESEMPIO: Calcolare lo sconto del 20% sul prezzo di un articolo

Occorre acquisire in input la descrizione dell'articolo e il suo prezzo. Viene calcolato lo sconto e sottratto dal prezzo iniziale. Da ultimo si devono scrivere la descrizione dell'articolo e il nuovo prezzo scontato.

PROGRAMMA C

```
/* CalcSconto.c : calcolo del prezzo scontato */

#include <stdio.h>

main() {
  /* input-output */
  char descrizione[20];
  float prezzo;
  /* lavoro */
  float sconto;

  printf("Descrizione e prezzo:  ");
  scanf("%s %f", descrizione, &prezzo);
  sconto = (float) prezzo * 20 /100;
  prezzo = prezzo - sconto;
  printf("%-20s %-10.2f \n", descrizione, prezzo);
}
```

Si noti che le variabili *prezzo* e *descrizione* sono sia di input sia di output.

8 La struttura di alternativa

Per la **selezione** nel linguaggio C si usa l'istruzione **if** che ha la sintassi seguente:

```
if (condizione)
  istruzioni-a;
else
  istruzioni-b;
```

Si noti che la condizione è racchiusa tra due parentesi tonde.

Se la condizione è vera, vengono eseguite le *istruzioni-a*, altrimenti vengono eseguite le *istruzioni-b*.

ESEMPIO: Scrivere in ordine crescente due numeri

Si tratta di scegliere tra due numeri il minore e di scrivere prima il minore e poi il maggiore.

PROGRAMMA C

```
/* Ordina.c : due numeri in ordine crescente */

#include <stdio.h>

main() {
  /* input */
  int a, b;

  printf("Due numeri: ");
  scanf("%d %d",&a, &b);
  if (a < b) {
     printf("%d \n",a);
     printf("%d \n",b);
  }
  else {
     printf("%d \n",b);
     printf("%d \n",a);
  }
}
```

Nel caso in cui la condizione $a < b$ sia verificata, devono essere eseguite in sequenza le istruzioni

```
     printf("%d \n",a);
     printf("%d \n",b);
```

Pertanto, in accordo con quanto detto nel paragrafo precedente, le due istruzioni sono state raggruppate tra due parentesi graffe.

Analoghe considerazioni vanno fatte per la sequenza

```
     printf("%d \n",b);
     printf("%d \n",a);
```

Nel caso in cui dopo la condizione o dopo *else* ci sia un'unica istruzione, le parentesi graffe possono essere omesse:

```
if (condizione) istruzione-a;
else istruzione-b;
```

Se si ha una selezione a una sola via, si omette il ramo *else*:

```
if (condizione) istruzione-a;
```

La condizione è un'espressione booleana di cui viene valutata la verità: vengono quindi utilizzati i segni del confronto: ==, <, >, >=, <=, !=, e gli operatori booleani && (*And*), || (*Or*), ! (*Not*) per costruire espressioni logiche combinando tra loro più condizioni.

Per esempio, nel seguente frammento di programma, la struttura di selezione:

```
if (classe == 5 && anni > 18) printf("%s \n", nome);
```

produce in output il nome di uno studente, soltanto nel caso in cui si verifichino entrambe le condizioni, perché viene usato l'operatore && (*And*): frequenta la 5.a classe *e* ha più di 18 anni di età.

9 La ripetizione

In C la **struttura di ripetizione** si rappresenta con l'istruzione **do ... while**:

```
do {
  istruzioni;
} while (condizione);
```

Le istruzioni comprese tra *do* e *while* vengono ripetute tante volte, mentre la condizione scritta dopo *while* si mantiene vera: in altre parole la ripetizione termina quando la condizione scritta dopo *while* diventa falsa.

Per esempio, si consideri il seguente frammento di programma:

```
do {
  printf("il mese:");
  scanf("%d",&mese);
} while (mese < 1 || mese > 12);
```

Solo quando il mese digitato dall'utente è compreso tra 1 e 12, l'esecuzione del programma procede con l'istruzione successiva a *while*.
Si noti che le istruzioni da ripetere sono raggruppate da una coppia di parentesi graffe e che la condizione scritta dopo *while* è racchiusa tra le parentesi tonde. La condizione è composta e utilizza l'operatore || (*Or*).

ESEMPIO: Calcolo del prodotto tra interi utilizzando la sola operazione di somma

Presi in considerazione due numeri interi positivi, si tratta di sommare il primo numero con se stesso un numero di volte pari al secondo numero. Per sapere quante volte deve essere eseguita l'operazione di somma e per sapere quando fermarsi, ogni volta che viene fatta la somma si decrementa di 1 il valore del secondo numero e si interrompe la ripetizione della somma quando questo diventa 0.

PROGRAMMA C

```
/* Prodotto.c : prodotto di due numeri */

#include <stdio.h>

main() {
  /* input */
  int a, b;
  /* output */
  int prod = 0;

  printf("Due numeri positivi: ");
  scanf("%d %d",&a, &b);
  do {
    prod += a;
    b--
  } while (b > 0);
  printf("prodotto = %d \n",prod);
}
```

ESEMPIO: Dato un elenco di persone, con l'indicazione per ciascuna di esse del nome e dell'età, contare i maggiorenni

Si deve organizzare un programma che, per ogni persona dell'elenco, ripeta la richiesta dei dati (nome ed età), controlli se l'età è maggiore o uguale a 18 anni e, in questo caso, incrementi il contatore dei maggiorenni. Alla fine della ripetizione viene scritto il numero dei maggiorenni registrato nel contatore. Dopo la richiesta dei dati di una persona e il controllo sull'età, viene posta all'utente la domanda se l'elenco è finito. Si ripetono le operazioni finché la risposta è uguale a 1 per 'sì'; la ripetizione continua se la risposta è 0 per 'no'.

PROGRAMMA C

```
/* Elenco1.c : elenco di persone */

#include <stdio.h>

main() {
  /* input */
  char nome[20];
  int eta;
  int risp;
  /* output */
  int conta = 0;

  do {
    printf("Nome: ");
    scanf("%s", nome);
    printf("Eta': ");
    scanf("%d", &eta);
    if (eta >= 18)
       conta++;
    printf("elenco finito: 0=no, 1=si ? ");
    scanf("%d", &risp);
  } while (risp == 0);
  printf("I maggiorenni sono = %d \n",conta);
}
```

10 La ripetizione precondizionale

Nel linguaggio C la **ripetizione precondizionale**, cioè con controllo iniziale della condizione, si rappresenta con l'istruzione **while**:

```
while (condizione) {
  istruzioni;
}
```

Mentre la *condizione* si mantiene vera, viene eseguito il blocco di istruzioni racchiuso tra le parentesi graffe.
Per esempio se si vuole far entrare da tastiera un elenco di numeri, segnalando la fine dell'elenco con il numero 0, si può utilizzare il seguente frammento di programma C.

```
printf("inserire un numero (0=fine): ");
scanf("%d", &numero);
while (numero != 0) {
  ..........;
  ..........;
  printf("inserire un altro numero (0=fine): ");
  scanf("%d", &numero);
}
```

Se, invece, si vuole inserire un elenco di nomi, usando il carattere * per segnalare la fine dell'elenco, si può scrivere:

```
char nome[20];

printf("Nome (*=fine): ");
scanf("%s", nome);
while (nome[0] != '*') {
  ..........;
  ..........;
  printf("Nome (*=fine): ");
  scanf("%s", nome);
}
```

Si deve osservare che, poiché la variabile *nome* è una stringa di 20 caratteri, il confronto del carattere *, che segnala la fine dell'inserimento, deve essere fatto con il primo carattere della stringa (carattere della posizione 0 della stringa) indicato con *nome[0].*

ESEMPIO: Divisione tra interi usando le sottrazioni successive

Presi in considerazione due numeri interi, si tratta di sottrarre al primo numero il secondo e di controllare quando il primo numero diventa minore del secondo. Le sottrazioni ripetute vanno contate e il numero di queste fornisce il quoziente. La parte rimanente del primo numero dopo le sottrazioni ripetute è il resto.

PROGRAMMA C

```
/* Divisione.c : divisione tra interi
                 con sottrazioni successive */
```

```c
#include <stdio.h>

main() {
  /* input */
  int a, b;
  /* output */
  int quoz;    /* quoziente della divisione tra interi */

  printf("Due numeri: ");
  scanf("%d %d", &a, &b);
  quoz = 0;
  while (a >= b) {
    a -= b;
    quoz++;
  }
  printf("quoziente = %d \n",quoz);
  printf("resto = %d \n",a);
}
```

ESEMPIO: Modificare il programma presentato nel paragrafo 9, per contare i maggiorenni presenti in un elenco di persone, controllando la ripetizione con una struttura di ripetizione precondizionale

Viene riproposto il problema del paragrafo 9 per mostrare come si presenta la soluzione quando si utilizza la struttura di ripetizione precondizionale.
Alla richiesta dei dati di ogni persona viene suggerito all'utente di segnalare la fine dell'elenco digitando un carattere speciale (per esempio un asterisco) al posto del nome. Il nome della prima persona viene richiesto all'esterno del ciclo e le operazioni di scansione dell'intero elenco sono eseguite *mentre* il nome è diverso dal carattere speciale (asterisco).

PROGRAMMA C

```c
/* Elenco2.c : elenco di persone */

#include <stdio.h>

main() {
  /* input */
  char nome[20];
  int eta;
  /* output */
  int conta = 0;

  printf("Nome (* = fine): ");
  scanf("%s", nome);
  while (nome[0] != '*') {
    printf("Eta': ");
    scanf("%d", &eta);
    if (eta >= 18)
       conta++;
    printf("Nome (* = fine): ");
    scanf("%s", nome);
  }
  printf("I maggiorenni sono = %d \n",conta);
}
```

11 La ripetizione con contatore

Nel linguaggio C, la **struttura di ripetizione con contatore** (o *enumerativa*) viene rappresentata con la struttura **for** nel modo seguente:

```
for (i=inizio; i<=fine; i++) {
  istruzioni;
}
```

La notazione *i++* indica l'incremento unitario per il contatore.

Le istruzioni da eseguire in modo iterato possono essere una o più istruzioni. In presenza di più istruzioni è necessario racchiuderle tra una coppia di parentesi graffe.

Le istruzioni vengono ripetute tante volte, quante ne occorrono per portare il valore del contatore dal valore iniziale al valore finale.
Il contatore deve essere una variabile dichiarata di tipo *int*.

Per esempio, il frammento di programma

```
for(i=1;i<=20;i++) {
 scanf("%d", &numero);
 printf("%d \n", numero*2);
}
```

rappresenta un'iterazione per ottenere il doppio di 20 numeri inseriti da tastiera.

La struttura di ripetizione con contatore è una struttura derivata dalla struttura fondamentale di ripetizione, nel senso che il controllo della ripetizione delle istruzioni mediante un contatore può essere realizzato in modo equivalente usando una struttura di ripetizione postcondizionale *do … while* o una struttura precondizionale *while*, come si vede dal seguente esempio.

Queste tre strutture sono funzionalmente equivalenti:

struttura **for**

```
for(i=1;i<=n;i++) {
  printf("xxxxxxxxx \n");
}
```

struttura **do ... while**

```
i = 1;
do {
   printf("xxxxxxxxx \n");
   i++;
} while (i<=n);
```

struttura **while**

```
i = 1;
while (i<=n) {
   printf("xxxxxxxxx \n");
   i++;
}
```

La struttura *for*, nel caso di ripetizione con un numero prefissato di volte, risulta più compatta e più semplice da rappresentare rispetto alla struttura *while*, in quanto dentro il *for* sono rappresentati l'assegnazione del valore iniziale del contatore e l'incremento del contatore stesso.

Il programma seguente calcola il doppio dei primi 30 numeri naturali:

```
#include <stdio.h>

main() {
  /* input */
  int i;

  for(i=1;i<=30;i++) {
   printf("%d \n", i*2);
  }
}
```

Il programma seguente produce su video la tavola pitagorica:

```
#include <stdio.h>

main() {
  /* input */
  int r,c;

  for(r=1;r<=10;r++) {
     for(c=1;c<=10;c++)
      printf("%5d ",r*c);
     printf("\n");
  }
}
```

r indica il numero di riga, *c* il numero di colonna. I prodotti di ciascuna *r* per i diversi *c* vengono visualizzati sulla stessa riga all'interno di 5 posizioni di stampa (specificatore di formato *"%5d"*). Alla fine di ogni riga c'è un ritorno a capo con l'istruzione *printf("\n")*.
Il programma mostra l'uso di un ciclo *for* interno con contatore *c*, e di un ciclo *for* esterno con contatore *r*.
Per ogni valore di *r* vengono eseguite due istruzioni *for* e *printf("\n")*; per questo motivo occorre usare le parentesi graffe.
Per ogni valore di *c* viene invece eseguita la sola istruzione *printf* e pertanto non occorrono le parentesi graffe.

La struttura di scelta multipla

Nel linguaggio C la **struttura di scelta multipla** è realizzata dall'istruzione **switch** e ha il seguente formato:

```
switch(variabile) {
case valore-1:
   istruzioni-1;
   break;
case valore-2:
   istruzioni-2;
   break;
. . . . . .
```

```
case valore-n:
   istruzioni-n;
   break;
default:
   istruzioni;
   break;
}
```

Dopo la parola *switch*, tra parentesi tonde, è indicato il nome della variabile (**selettore**) di cui si deve controllare il valore per decidere quale strada seguire tra quelle possibili.
Accanto ai valori previsti, per ogni **case**, devono essere scritti l'istruzione o il blocco di istruzioni da eseguire nel caso che la variabile assuma quei valori.
Se nessuno dei valori elencati corrisponde al valore di variabile, vengono eseguite le istruzioni scritte dopo **default**.
Di solito, uno solo dei casi tra quelli previsti deve essere eseguito: è possibile quindi interrompere il controllo degli altri casi inserendo l'istruzione **break** che provoca l'uscita dalla struttura *switch*.
La struttura di selezione multipla è una **struttura derivata** dalla struttura fondamentale di selezione binaria, in quanto potrebbe essere rappresentata utilizzando una successione di controlli annidati del tipo *if (condizione) ... else ...*

Per esempio:

```
if (variabile == 1)
   istruzioni1;
else
      if (variabile == 2)
         istruzioni2;
      else
            if (variabile == 3)
               istruzioni3;
            else
               istruzioni4;
```

ESEMPIO: Sconto progressivo nella vendita di prodotti

Per la vendita di un prodotto si deve applicare uno sconto progressivo in base al numero dei pezzi ordinati secondo la tabella:

Pezzi	Sconto
fino a 3	5%
fino a 5	10%
fino a 10	20%
più di 10	30%

Le informazioni necessarie per risolvere il problema sono: il numero dei pezzi e il prezzo dell'articolo. In base al numero dei pezzi si determina la percentuale dello sconto e si calcola l'importo totale scontato.

PROGRAMMA C

```
/* ScontoProgr.c : sconto progressivo sui prodotti */

#include <stdio.h>

main() {
  /* input */
  float prezzo;       /* prezzo del prodotto   */
  int pezzi;          /* pezzi acquistati      */
  /* output */
  float importo;      /* importo da pagare     */
  /* lavoro */
  int sconto;         /* percentuale di sconto */

  printf("Pezzi acquistati: ");
  scanf("%d", &pezzi);
  printf("Prezzo del prodotto: ");
  scanf("%f", &prezzo);
  switch(pezzi) {
  case 1:
  case 2:
  case 3:
     sconto = 5;
     break;
  case 4:
  case 5:
     sconto = 10;
     break;
  case 6:
  case 7:
  case 8:
  case 9:
  case 10:
     sconto = 20;
     break;
  default:
     sconto = 30;
     break;
  }
  importo = (float)pezzi * prezzo * (100-sconto)/100;
  printf("Importo da pagare = %10.2f \n", importo);
}
```

Si noti che le stesse istruzioni possono essere eseguite in corrispondenza di più valori possibili per la variabile: i diversi valori sono elencati con la parola *case* uno di seguito all'altro.

ESERCIZI

Struttura sequenziale

1. Scrivere il programma che, ricevuto in input un numero intero positivo L, calcoli il perimetro e l'area di un quadrato di lato L.

2. Scrivere il programma che, letto un numero intero, calcoli e scriva la sua metà.

3. Dato il raggio del cerchio di base di un cilindro e la sua altezza, calcolare il peso del cilindro sapendo che la sostanza di cui è composto ha peso specifico = 3.4 g/cm³ (Volume = area della base x altezza; Peso = Volume x Peso specifico).

4. Data la parabola di equazione $y = a\ x^2 + b\ x + c$, determinare le coordinate del vertice V (x_V, y_V), ricordando che la sua ascissa si ottiene con la formula $x_V = - b / (2 * a)$.

Struttura di alternativa

5. Una nuova compagnia telefonica ha promosso l'offerta "oltre80": alla cifra fissa di 0,10 euro (costo alla risposta) occorre aggiungere la cifra di 0,15 euro per ogni secondo del tempo della telefonata; però oltre gli 80 secondi la tariffa per ogni secondo è di 0,09. Fornito da tastiera il numero dei secondi della telefonata, visualizzare il costo totale della chiamata. (Le tariffe indicate nel testo del problema possono diventare più opportunamente costanti).

6. Conoscendo il risultato di una partita di calcio ad eliminazione diretta (non è consentito il pareggio), si vuole visualizzare il nome della squadra vincente.

7. Dati i nomi di tre studenti e la distanza in Km. da casa a scuola, visualizza il nome degli studenti che abitano in comuni distanti più di 30 Km.

8. Su un reddito si paga l'imposta del 35% se il reddito è superiore a 60.000 euro, altrimenti il 23%. Quanto si deve pagare?

Struttura di ripetizione

9. Costruire un programma che accetta da tastiera dei numeri finché viene inserito il valore 0; man mano li visualizza.

10. Costruire un programma che accetta da tastiera dei nomi e li conta. L'inserimento si interrompe quando il numero dei nomi inseriti diventa 10.

Struttura di ripetizione precondizionale

11. Dato un elenco di prodotti con descrizione e prezzo, calcolare l'incremento del prezzo secondo una percentuale fornita da tastiera. Comunicare per ciascun prodotto la descrizione e il prezzo incrementato.

12. Calcolare il consumo medio di carburante di un elenco di veicoli utilizzando come dati di input i valori dei km percorsi e dei litri di carburante per ciascun veicolo.

Struttura di ripetizione con contatore

13. Costruire un programma che accetta da tastiera un elenco di N prezzi e li somma. Alla fine fornisce il totale dei prezzi inseriti.

14. Dati i risultati in un test di una classe di 20 studenti, con nome e giudizio rappresentato con un valore numerico (1= sufficiente, 2 = buono, 3 = distinto, 4 = ottimo), calcolare la media dei giudizi usando i valori come peso.

Struttura di scelta multipla

15. Dato un elenco di numeri compresi tra 1 e 7, visualizzare il giorno della settimana corrispondente a ciascuno, usando la seguente decodifica: 1 = lunedì, 2 = martedì, …, 7 = domenica.

16. Un grande magazzino ha 4 reparti, rappresentati con i numeri 1, 2, 3, 4. La Direzione decide di applicare sui prodotti di ciascun reparto sconti con percentuali differenziate e precisamente: reparto 1 = sconto 0% (nessuno sconto), reparto 2 = sconto 3%, reparto 3 = sconto 2%, reparto 4 = sconto 5%. Dati N prodotti con reparto di appartenza e prezzo, calcolare e visualizzare per ciascuno il prezzo scontato.

12 Lo sviluppo top-down e le funzioni

I programmi visti negli esempi precedenti riguardavano problemi relativamente semplici. Quando il livello di complessità aumenta, conviene suddividere il programma in sottoprogrammi secondo la metodologia di **sviluppo top-down**.
Si tratta in sostanza di definire quali siano le parti fondamentali di cui si compone il programma (*top*), e procedere poi al dettaglio di ogni singola parte (*down*). Ogni parte, che compone il programma, corrisponde ad un modulo che svolge una specifica funzionalità per la risoluzione del problema, cioè il programma viene scomposto in **moduli funzionalmente indipendenti**.
Viene facilitato il lavoro di **manutenzione del software**, perché si può intervenire con modifiche o correzioni su un solo modulo, avendo nel contempo cognizione di quello che fa l'intero programma. Inoltre alcuni moduli, essendo funzionalmente indipendenti, possono essere riutilizzati, senza bisogno di grandi modifiche, all'interno di altri programmi.

Nel linguaggio C i sottoprogrammi sono rappresentati attraverso le **funzioni**.

Nei programmi presentati precedentemente sono già state utilizzate le **funzioni**, anche se sono state identificate come istruzioni del linguaggio C. L'istruzione *scanf* è in realtà una funzione che serve a ricevere i dati che l'utente introduce da tastiera; l'istruzione *printf* è una funzione che si occupa di mandare sul video i caratteri corrispondenti ai dati o ai messaggi di output. Inoltre abbiamo utilizzato funzioni di calcolo, quali la funzione *sqrt* per la radice quadrata di un numero. Sono tutti esempi di funzioni predefinite nel linguaggio C o, come si dice nel linguaggio informatico, **funzioni built-in**, cioè moduli software che il programmatore può usare senza implementarli. Questi moduli vengono utilizzati indicandone semplicemente il nome.

La **dichiarazione delle funzioni** che si intendono utilizzare nel programma va posta in testa al programma, immediatamente dopo la sezione dedicata alle direttive e alla dichiarazioni delle variabili del programma.
La sintassi generale per la dichiarazione di una funzione è:

tipo di dato restituito **nome della funzione** *(elenco dei parametri)* {
……..;
……..;
……..;
return *valore restituito;*
}

Poiché la funzione restituisce un valore, occorre specificare, prima del nome che identifica la funzione, il **tipo** del valore restituito. Dopo il nome della funzione, le parentesi tonde servono a contenere l'elenco degli argomenti passati alla funzione, detti **parametri**. Le istruzioni che formano la funzione sono delimitate dalle parentesi graffe e rappresentano il codice che viene eseguito alla chiamata della funzione.

Il corpo della funzione contiene come ultima istruzione, la parola **return** seguita dal valore o dalla variabile contenente il risultato restituito dalla funzione. L'istruzione *return* provoca il ritorno del controllo alla funzione principale o, in generale, alla funzione chiamante.

Se il tipo restituito dalla funzione non è specificato, si assume, in mancanza (per *default*), che il tipo sia **int.**
Se si vuole invece che la funzione non restituisca alcun valore, bisogna specificare, come tipo di dato restituito, il tipo **void** (in italiano *vuoto* o *privo*):

void nome della funzione (elenco dei parametri)

L'elenco dei parametri segue la sintassi della dichiarazione delle variabili, vista negli esempi precedenti; i parametri sono separati dalla virgola e per ciascun parametro deve essere indicato il tipo (diversamente dalla dichiarazione delle variabili):

tipo nome della funzione (*tipo1* nome1, *tipo2* nome2, …)

Il tipo *void* può essere usato anche nella dichiarazione dei parametri: il tipo *void* scritto tra parentesi tonde indica che nessun parametro viene passato alla funzione. Questo è il valore di *default;* infatti:

void f (**void**)

si può anche scrivere:

void f ( )

L'esempio seguente mostra la codifica di una funzione che esegue la somma di due numeri interi, ricevuti come parametri, e restituisce il valore calcolato.

```
int Somma (int a, int b){
   int x;

   x = a + b;
   return x;
}
```

Quando la funzione ha, come tipo di ritorno, il tipo *void*, l'istruzione *return* non è seguita da alcun valore o espressione, in quanto tale tipo indica proprio il fatto che la funzione non restituisce nulla alla funzione chiamante; in questo caso l'istruzione *return*, in fondo all'implementazione, può essere omessa.
L'esempio seguente mostra la codifica di una funzione che esegue la somma di due numeri e visualizza il risultato, senza restituire nulla alla funzione chiamante.

```
void StampaSomma (int a, int b){
   int x;

   x = a + b;
   printf("Il totale e': %d \n", x);
}
```

La seguente funzione di stampa è un esempio di funzione che non restituisce alcun valore e che non riceve alcun parametro come argomento: il tipo restituito è *void* e il tipo *void* scritto tra parentesi tonde indica che nessun parametro viene passato alla funzione.

```
void Stampa (void){
   printf("stampa di prova \n");
   printf("fine della funzione \n");
}
```

Vediamo ora come si possa effettuare la **chiamata di una funzione** da un qualunque punto del programma, dopo averla dichiarata e definita: occorre specificare il nome della funzione seguito dall'elenco, tra parentesi tonde, dei valori da assegnare ai parametri della funzione:

nome della funzione (*elenco dei valori da passare ai parametri*);

Al momento della chiamata, il compilatore esegue il controllo di corrispondenza tra i parametri specificati nella definizione della funzione e i valori passati alla funzione.
Un esempio di chiamata della funzione *Somma* definita in precedenza è il seguente:

```
int totale;
int subtot1;
int subtot2;
...
totale = Somma(subtot1, subtot2);
...
```

Il **main** è a tutti gli effetti una funzione: questo è il motivo della presenza della coppia di parentesi tonde dopo la parola *main*, come visto negli esempi precedenti. Tuttavia la funzione *main* ha una caratteristica specifica: è la funzione che viene eseguita per prima all'avvio del programma. L'istruzione *return* può essere presente anche nella funzione *main*, in tal caso il valore viene restituito direttamente al sistema operativo, che è il programma chiamante della funzione.
Per completezza quindi, negli esempi successivi, sarà indicata la parola **int** prima di *main*, per indicare il tipo del valore restituito, e alla fine del *main* verrà scritta l'istruzione **return** seguita dal valore **0**.
L'uso di *return* con il valore 0 è il modo più comune per indicare la terminazione di un programma che non ha trovato errori durante l'esecuzione.

ESEMPIO: Calcolo delle soluzioni di un'equazione di secondo grado

Un'equazione di secondo grado scritta nella forma $ax^2 + bx + c = 0$ è caratterizzata dai coefficienti a,b,c. Dopo aver calcolato il discriminante Δ (delta) con la formula b * b – 4 * a * c, si possono riconoscere tre situazioni:

$\Delta < 0$ non esistono soluzioni reali,
$\Delta = 0$ le due soluzioni reali sono coincidenti,
$\Delta > 0$ ci sono due soluzioni reali, distinte.

Se esistono soluzioni reali, queste si ottengono dalla formula $(-b \pm \sqrt{\Delta})/(2 * a)$.
Perché il programma comprenda tutte le situazioni possibili, occorre prevedere anche che sia uguale a zero. Infatti, se a = 0 si deve procedere alla soluzione dell'equazione di primo grado:

$bx + c = 0$.

In questa eventualità bisogna poi riconoscere i sottocasi:

b = 0 e c = 0 equazione indeterminata,
b = 0 equazione impossibile.

Da questa analisi sono state individuate tre sottofunzioni richiamate dal *main*:

- calcolo di *delta*
- risoluzione dell'equazione di primo grado nel caso *a = 0*
- visualizzazione delle soluzioni quando *delta* è maggiore o uguale a 0.

PROGRAMMA C

```
/* Equazioni.c : soluzioni di un'equazione di secondo grado */
#include <stdio.h>
#include <math.h>

  /* input */
  float a, b, c;      /* coefficienti equazione  */
  /* output */
  float x1, x2;
  /* lavoro */
  float delta;        /* discriminante */

void RisolviPrimoGrado() {

  if ((b==0) && (c==0)) {
     printf("Equazione indeterminata \n");
  }
  else {
    if (b==0) printf("Equazione impossibile \n");
    else printf("x = %8.2f \n", -c/b);
  }
  return;
} /* RisolviPrimoGrado */

float CalcolaDelta() {

   delta = b*b - 4*a*c;
   return delta;
} /* CalcolaDelta */

void ScriviSoluzioni() {

   if (delta < 0) {
      printf("Non esistono soluzioni reali \n");
   }
   else {
      x1 = (-b-sqrt(delta)) / (2*a);
      x2 = (-b+sqrt(delta)) / (2*a);
      printf("x1 = %8.2f \n", x1);
      printf("x2 = %8.2f \n", x2);
   }
   return;
} /* ScriviSoluzioni */
```

```
/* funzione principale */
int main(void) {

  printf("Tre coefficienti: ");
  scanf("%f %f %f", &a, &b, &c);
  if (a != 0) {
     CalcolaDelta();
     ScriviSoluzioni();
  }
  else {
     RisolviPrimoGrado();
  }
  return 0;
}
```

In questo esempio tutte le funzioni sono senza parametri. La funzione *CalcolaDelta* restituisce un valore di tipo *float*, le altre non restituiscono alcun valore e quindi sono di tipo *void*.

Nel programma inoltre le variabili sono dichiarate all'inizio, dopo le direttive e al di fuori del *main*, perché esse sono utilizzate anche dalle funzioni: le variabili dichiarate all'esterno del *main* e delle funzioni si chiamano **variabili globali**, in contrapposizione a quelle dichiarate all'interno delle funzioni o all'interno del *main*, che prendono il nome di **variabili locali**.

Si noti infine l'uso della direttiva *#include <math.h>*, perché il problema richiede il calcolo della radice quadrata, realizzato dalla funzione predefinita **sqrt**.

Per rendere più efficace la lettura e l'interpretazione del codice, può essere utile ripetere, dopo la parentesi graffa che chiude il corpo della funzione, il nome della funzione stessa come commento (racchiuso tra i delimitatori /* … */): in questo modo appare più evidente dove inizia e dove termina la funzione.

13 Funzioni con parametri

L'uso delle funzioni risponde all'esigenza fondamentale di costruire programmi ben organizzati e strutturati secondo la metodologia top-down.

Un'altra esigenza che viene risolta dalle funzioni è rappresentata dalla possibilità di poter utilizzare uno stesso gruppo di istruzioni in programmi diversi, proprio come accade con le funzioni predefinite. In questo caso conviene rendere parametrici i valori utilizzati dalle funzioni, in modo da ricevere come argomenti i valori passati dalla funzione *main* o da un'altra funzione chiamante.

ESEMPIO: Calcolo delle soluzioni di un'equazione di secondo grado

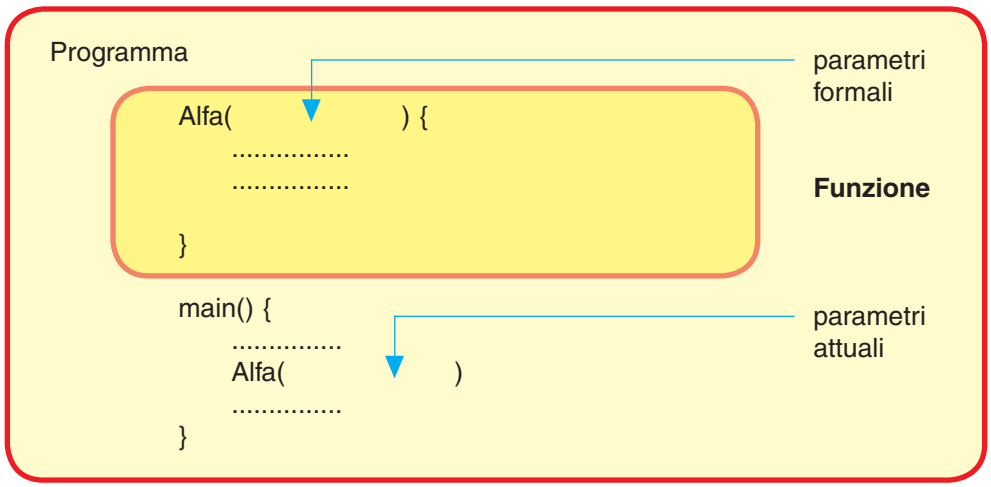
Viene ripreso l'esempio del paragrafo precedente per il calcolo delle soluzioni di un'equazione di secondo grado, introducendo i parametri nelle funzioni.

```
/* Equazioni2.c : soluzioni di un'equazione di secondo grado */
#include <stdio.h>
#include <math.h>
```

```
void RisolviPrimoGrado(float c2, float c3) {

  if ((c2==0) && (c3==0)) {
     printf("Equazione indeterminata \n");
  }
  else {
    if (c2==0) printf("Equazione impossibile \n");
    else printf("x = %8.2f \n", -c3/c2);
  }
  return;
} /* RisolviPrimoGrado */

float CalcolaDelta(float c1, float c2, float c3) {

   return c2*c2 - 4*c1*c3;

} /* CalcolaDelta */

void ScriviSoluzioni(float c1, float c2, float c3, float d) {
   float x1, x2;

   if (d < 0) {
      printf("Non esistono soluzioni reali \n");
   }
   else {
      x1 = (-c2-sqrt(d)) / (2*c1);
      x2 = (-c2+sqrt(d)) / (2*c1);
      printf("x1 = %8.2f \n", x1);
      printf("x2 = %8.2f \n", x2);
   }
   return;
} /* ScriviSoluzioni */

/* funzione principale */
int main(void) {

  float a, b, c;      /* coefficienti equazione  */
  float delta;        /* discriminante */

  printf("Tre coefficienti: ");
  scanf("%f %f %f", &a, &b, &c);
  if (a != 0) {
     delta = CalcolaDelta(a, b, c);
     ScriviSoluzioni(a, b, c, delta);
  }
  else {
     RisolviPrimoGrado(b, c);
  }
  return 0;
}
```

In questa versione del programma *Equazioni*, le variabili *a*, *b*, *c*, *delta* sono variabili interne al *main*, o come si dice più correttamente, **variabili locali** del *main*, mentre le funzioni operano sui parametri. Le variabili *x1* e *x2*, che sono utilizzate solo all'interno della funzione per il calcolo delle soluzioni, sono **variabili locali** della funzione stessa.
Il vantaggio di questa modalità d'uso delle variabili consiste nel fatto che le funzioni così strutturate diventano moduli software che possono essere utilizzati trasportando senza modifiche il loro codice (con un semplice *copia e incolla*) in altri programmi che richiedano le stesse funzionalità.

L'operazione, con la quale il *main* (o un'altra funzione chiamante) invia i valori alla funzione, assegnandoli ai parametri, si chiama **passaggio di parametri**.
Le variabili indicate nell'intestazione della funzione (nell'esempio, le variabili *c1*, *c2*, *c3*, *d*) si chiamano **parametri formali**; le variabili che forniscono i valori ai parametri si chiamano **parametri attuali**.



14 Il passaggio di parametri

L'operazione, con la quale vengono associate le due liste dei parametri attuali e formali, detta *passaggio dei parametri,* può avvenire in due modi diversi:

- **passaggio di parametri per valore***,* quando i valori delle variabili della funzione principale vengono ricopiati nei parametri della funzione; i cambiamenti effettuati sui parametri formali durante l'esecuzione della funzione non influenzano i valori delle variabili nella funzione chiamante (*main* o altra funzione).
- **passaggio di parametri per referenza** (o **per indirizzo**), quando i parametri attuali e formali fanno riferimento alla stessa cella di memoria centrale, cioè allo stesso **indirizzo** di memoria; questo è il caso in cui il programmatore vuole che i cambiamenti di valore ai parametri, durante l'esecuzione della funzione, influenzino i valori delle variabili corrispondenti nella funzione chiamante.

Nel linguaggio C il passaggio di parametri per valore avviene indicando nell'intestazione della funzione il tipo e il nome di ciascun parametro formale.

Il programma, presentato in precedenza per il calcolo delle soluzioni di un'equazione di secondo grado fornisce alcuni esempi di funzioni che utilizzano il passaggio di parametri *per valore*.
Consideriamo ora il caso di un problema nel quale è necessario usare il passaggio di parametri *per referenza*.

ESEMPIO: Ordinamento crescente di tre numeri

Per disporre in ordine crescente tre numeri, si può procedere nel seguente modo: si confronta il primo numero con il secondo, scambiandoli tra loro se non sono in ordine crescente; successivamente si confronta il primo con il terzo, scambiandoli tra loro se non sono in ordine crescente, e infine il secondo con il terzo, scambiandoli tra loro se non sono in ordine crescente. Si noti che nel programma viene (volutamente) ripetuta per tre volte la struttura *if.*

PROGRAMMA C

```
/* TreNumeri.c : ordinamento crescente di tre numeri */

#include <stdio.h>

int main(void) {

  /* input-output */
  int a, b, c;
  /* lavoro */
  int temp;

  printf("Tre numeri: ");
  scanf("%d %d %d", &a, &b, &c);
  if (a > b) {
     temp = a;
     a = b;
     b = temp;
  }
  if (a > c) {
     temp = a;
     a = c;
     c = temp;
  }
  if (b > c) {
     temp = b;
     b = c;
     c = temp;
  }
  printf("Numeri ordinati: \n");
  printf("%d \n", a);
  printf("%d \n", b);
  printf("%d \n", c);

  return 0;
}
```

In questo programma l'operazione di controllo e scambio del contenuto di due variabili viene eseguita più volte nel corso del programma su variabili diverse, ma usando le stesse istruzioni. Queste istruzioni, che vengono usate tre volte, in un programma meglio organizzato possono essere convenientemente raggruppate in una sola funzione *Ordina*, che verrà richiamata dal *main* ad operare tre volte con variabili diverse.
Quindi l'uso della funzione risponde all'esigenza di poter utilizzare più volte uno stesso gruppo di istruzioni in momenti diversi del programma, e magari con dati diversi.

Vediamo come si realizza nel linguaggio C il passaggio di parametri per referenza, in modo che le modifiche apportate all'interno della funzione permangano anche dopo il ritorno al programma chiamante. Per far questo è necessario che la funzione non operi su una copia locale dei dati, ma direttamente sui dati originali.
Per attivare questo tipo di comportamento è necessario passare, anziché la variabile direttamente, il suo **indirizzo**, in modo che la funzione veda la locazione di memoria dove è contenuta la variabile e possa quindi modificarla.

Per passare l'indirizzo di una variabile è necessario utilizzare l'**operatore &**, che restituisce appunto l'indirizzo della variabile alla quale è applicato.

D'altra parte la funzione, che riceve una variabile con passaggio per indirizzo deve essere in grado di risalire, dall'indirizzo della cella di memoria, al valore in essa contenuto, tramite l'**operatore *** che, applicato ad una variabile di tipo indirizzo, ne restituisce il valore.

Nella scrittura del codice sia il carattere &, sia il carattere *, precedono il nome della variabile alla quale sono applicati, come si vede nel codice seguente.
Infatti la funzione *Ordina*, per lo scambio di due numeri che non sono in ordine crescente, fornisce un esempio di passaggio di parametri per referenza. In questo caso si vuole che lo scambio, effettuato sui parametri formali *x* e *y*, provochi lo scambio sulle corrispondenti variabili passate dal *main* nella chiamata alla funzione.

PROGRAMMA C

```
/* TreNumeri2.c : ordinamento crescente di tre numeri */

#include <stdio.h>

void Ordina (int *x, int *y) {
  int temp;

  if (*x > *y) {
    temp = *x;
    *x = *y;
    *y = temp;
  }
  return;
}

/* funzione principale */
int main(void) {

  /* input-output */
  int a, b, c;

  printf("Tre numeri: ");
  scanf("%d %d %d", &a, &b, &c);

  Ordina(&a, &b);
  Ordina(&a, &c);
  Ordina(&b, &c);

  printf("Numeri ordinati: \n");
  printf("%d \n", a);
  printf("%d \n", b);
  printf("%d \n", c);

  return 0;
}
```

La funzione principale chiama tre volte la funzione *Ordina*, chiedendo di operare su tre diverse coppie di dati. Ogni chiamata causa l'esecuzione della funzione, al termine della quale la coppia di dati forniti risulta ordinata.
I parametri attuali, passati dal *main* alla funzione, sono gli indirizzi delle variabili. I parametri formali, sui quali opera la funzione per fare il confronto e l'eventuale scambio di valore, utilizzano l'operatore * per ottenere il valore corrispondente dell'indirizzo di memoria.
L'eventuale scambio, effettuato sui parametri formali **x* e **y*, provoca lo scambio sulle corrispondenti variabili identificate dagli indirizzi.

La chiamata della funzione nel *main* contiene, tra parentesi dopo il nome della funzione, gli indirizzi delle variabili che mandano i valori ai parametri:

```
Ordina(&a, &b);
```

I due esempi seguenti possono chiarire meglio la differenza tra passaggio per valore e passaggio per indirizzo.

Nel primo programma il passaggio del parametro è *per valore*:

```
/* Parametri1.c : per valore */

#include <stdio.h>

void Modifica (int y) {
   y += 2;
}

int main(void) {
  int x;

  x = 1;
  printf("%d \n", x);
  Modifica(x);
  printf("%d \n", x);

  return 0;
}
```

La variabile *x* passa il suo valore al parametro *y*: il parametro *y* viene raddoppiato all'interno della funzione, ma l'operazione *y += 2* non influenza il valore originale di *x*. Sul video quindi compariranno i numeri 1 e 1, in corrispondenza delle due istruzioni *printf*.

Nel secondo programma il passaggio è *per referenza* (o indirizzo):

```
/* Parametri2.c : per indirizzo */

#include <stdio.h>

void Modifica (int *y) {
   *y += 2;
}

int main(void) {
  int x;
```

```
  x = 1;
  printf("%d \n", x);
  Modifica(&x);
  printf("%d \n", x);

  return 0;
}
```

La variabile *x* passa il suo indirizzo al puntatore *y*: il valore puntato da *y* viene incrementato all'interno della funzione, e quello che succede alla variabile, il cui indirizzo è *y,* è ciò che succede alla *x*, perché *x* e *y* fanno riferimento allo stesso indirizzo di memoria centrale. Si dice infatti che *y* **punta alla memoria** di *x*. Sul video quindi compaiono i numeri 1 e 3, in corrispondenza delle due istruzioni *printf*.

Riassumendo, il passaggio per refenza si usa quando l'elaborazione effettuata all'interno della funzione deve produrre effetti alle variabili passate, necessari nelle successive elaborazioni del *main* (o della funzione chiamante). Si usa il passaggio per valore quando la funzione corrisponde ad un modulo che per svolgere il suo lavoro ha bisogno dei valori forniti dalla funzione chiamante, per esempio funzioni che devono stampare intestazioni o righe di totali.

Per rafforzare la differenza tra parametri passati per indirizzo e per valore, vediamo anche il seguente esempio.

```
/* Incrementa.c : per valore e per indirizzo */

#include <stdio.h>

void Aggiungi (int x, int *y) {
   x++;
   (*y)++;
   printf("Nella funzione chiamata: %d %d \n", x, *y);
   return;
}

int main(void) {
  int a, b;

  a = 0;
  b = 0;
  Aggiungi (a, &b);
  printf("Nella funzione principale: %d %d \n", a, b);

  return 0;
}
```

Nel programma presentato il valore di *a* viene passato al parametro *x* per valore e l'indirizzo di *b* viene passato al parametro **y* per referenza. Per controllare il diverso comportamento dei parametri, i valori di *x* e **y* vengono visualizzati all'interno della funzione, mentre i valori di *a* e di *b* all'interno del *main* dopo che è stata eseguita la funzione *Aggiungi*.
Al termine dell'esecuzione, in output compaiono nell'ordine le coppie di numeri:

```
1   1
0   1
```

perché l'incremento di *x* (parametro per valore) all'interno della funzione non influenza il valore di *a*, mentre l'incremento della variabile indirizzata da *y* (parametro per referenza) modifica il valore iniziale di *b*.

15 Dichiarazione delle funzioni con i prototipi

Le funzioni di un programma C possono comparire in qualsiasi ordine nel file sorgente. Le funzioni devono però essere definite prima che esse vengano utilizzate dal *main* o da altre funzioni, come è stato fatto negli esempi precedenti.
L'ordine in cui le funzioni sono definite è quindi importante: infatti una funzione non ne può richiamare un'altra se questa non è stata ancora definita.
Il programma sorgente inoltre può essere suddiviso in più file con estensione *.c* contenenti diverse dichiarazioni di variabili e di funzioni. In fase di compilazione dovranno poi essere specificati tutti i file sorgente che devono essere tradotti nel programma oggetto.
Per queste ragioni, e per una migliore organizzazione dei programmi, è opportuno inserire le funzioni nel programma C attraverso due fasi distinte tra loro:

- dichiarazione delle funzioni per mezzo dei loro **prototipi**;
- definizione (o *implementazione*) delle funzioni.

La **dichiarazione della funzione**, che si intende utilizzare nel programma, di norma va posta in testa al programma, immediatamente dopo la sezione dedicata alle direttive e alla dichiarazione delle variabili globali del programma. Tramite la dichiarazione il programmatore si limita a comunicare al compilatore la sua intenzione di utilizzare una funzione che restituisce un dato di un certo tipo e che utilizza determinati parametri, riservandosi di specificarne il codice in seguito, nella fase di definizione. Questa dichiarazione si chiama **prototipo** della funzione.
Tutte le funzioni devono essere dichiarate, fatta eccezione per la funzione *main* che non necessita di dichiarazione, in quanto, essendo la prima funzione ad essere eseguita, non richiede controlli alla chiamata.
La sintassi generale per la dichiarazione di una funzione è:

tipo di dato restituito nome della funzione (*elenco dei parametri*);

La **definizione** (o **implementazione**) **della funzione**, con le istruzioni che la compongono, viene normalmente posta dopo la funzione *main*.

ESEMPIO: Scrivere il programma per calcolare l'area e il perimetro di un quadrato, fornita da tastiera la misura del lato

Il programma può essere scomposto in tre parti fondamentali:

- inserimento da tastiera della lunghezza del lato
- calcolo dell'area
- calcolo del perimetro.

A queste tre parti corrispondono tre funzioni che vengono codificate in C nel seguente modo:

```
int Tastiera(void) {
  int misura;
  printf("Introduci il lato: ");
  scanf("%d", &misura);
  return misura;
}

void CalcoloArea (int l) {
  float area;
  area = l * l;
  printf("%10.2f \n", area);
  return;
}
```

```
void CalcoloPerimetro (int l) {
  float perim;
  perim = l * 4;
  printf("%10.2f \n", perim);
  return;
}
```

Le variabili *misura*, *area* e *perim* vengono usate solo all'interno delle rispettive funzioni e sono quindi variabili **locali**.

Il programma completo può quindi essere organizzato in questo modo:

```
/* Quadrato.c : perimetro e area del quadrato */
#include <stdio.h>

int Tastiera(void);
void CalcoloArea (int l);
void CalcoloPerimetro (int l);

/* funzione principale */
int main(void) {

  int lato;

  lato = Tastiera();
  CalcoloArea(lato);
  CalcoloPerimetro(lato);

  return 0;
}

int Tastiera(void) {
  int misura;      /* variabile locale */
  printf("Introduci il lato: ");
  scanf("%d", &misura);
  return misura;
}

void CalcoloArea (int l) {
  float area;      /* variabile locale */
  area = l * l;
  printf("%10.2f \n", area);
  return;
}

void CalcoloPerimetro (int l) {
  float perim;     /* variabile locale */
  perim = l * 4;
  printf("%10.2f \n", perim);
  return;
}
```

All'inizio del programma le funzioni vengono soltanto dichiarate, specificandone per ciascuna il tipo restituito, il nome e i parametri richiesti. In questo modo si ottiene una semplice elencazione dei prototipi che facilita la comprensione del programma e realizza in pratica la metodologia top-down.

Se si vuole poi vedere in dettaglio le modalità di elaborazione delle singole funzioni, si può analizzare la definizione delle funzioni scritta dopo la funzione *main*.
Come abbiamo già visto in precedenza, nelle direttive *#include* sono specificati i file di intestazione (file con estensione *.h*) che contengono i prototipi delle funzioni predefinite del linguaggio. Si osservi che nella dichiarazione dei prototipi delle funzioni, sono indicati per ogni parametro il tipo e il nome, per rendere più comprensibile la lettura del codice. In effetti il linguaggio C consente la dichiarazione delle funzioni con i prototipi indicando solo il tipo, senza il nome, dei parametri.

ESEMPIO: Scrivere un programma per acquisire da tastiera un numero e comunicare all'esterno un messaggio che indichi se il numero è pari o dispari

La funzione *Pari* riceve come parametro il numero e calcola il resto della divisione per 2. Se il resto è 0, restituisce il valore 1 (Vero), altrimenti ritorna il valore 0 (Falso).

PROGRAMMA C

```
/* PariDispari.c : numero pari o dispari */
#include <stdio.h>

int Pari(int x);

/* funzione principale */
int main(void) {

  int y;

  printf("un numero: ");
  scanf("%d", &y);
  if (Pari(y)) printf("%d numero pari \n", y);
  else printf("%d numero dispari \n", y);

  return 0;
}

int Pari(int x) {
  int b;
  if (x%2 == 0) b = 1;
  else b = 0;
  return b;
}
```

La scrittura

```
   if (Pari(y)) .....
```

significa

```
   if (Pari(y)==1) .....
```

cioè rappresenta un controllo sul valore logico *Vero* (=1) della funzione.

16 Le funzioni predefinite

Il linguaggio C possiede alcune **funzioni predefinite** (*built-in*), che il programmatore può usare nel programma senza dichiarazione. Queste funzioni sono raggruppate in **librerie di funzioni**, che sono l'insieme dei file contenenti i prototipi delle funzioni. Le funzioni di libreria sono richiamabili dai programmi C, inserendo all'inizio del programma la direttiva **#include** che specifica il file di intestazione che contiene la libreria stessa (file con estensione **.h**).

Esempi di file di inclusione per le librerie, che sono già stati utilizzati nei programmi precedenti, sono:

- **stdio.h** per le funzioni standard di input/output
- **math.h** per le funzioni matematiche.

Per esempio la funzione *sqrt(x)* calcola il quadrato di un numero, mentre la funzione *fabs(x)* restituisce il valore assoluto di un numero in virgola mobile (*float*).
Se la funzione *fabs* non fosse già predefinita nel linguaggio, il programmatore la dovrebbe creare, dichiarandola e usandola nel programma come nell'esempio seguente.

ESEMPIO: Scrivere un programma che calcoli il valore assoluto di un numero reale

La funzione per il calcolo del valore assoluto ritorna l'opposto del numero ricevuto come parametro, se il numero è negativo, altrimenti ritorna il numero stesso.

PROGRAMMA C

```
/* ValoreAssoluto.c : valore assoluto di un numero */
#include <stdio.h>

float fabs(float x);

/* funzione principale */
int main(void) {

  float y;

  printf("Introduci un numero: ");
  scanf("%f", &y);
  printf("%10.2f \n", fabs(y));

  return 0;
}

float fabs(float x) {
  float ValAss;
  if (x < 0) ValAss = -x;
  else ValAss = x;
  return ValAss;
}
```

Si osservi che il programma precedente con alcuni compilatori può generare un *warning* in fase di compilazione, per un conflitto di nomi tra la funzione *fabs* definita dal programmatore e la funzione predefinita *fabs* del linguaggio. Tuttavia questo errore non impedisce l'elaborazione corretta del procedimento di calcolo contenuto nella funzione implementata.

Il linguaggio C possiede anche la funzione **abs(x)** per il calcolo del valore assoluto di un numero *x* di tipo *int*. Per utilizzare questa funzione occorre includere all'inizio del programma il file di intestazione **stdlib.h**.

Il programma seguente realizza le stesse funzionalità della funzione matematica predefinita **pow** del linguaggio C.
La funzione *pow* riceve due argomenti (base e esponente) di tipo *double* e restituisce un valore di tipo *double*.

ESEMPIO: Calcolo della potenza di un numero

Si tratta di organizzare un programma che richieda in input i valori della base e dell'esponente ed esegua l'elevamento a potenza. Per realizzare l'elevamento a potenza basta moltiplicare la base per se stessa un numero di volte pari all'esponente.

PROGRAMMA C

```
/* PotInt.c : potenza di un numero */
#include <stdio.h>

double Pot(double x, double y);

/* funzione principale */
int main(void) {
  /* input */
  double base, esponente;
  /* output */
  double risultato;

  printf("Base: ");
  scanf("%lf", &base);
  printf("Esponente: ");
  scanf("%lf", &esponente);
  risultato = Pot(base, esponente);
  printf("Base = %lf \n", base);
  printf("Esponente = %lf \n", esponente);
  printf("Potenza = %lf \n", risultato);

  return 0;
}

double Pot(double x, double y) {
  double potenza = 1.0;
  double i;
  for(i=1.0; i<=y; i++) potenza *= x;
  return potenza;
}
```

Per verificare il corretto funzionamento del programma precedente si provi, per esercizio, ad eseguire il programma che calcola la potenza anche tramite la funzione **pow**; per poter utilizzare questa funzione, occorre inserire all'inizio del programma la direttiva **#include <math.h>** per rendere disponibili i prototipi delle funzioni matematiche.

```
/* PotInt2.c : potenza di un numero */
#include <stdio.h>
#include <math.h>

/* funzione principale */
int main(void) {
  double base, esponente;
  double risultato;

  printf("Base: ");
  scanf("%lf", &base);
  printf("Esponente: ");
  scanf("%lf", &esponente);
  risultato = pow(base, esponente);
  printf("Potenza = %lf \n", risultato);

  return 0;
}
```

Chiaramente la funzione predefinita *pow* è più sofisticata della funzione *Pot* implementata nel precedente programma. Quest'ultima infatti considera solo il caso in cui l'esponente è intero positivo.

Le funzioni *built-in* della libreria **math.h** più utilizzate nel linguaggio C sono:

- *pow (base, esponente)* utilizzata per l'elevamento a potenza (il primo argomento costituisce la base, il secondo l'esponente; entrambi gli argomenti devono essere di tipo *double*)
- *exp(esponente),* che eleva il numero di Nepero *(e* = 2.17..) all'esponente passato come parametro
- *sqrt(argomento)* che calcola la radice quadrata del numero passato come argomento
- *log(argomento)* che calcola il logaritmo naturale (in base *e*) dell'argomento
- *log10(argomento)* che calcola il logaritmo in base 10.

Sono disponibili anche le funzioni di manipolazione delle stringhe di caratteri nella libreria **string.h**:

- *strlen(stringa)* per calcolare il numero di caratteri della stringa (lunghezza della stringa)
- *strcpy(destinazione, sorgente)* per copiare una stringa (*sorgente*) in un'altra (*destinazione*)
- *strncpy(destinazione, sorgente, n)* per copiare nella stringa di *destinazione* i primi *n* caratteri della stringa *sorgente*
- *strcmp(stringa1, stringa2)* per operare il confronto tra due stringhe (restituisce il valore 0 se le stringhe sono uguali, un valore < 0 se *stringa1* è minore di *stringa2*, un valore > 0 se *stringa1* è maggiore di *stringa2*)
- *strcat(stringa1, stringa2)* per concatenare due stringhe: la *stringa1* è la destinazione della concatenazione e la *stringa2* è la stringa da concatenare; quindi *stringa1* deve essere in grado di contenere, oltre ai caratteri in essa presenti, anche tutti i caratteri di *stringa2*.

Rispettando le regole di visibilità, per poter utilizzare le funzioni predefinite, occorre includere i loro prototipi all'inizio del programma. Quando il programma è complesso, per rendere il codice più leggibile, è buona abitudine includere gli *header file* necessari prima di ciascuna funzione che li richiede. È infatti possibile ripetere l'inclusione delle intestazioni predefinite senza che si generi alcun conflitto o ridefinizione.

ESERCIZI

Funzioni senza parametri

1. Definire una funzione che ripeta sul video in tre righe diverse la stringa acquisita da tastiera.

2. Organizzare una funzione che acquisisca da tastiera il lato di un quadrato e ne calcoli la misura dell'area e del perimetro.

Funzioni con parametri

3. Data la parabola $y = ax^2 + bx + c$, definire tre funzioni per calcolarne i punti significativi: vertice, fuoco, intersezione con gli assi. Le tre funzioni ricevono come parametri i coefficienti a, b, c e restituiscono il valore calcolato.

4. Definire una funzione che abbia come parametro un numero intero positivo e che visualizzi la sequenza dei primi 5 numeri successivi al numero dato.

5. Scrivere una funzione che abbia come parametri il numero dei valori e la loro somma e che calcoli la media aritmetica.

6. Dati in input le età di tre persone, scrivere i dati acquisiti in ordine decrescente.

7. Scrivere la funzione che restituisca l'età di una persona, conoscendo il suo anno di nascita (si consideri l'anno attuale come costante).

8. Scrivere una funzione che abbia come parametri il prezzo di vendita e la percentuale di sconto e che restituisca il prezzo scontato.

9. Scrivere una funzione che restituisca un valore booleano Vero (1) o Falso (0), controllando se due rette, di cui si conoscono le equazioni $y = m_1 x + q_1$ e $y = m_2 x + q_2$, sono perpendicolari.

10. Data la retta $y=m x + q$ e il punto P di coordinate (x1, y1), controllare se il punto appartiene alla retta.

11. Dato un elenco di reparti con l'indicazione del nome e dell'incasso giornaliero, calcolare l'incasso medio. Definire una funzione per l'acquisizione e la somma dei dati e una funzione per il calcolo della media.

12. Dato un elenco di prodotti con descrizione e prezzo, tramite una funzione viene calcolato l'incremento del prezzo secondo una percentuale fornita da tastiera. Comunicare per ciascun prodotto la descrizione e il prezzo incrementato.

13. Dopo aver acquisito da tastiera i nomi e i voti ottenuti da due candidati in un ballottaggio, calcolare la percentuale di ciascuno rispetto alla somma dei voti e scrivere i nomi dei candidati in ordine decrescente di percentuale.

Funzioni predefinite

14. Scrivere un programma per acquisire in input le coordinate di due punti del piano cartesiano e per calcolare la distanza tra i due punti.

15. Scrivere il programma che calcola l'esponenziale, il logaritmo naturale, in base e, e il logaritmo in base 10 di un numero fornito da tastiera.

16. Scrivere il programma che acquisisca da tastiera il nome e la lingua straniera studiata da N studenti e che visualizzi il nome di quelli che studiano la lingua inglese.

17 L'array

In generale i **dati** sono le informazioni rappresentate in una forma trattabile con il computer.

In molti problemi si ha la necessità di aggregare molti dati di tipo semplice, per facilitarne la rappresentazione e rendere più veloce il loro ritrovamento.
I dati sono cioè organizzati in un insieme che prende il nome di **struttura di dati**.

L'**array** è un insieme di elementi omogenei tra loro. Con una variabile possiamo indicare solo un dato, con l'array possiamo indicare tanti dati dello stesso tipo con un solo nome collettivo di variabile: l'identificatore dell'array. Gli elementi si distinguono uno dall'altro attraverso l'indice che viene assegnato nell'array, e che viene posto accanto all'identificatore dell'array.

L'array è quindi un insieme omogeneo di dati: è un esempio di *struttura di dati*. Nella terminologia matematica al posto del termine *array* si usa il termine **vettore**.

L'array si ottiene in C aggregando variabili dello stesso tipo. Un array si definisce nella zona delle dichiarazioni nel modo seguente:

tipo NomeArray[*dimensione*];

Alla normale dichiarazione di variabile si aggiunge semplicemente, tra parentesi quadre, il numero di elementi (dimensione) che compongono l'array.

Le componenti di un array possono essere non solo numeriche, ma di uno qualsiasi dei tipi standard del linguaggio C.
Per esempio, la seguente dichiarazione crea un array di 10 coefficienti di tipo *double*:

```
double coeff[10];
```

La numerazione degli indici nel linguaggio C **inizia da 0** e quindi gli elementi dell'array sono coeff[0], coeff[1], …, coeff[9].

Nel programma la componente dell'array viene indicata dal nome dell'array, seguito dall'indice della componente, racchiuso tra due parentesi quadre.

Per esempio, si può usare un'istruzione di assegnazione:

```
coeff[i] = 7 ;
```

oppure produrre in output i valori delle componenti di un array con un ciclo *for*:

```
for (i=0; i<10; i++) {
  printf("%f \n", coeff[i]);
}
```

Se una variabile è definita di tipo array, deve essere sempre usata all'interno del programma accompagnando l'identificatore della variabile con un indice.
L'indice è solitamente una variabile di tipo *int*.

È possibile **inizializzare un array** assegnando i valori alle componenti in fase di dichiarazione, come nell'esempio seguente:

```
double numeri[3] = {2.5, 1.2, 3.0};
```

I valori sono indicati all'interno delle parentesi graffe e separati dalla virgola.

Negli esempi precedenti, sono state utilizzate variabili stringa come la seguente:

```
  char nome[30];
```

La notazione è coerente con la dichiarazione degli array, in quanto nel linguaggio C le stringhe sono considerate come un **array di caratteri**. Quindi il tipo stringa è una struttura di tipo array le cui componenti sono del tipo *char*.

Per esempio, si consideri il seguente programma:

```
#include <stdio.h>

char stringa[10]="abcdefghi";

int main(void) {
  int i;

   for(i=0;i<9;i++) {
    printf("Stringa[%d] = %c \n",i, stringa[i]);
   }
  return 0;
}
```

Esso produce in output il valore di ogni componente dell'array *stringa*:

```
Stringa[0] = a
Stringa[1] = b
Stringa[2] = c
Stringa[3] = d
Stringa[4] = e
Stringa[5] = f
Stringa[6] = g
Stringa[7] = h
Stringa[8] = i
```

Come si vede dal programma, l'array ha dimensione 10, ma la stringa è formata da 9 caratteri (dall'indice 0 all'indice 8). Infatti il compilatore, nella rappresentazione interna, aggiunge automaticamente al termine di una stringa costante il carattere **Nul** del codice ASCII che corrisponde alla sequenza di escape **\0** (carattere **terminatore di stringa**), in modo che il programma possa trovarne la fine.
Pertanto la dimensione effettiva dell'array è superiore di una unità al numero di caratteri che formano la stringa.

Un'ultima considerazione sugli array riguarda il **passaggio di parametri** alle funzioni. Quando un array viene passato come parametro ad una funzione, in realtà viene passata la locazione (cioè l'*indirizzo*) della prima componente dell'array. All'interno della funzione, il parametro diventa una variabile come le altre: il nome dell'array è a tutti gli effetti una variabile contenente un indirizzo, cioè un **puntatore**. Di conseguenza quando occorre effettuare il passaggio per referenza di un array alla funzione, basta indicare il nome dell'array. Si deve anche osservare che il passaggio per referenza è il solo modo di passare un array come parametro.

ESEMPIO: Acquisire da tastiera due array e sommare le loro componenti

L'esercizio si divide in due parti. Nella prima parte si deve affrontare il problema di acquisire da tastiera gli elementi di ciascun vettore.
Per memorizzare tali elementi si utilizza una struttura di tipo array.
La seconda parte è relativa alla somma degli elementi dei vettori: per ottenere il risultato è necessario accumulare il valore di ciascun componente del vettore.

PROGRAMMA C

```
/* Somme.c: somma le componenti di due array */
#include <stdio.h>
#define MAX 100

/* prototipi delle funzioni */
int ChiediDimensione(void);
void CaricaVettore(int v[], int d);
int Somma (int v[], int d);

/* funzione principale */
int main(void) {
  int n;
  int v1[MAX], v2[MAX];
  int tot1, tot2;

  n = ChiediDimensione();
  printf("Carica gli elementi del primo vettore \n");
  CaricaVettore(v1,n);
  printf("Carica gli elementi del secondo vettore \n");
  CaricaVettore(v2,n);
  tot1 = Somma(v1,n);
  tot2 = Somma(v2,n);
  printf("Somma del primo vettore = %d \n",tot1);
  printf("Somma del secondo vettore = %d \n",tot2);

  return 0;
}

/* dimensione dell'array */
int ChiediDimensione(void) {
  int d;

  do {
    printf("Dimensione degli array: ");
    scanf("%d", &d);
  } while (d<1 || d>MAX);

  return d;
} /* ChiediDimensione */

/* caricamento delle componenti */
void CaricaVettore(int v[], int d) {
  int i;

  for (i=0; i<d; i++) {
    printf("Elemento di posto %d: ", i);
    scanf("%d", &v[i]);
  }
```

```
  return;
} /* CaricaVettore */

/* somma delle componenti */
int Somma (int v[], int d) {
  int i;
  int s = 0;

  for(i=0;i<d;i++) s+=v[i];

  return s;
} /* Somma */
```

Nel programma precedente si può osservare che la specificazione della dimensione degli array *v*, che sono parametri formali delle funzioni, può essere tralasciata.

18 L'array a due dimensioni

La **matrice** (o **array a due dimensioni**) è un insieme di elementi dello stesso tipo che sono in corrispondenza biunivoca con un insieme di coppie ordinate di numeri interi, che rappresentano rispettivamente il numero della riga e il numero della colonna della matrice.

Le matrici in linguaggio C vengono definite in modo analogo agli array:

tipo NomeMatrice[N][M];

indicando il tipo delle sue componenti, il nome della matrice, il numero N di righe, il numero M di colonne.

Se la matrice è quadrata la definizione può essere fatta in questo modo:

```
#define MAX 20
int tabella[MAX][MAX];
```

Per esempio, per organizzare gli incassi nei 20 reparti di un grande magazzino nei 6 giorni della settimana, la struttura dati più adatta è una matrice di 20 righe e 6 colonne, di numeri *float*, che viene dichiarata in C con il seguente frammento di codice:

```
#define NR 20
#define NG 6
float incassi[NR][NG];
```

Se si vuole calcolare l'incasso totale di tutti i reparti nel quarto giorno della settimana (giorno di indice 3), occorre utilizzare una struttura *for* all'interno del seguente segmento di programma:

```
float somma = 0;

for (i=0; i<NR; i++) {
    somma += incassi[i][3];
}
printf("%f \n", somma);
```

La struttura dati più adatta ad organizzare le temperature massime registrate in 50 città nei 31 giorni del mese è una matrice di 50 righe e 31 colonne:

```
#define NCIT 50
#define NGG 31
int temper[NCIT][NGG];
```

La temperatura registrata nel giorno 18 nella 27esima città si ottiene con la seguente istruzione di output, ricordando che gli indici partono da 0:

```
printf("%d \n", temper[26][17]);
```

ESEMPIO: Calcolare i totali di riga e di colonna degli elementi di una matrice

Il problema va scomposto in due parti: la prima riguarda il caricamento dei dati in memoria, la seconda è relativa ai calcoli e alla scrittura dei risultati.

La struttura dati definita è un *array a due dimensioni*.

Per il caricamento dei dati si consideri ogni riga della matrice come un vettore e si proceda al caricamento di un array attraverso una ripetizione con contatore che va da 0 al numero delle colonne – 1. Questa ripetizione viene annidata in una ripetizione più esterna che permette di ripetere l'operazione di acquisizione dei dati per tutte le righe della matrice.

Nella seconda parte relativa al calcolo dei totali di riga e di colonna, per ogni riga si azzera la variabile di accumulo, si totalizzano in questa variabile i valori della riga con una ripetizione con contatore che va da 0 al numero di colonne - 1 e alla fine si scrive il contenuto della variabile di totalizzazione. Per ottenere i totali di colonna il procedimento si sviluppa in modo analogo al precedente sostituendo il termine "righe" con il termine "colonne".

PROGRAMMA C

```
/* Totali.c: totali di riga e colonna di una matrice */
#include <stdio.h>
#define MAX 100

/* prototipi delle funzioni */
int ChiediRighe(void);
int ChiediColonne(void);
void CaricaMatrice(int m[MAX][MAX], int r, int c);

/* funzione principale */
int main(void) {

  /* input */
  int righe, colonne; /* dimensioni indicate dall'utente  */
  int mat[MAX][MAX];  /* matrice */
  /* output */
  int TotRiga;        /* totale di riga */
  int TotColonna;     /* totale di colonna */
  /* lavoro */
  int i, j;           /* contatori dei cicli for */
```

```
  righe = ChiediRighe();
  colonne = ChiediColonne();
  printf("Carica gli elementi della matrice \n");
  CaricaMatrice(mat, righe, colonne);
  printf("\n");
  for (i=0; i<righe; i++) {
    TotRiga = 0;
    for (j=0; j<colonne; j++) {
      TotRiga += mat[i][j];
    }
    printf("%7d \n",TotRiga);
  }
  printf("\n");
  for (j=0; j<colonne; j++) {
    TotColonna = 0;
    for (i=0; i<righe; i++) {
      TotColonna += mat[i][j];
    }
    printf("%5d",TotColonna);
  }
  printf("\n");
  return 0;
}

/* prima dimensione della matrice */
int ChiediRighe(void) {
  int d;
  do {
    printf("Numero di righe: ");
    scanf("%d", &d);
  } while (d<1 || d>MAX);

  return d;
} /* ChiediRighe */

/* seconda dimensione della matrice */
int ChiediColonne(void) {
  int d;
  do {
    printf("Numero di colonne: ");
    scanf("%d", &d);
  } while (d<1 || d>MAX);

  return d;
} /* ChiediColonne */
```

```
/* caricamento delle componenti */
void CaricaMatrice(int m[MAX][MAX], int r, int c) {
  int i, j;

  for (i=0; i<r; i++) {
    for (j=0; j<c; j++) {
      printf("Elemento di posto %d, %d : ", i, j);
      scanf("%d", &m[i][j]);
    }
  }

  return;
} /* CaricaMatrice */
```

ESERCIZI

Array

1. Dopo aver caricato in memoria un array di numeri interi con 10 componenti, calcolare la somma delle componenti.

2. Dopo aver caricato in memoria un array di numeri interi con 10 componenti, contare le componenti che hanno valore superiore a 5.

3. Dopo aver caricato in memoria un array di numeri interi con 10 componenti, raddoppiare il valore delle prime tre componenti.

Matrici

4. Costruire la tavola pitagorica per i numeri da 0 a 9, in pratica la matrice che ha per componenti il valore che si ottiene moltiplicando l'indice della riga per l'indice della colonna.

5. Dopo aver caricato in memoria una matrice di interi quadrata di ordine n (con n inserito da tastiera non superiore a 10), sommare gli elementi della diagonale principale.

6. I dati sugli stipendi di 5 dipendenti nei primi 3 mesi dell'anno sono organizzati in una matrice, il numero di riga indica il numero del dipendente, il numero di colonna indica il mese. Calcolare la somma degli stipendi pagati al secondo dipendente.

7. I dati sugli incassi di 3 reparti di un grande magazzino nei primi 6 mesi dell'anno sono organizzati in una matrice, il numero di riga indica il numero del reparto, il numero di colonna indica il mese. Calcolare l'incasso totale del mese di aprile.

8. I dati sulle votazioni di 10 studenti in 4 prove sono organizzati in una matrice, il numero di riga indica il numero dello studente, il numero di colonna indica il numero della prova. Calcolare la media dei voti dello studente che occupa la quinta posizione nell'elenco.